Available online at www.sciencedirect.com
ScienceDirect
Computer Physics Communications ••• (••••) •••–•••
Computer Physics Communications
www.elsevier.com/locate/cpc

The Cracow code—an interactive method of sophisticated online analysis

Jerzy Grębosz [a,b,1]

[a] *The Henryk Niewodniczański Institute of Nuclear Physics (IFJ PAN), Kraków, Poland*
[b] *Gesellschaft fuer Schwerionen Forschung (GSI), Darmstadt, Germany*

Received 3 August 2006; accepted 28 September 2006

Abstract

A crucial issue in many complex experiments is the flexibility and ease of the online data analysis. Here we present an easy-to-learn and intuitive-to-operate method of interactive online analysis for use in projectile fragmentation induced gamma-ray spectroscopy experiments at the GSI facility (the RISING experiments). With a sequence of dialogue boxes the experimenter can create a complex definition, which will produce a conditional spectrum. These definitions can be immediately applied by the online analysis, which runs in parallel as a separate program. Some problems regarding the logic of gating conditions are discussed.

PACS: 29.85.+c; 29.30.Kv

Keywords: Online analysis; Data acquisition and analysis system; Event sorting

1. Introduction

During in-beam nuclear physics experiments, which require complicated apparatus, one of the crucial issues is the quality of the online data analysis. High quality online analysis allows the right decisions to be made about possible changes to the settings of the running experiments. The RISING [1] experiments use the cluster detectors from the former EUROBALL IV germanium-detector array [2]. Although RISING is similarly a γ-ray spectroscopy apparatus, there is a fundamental difference in the online analysis used by these two instruments. In the case of EUROBALL (installed at the beam line from the VIVITRON accelerator) the experimenter was sure what kind of projectile (i.e. the 'beam') was hitting the target. This is different for the RISING experiments. RISING receives the beam from the GSI Fragment Separator FRS [3]. During typical usage the FRS separator gives rise to a 'cocktail' beam consisting of several secondary projectiles. Several types of isotope of numerous elements hit the user's final target, it is the user's responsibility to select events associated with the desired projectile. The user can do this using algorithms implemented in the online analysis program. To make this feasible, the data acquisition system (DAQ) collects not only data from the RISING germanium detectors, but also from many particle detectors in the FRS (such as: multiwire chambers, scintillators, MUSIC ionisation chamber, etc.). After using complicated algorithms, the online analysis program should finally show two-dimensional scatter plots, on which the different projectiles (isotopes) can be separated.

Unfortunately there is no single universal particle identification algorithm. Depending on the experiment there are different procedures used for selecting a desired projectile from the rest of the projectiles hitting the target/final focus. This is why experimenters need a tool which is flexible, enabling them to produce the highest resolution selection for each particular isotope.

Complicated experiments require complicated software. This implies that scientists present during the experiment are dependent on the one or few people who are able to modify the online analysis program and instantly adapt it for a specific situation.

Sometimes, when everything goes wrong during a night shift, a professor might ask: *"Could we see this spectrum—gated by time, by this isotope coming from the Fragment Sep-*

E-mail address: jerzy.grebosz@ifj.edu.pl.
[1] Present address: Institute of Nuclear Physics, Polish Academy of Sciences (IFJ PAN), ul. Radzikowskiego 152, 31-342 Cracow, Poland.

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2006.09.006

arator, and by the position of fragments on this scintillator?" Very often the answer is: *"No, because the Ph.D. student, who knows how to make it, is currently sleeping"*. This sounds like an anecdote, but is not it so?...

As a technician, I always keep in my mind the following bitter saying: "The humanists are the people who know '*what*', but do not know '*how*'. The technicians...—they would know '*how'*, but they do not know '*what for*' ". There is something wrong in this attitude. Why should it not be possible to combine these two skills? Why not introduce to the online analysis an instrument that the people who have brilliant ideas can use, so that they can check their ideas by themselves, instantly; without the necessity of reading a long manual.

This was the inspiration for the work described in this paper.

2. Online analysis

RISING is an example of experiments where the data is collected using "event by event" acquisition. Its data-acquisition system DAQ (called MBS) [4] collects events and stores information on disk, but does not provide any data viewing facilities, this is the work of the online analysis program. Therefore, the basic task of the online analysis is to extract the events (i.e. unpack them from the blocks of data) and to display one-dimensional histograms (spectra) of the raw data. By watching these (typically) hundreds of individual spectra, the experimenter can monitor the operation of the experiment's detectors and their associated electronic modules.

However, in most cases, the experimenter would like to see more. Thus, the real task of the online analysis is to take the events, coming one by one, from the data acquisition system, and then analyse them using algorithms specially dedicated to that particular experiment. These algorithms are usually carefully prepared before the experiment. By monitoring these spectra, the scientist would hope to see the first signs of a successful experiment. If they cannot see the expected results of the analysis, they may wish to alter the algorithms, in particular, algorithms used to gate other online created histograms, or they may decide to change the particle separation or beam settings. For example, they may wish to see newly defined spectra and to collect them only when some specific conditions occur.

The current article describes one way of performing the online analysis which enables the experimenter the freedom to create any kind of new one or two-dimensional spectrum, gated by sophisticated conditions, which are also user defined. This program does not require the experimenter to have significant knowledge of programming, nor does it require any modification of the online analysis program. There is no need to recompile the program or to stop the running online analysis program to update the gating conditions. All that is needed is the specially prepared Graphic User Interface (GUI) program—called 'Cracow'.

Note, that the package contains two distinct programs. The first program, 'spy' is run as a "command line" program and is responsible for making the online analysis. The second program, 'Cracow', has a graphic user interface which allows the spectra produced by spy to be displayed. It also provides the methods by which the experimenter can instruct the online analysis program to create the new, sophisticated conditional spectra.

3. 'Cracow' GUI

As shown in Fig. 1, there are two programs at work, but this article concentrates only on the 'Cracow' code and, in particular, on its methods of creating the user-defined spectra and user-defined conditions.

The main goal in the design of the 'Cracow' code was to make it sufficiently experimenter friendly that the user does not need to read a manual. Using the 'Cracow' GUI, the experimenter follows sequences of dialogue boxes—called 'creators' (or 'wizards'). By asking questions and expecting straightforward answers the wizards can create any kind of spectra, gated by many different types of conditions. The products of these

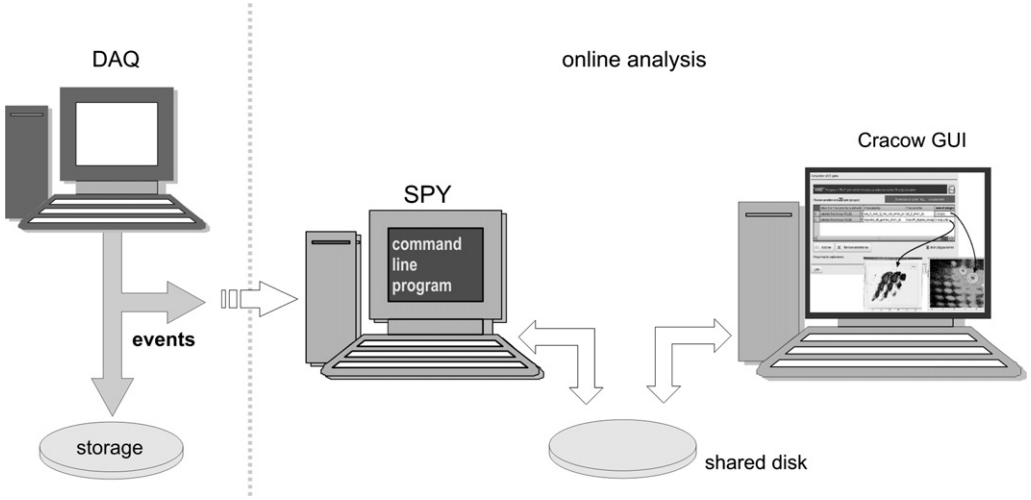

Fig. 1. Schematic of the RISING online analysis. The two separate programs 'spy' (which analyses events), and 'Cracow' (which allows on-line viewing of the spectra) communicate only by a shared disc. Using the graphic user interface of 'Cracow' it is possible to tell the 'spy' to start collecting a new kind of conditional (i.e. gated) spectrum.

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006

wizards, which are definitions of spectra or definitions of conditions, are stored on the disk as text files. (In this way they are 'persistent' and may be used in all future analysis.)

Once finished working with the wizard, 'Cracow' checks whether the online analysis program ('spy') is currently running. If yes, it is automatically informed about the new "wish" of the experimenter. The 'spy' program then:

- opens the definitions of the new spectra and the new conditions,
- creates the corresponding objects in the program,
- immediately starts to increment the new spectra with data coming from the experiment.

The experimenter using the same 'Cracow' program can then observe the result of their work, i.e. the new spectra collected using his/her newly introduced gating algorithm.

4. Incrementer

The key concept of the system described in this article is the so-called *incrementer*. An incrementer represents a variable in the online analysis program; a variable, which can be used to increment any given spectrum. Three types of variables from the RISING online analysis C++ 'spy' program can become incrementers: *int*, *double*, *bool*.

Of course not every variable in the program is worth being defined as an incrementer. This decision is made by the scientist who writes the online analysis program. If he decides that a variable may be useful as an incrementer, he defines a meaningful name which is then included in the list of the available incrementers. The GUI program reads the list of names and presents them on the screen. At any time, the experimenter can select the desired incrementer from such a list, when for example they need to create a histogram of a variable.

An example of an incrementer available in the RISING online analysis program is the energy of a γ-ray detected by one of the germanium detectors. The DAQ delivers the raw value of this data variable. Having this raw data, the online analysis program produces a gain-matched (calibrated) version and if required, a Doppler corrected energy. These three variables, raw, calibrated and Doppler corrected, can be chosen to be accessible as incrementers. As there are 105 such individual germanium detectors in the RISING array, by this action there would be $105 * 3 = 315$ incrementers available.

5. Some incrementers must be validated

Most of the users of the 'Cracow' GUI do not have to understand the concept of incrementers. They can simply treat them as variables in the analysis program. The user should however understand that in some events these 'variables' (incrementers) may contain undefined values.

Some incrementers always have a physically meaningful value, for instance, a variable which represents the multiplicity of germanium crystals which fired in a particular event. Since the online analysis program registers how many crystals fired in a given event, the incrementer representing this variable always contains a physically meaningful value.

However not all incrementers are of this nature. For example, in the RISING experiments there is a multiwire chamber which is used to trace the trajectory of projectile fragment ions. One expects that for any particular event both the 'left' and 'right' cathodes will deliver signals in the 'data'. These two data values are used by the online-analysis program to calculate the horizontal position of the ion in millimetres. The calculated value of this geometrical position is a useful incrementer, usually containing a value in the range $[-150, +150]$. Unfortunately sometimes only one cathode delivers the data. In this case it is impossible to calculate the position of the ion.

What, in such a case, should the contents of the incrementer representing the geometric position be? Zero? No, zero means that the geometric position of the ion is precisely centred in the axis of the beam line.

There must be another way to inform the rest of the analysis code (and the user of the incrementer) that in this particular event it was not possible to calculate the position (i.e. that this variable does not contain a meaningful value). For this purpose some incrementers are associated with another Boolean value which validates the current contents of the incrementer. In the case of the multiwire chamber this is a flag noting if the calculation of the position was successful or not.

The name of an incrementer which has a validator usually ends with the word "_when_...". For example:

mw41_x_**when**_ok

which means: Multiwire chamber named "mw41", its horizontal position x *when* its calculation was successful. This suffix "*when...*" informs the user about the nature of the incrementer. If the 'Cracow' GUI notices the use of an incrementer which is validated, the GUI may also ask us how to behave when the incrementer is not valid.

6. The user-defined spectra creator

The online analysis program of the RISING experiments produces typically more than 2000 standard spectra related to the different detectors used in the experiment. Many are defined by default, primarily because they are used the most often. If the experimenter would like to see a special spectrum, he can make it using a special instrument, a *spectra creator* (a user-defined spectra "wizard") available in 'Cracow' GUI. Now we will see the method of creating a user-defined spectrum in a "step by step" manner.

The first thing to do is to choose the name of the new spectrum. The chosen name is always preceded by the prefix "user_". This later helps to distinguish user-defined spectra from the standard spectra produced in the online analysis program by default.

The first page of the wizard is shown in Fig. 2. (As displayed in the figure, the spectrum name suggests that the histogram will define the sum of energy spectra of the 7 Germanium crystals belonging to the cluster detector named 'B'.)

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006

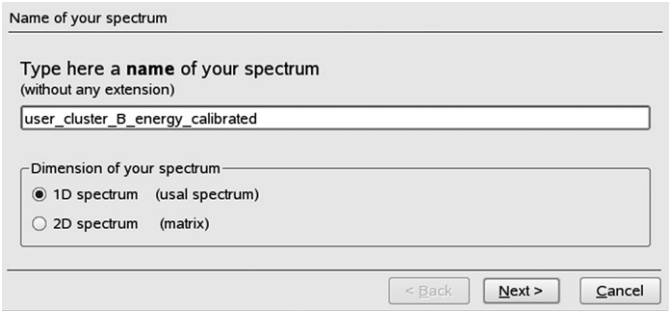

Fig. 2. The user can define a 1D or 2D spectrum. Depending what the user chooses the following pages of the wizard look differently.

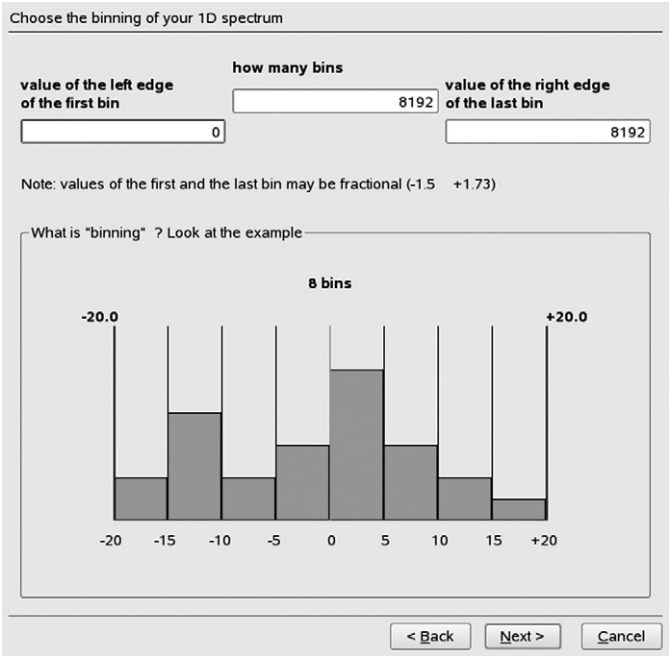

Fig. 3. Choosing the range of a 1D spectrum.

The second decision on this page of the wizard, concerns the dimension of the spectrum. The user can select a one-dimensional spectrum or a two dimensional matrix (scatter plot). If the user chooses a 1D spectrum and presses the button 'Next', then the following page of the wizard requests information about the range and binning of the spectrum. (See Fig. 3.)

Experience shows that some users mix the concepts of "bin" and "channel". In order to help clarify these concepts there follows a graph explaining the idea of binning.

On the following page of the wizard (see Fig. 4) the experimenter defines which variables from the online analysis program may increment this spectrum. There is a table in which the experimenter places the names of the chosen incrementers. During the online analysis the chosen incrementers will increment the spectrum for every event (assuming it is valid and no further condition is applied).

The table shown in Fig. 4 already contains a list of seven incrementers. They have rather long names, intended to be self-explanatory. The user does not have to type these names but rather selects them from a list. In the lower part of the dialogue

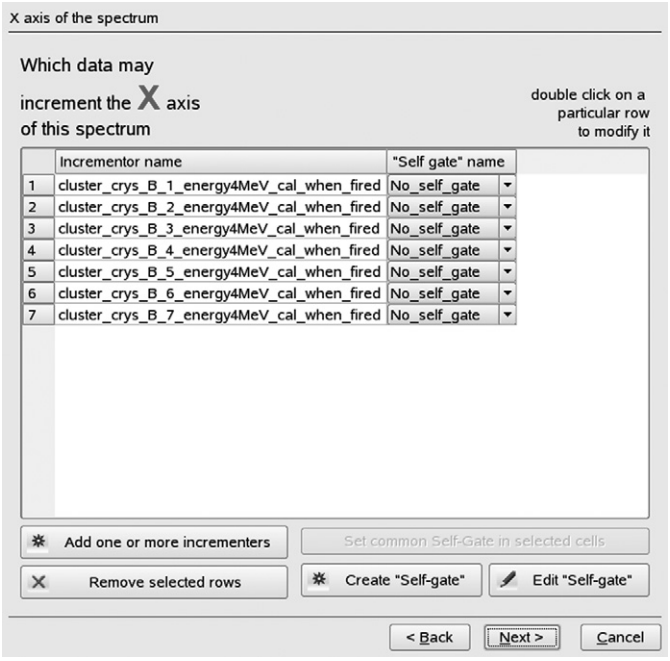

Fig. 4. A list of variables chosen by the user to 'contribute' to their spectrum. Placing more than one incrementer here creates the sum spectrum of those incrementers. Each of the (valid) incrementers will then increment the same spectrum.

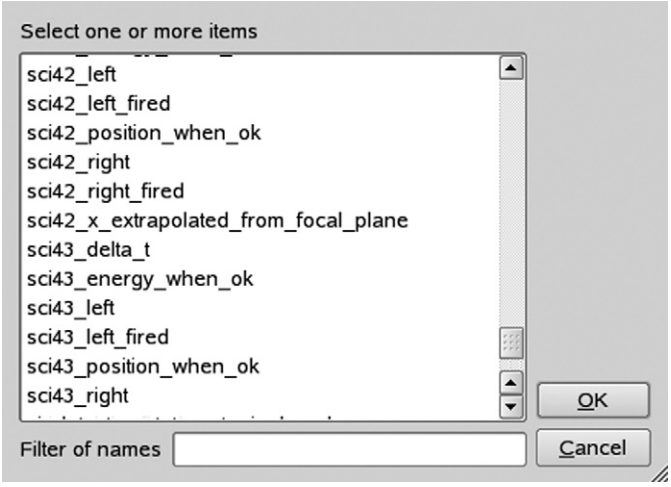

Fig. 5. A list of all incrementers offered by the 'spy' program, visible in the special dialogue box.

page there is a button called "Add one or more incrementers". This button opens a new dialogue box with the list of all the available incrementers (see Fig. 5).

There are circa 2500 available incrementers which are listed alphabetically. To ease the orientation, the names of the incrementers consist of the name of the object (detector) which the incrementer belongs to, this is then followed by the description of the meaning of a particular incrementer (such as energy, position, time, etc.).

For example, looking at Fig. 5, the last but one incrementer in this dialogue box has the name "sci43_position_when_ok". This name explains to the experimenter that the incrementer

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006

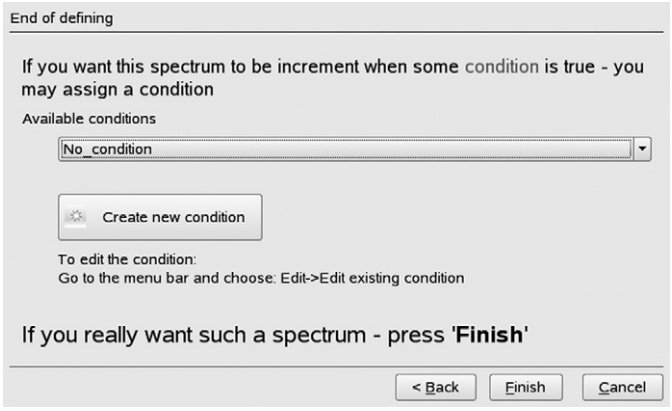

Fig. 6. The user defined spectrum may be incremented under some condition. The user can choose any condition from the list, or create a new condition.

is defined in the scintillator detector called "sci43" and it represents the position calculated by this object. The suffix "when_ok" informs the user that incrementer does not always contain a sensible value (it has a validator checking whether it was possible/impossible to calculate the position).

By knowing this naming convention the user can find the incrementer of interest from this list. There is a tool which makes such a search easier. At the bottom of the window there is a text filter, which allows the list to only include incrementers which pass a given filter.

Returning to the example, using the filter one can display only those incrementers which represent the calibrated (gain-matched) energy data of all the germanium detector crystals. In this case the experimenter is interested only in those belonging to cluster 'B', they can be selected and confirmed by pressing the *OK* button. The dialogue window disappears and the selected incrementers are automatically placed in the table shown on Fig. 4.

The 'Next' button moves us to the last page of the wizard. (See Fig. 6.)

Here the user may apply some condition, but if a condition is not needed, they can finish the work of the wizard. By this the definition of the spectrum is stored on the disk as a text file. The 'Cracow' GUI knows whether the online analysis program ('spy') is currently running, so it can command the 'spy' to read the definition prepared on the disk. The 'spy' reads the definition, and starts to collect the desired spectrum. The new spectrum can be observed immediately using the 'Cracow' GUI spectra viewer in the same way as any other standard spectra. The only difference is that the name of our spectrum starts with the prefix "user_".

Practice shows that the experimenters learn this tool very quickly. Some may think that the names of incrementers and their meaning are the most difficult things here. On the contrary, the names of the incrementers are self-explanatory for experimenters. These are just the general terms which are used while working on experiments and the algorithms of the analysis.

The experimenter can define many different user-defined spectra. At anytime they can also modify an already existing definition. If the definition of some new spectrum is going to be similar to the one which already exists, there is a time saving

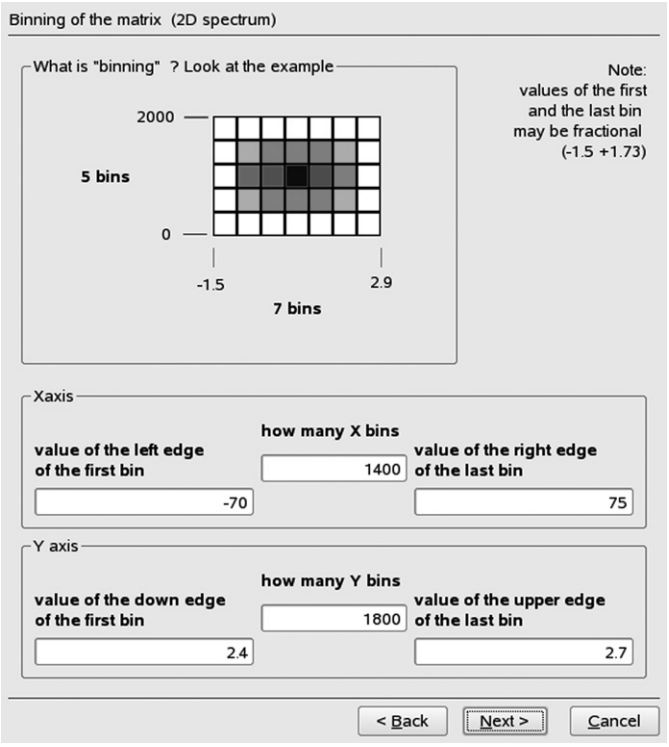

Fig. 7. If the user decided to create the definition of a 2D spectrum (matrix), the second page of the spectrum wizard looks different—it asks questions about the size of two axes instead of just one.

option available of cloning the definition and then modifying this 'clone'.

7. Two-dimensional spectra (matrices)

The wizard is a powerful tool in the GUI. It asks questions "step by step" and, depending upon the answers, adapts the following pages of the wizard to the current situation. Therefore if on the first page of the wizard the user decides that a spectrum needs to be two-dimensional, the next page will look different (see Fig. 7) to the one-dimensional option.

In the case of choosing a 2D spectrum, after the page dedicated to incrementers of the X-axis (Fig. 4), there is a new page which asks a similar question about the incrementers used for the Y-axis. For example, if the user wishes to construct a matrix of "γ-time versus γ-energy" the incrementers representing the γ-energy should be placed on the X-axis while those representing the γ-time on the Y-axis. Fig. 8 shows this example.

In one table the user can place as many incrementers, as they want. What does this actually mean for a 2D spectrum? In the case of only one incrementer for the X-axis of the matrix, and only one for the Y-axis it is clear that the user wants the spy program (during the analysis of each event) to increment the matrix at a point with the following coordinates $P(x, y)$, where

x is current value of the incrementer X and
y is current value of the incrementer Y.

But, as mentioned earlier, the user can apply more than one incrementer on a particular axis.

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006

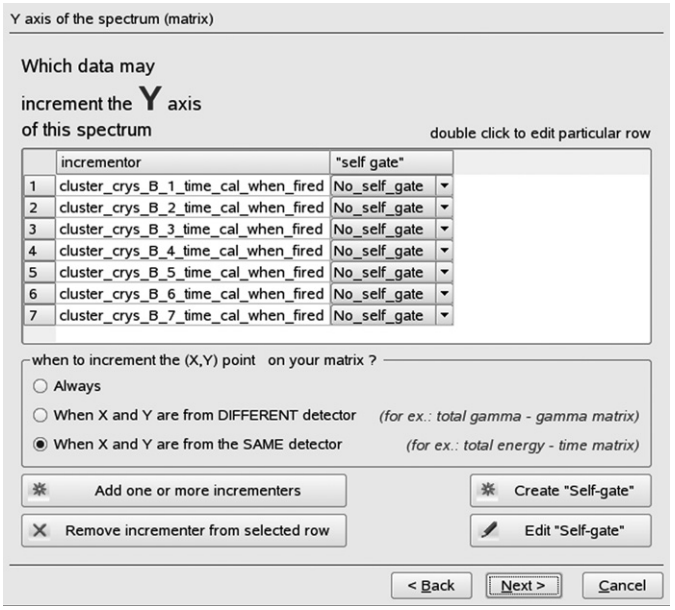

Fig. 8. The matrix must have a list of incrementers responsible for y coordinates of the incremented point $P(x, y)$. Below the list of incrementers there is set of radio buttons, which allows the choice of one of three modes of work. This is important if there is more than one item on the X or Y lists.

For example, for such a user-defined spectrum where there are 7 incrementers defined for the X-axis and also 7 defined for the Y-axis, there is a table produced as follows:

Table of X incrementers	Table of Y incrementers
x_1	y_1
x_2	y_2
x_3	y_3
x_4	y_4
x_5	y_5
x_6	y_6
x_7	y_7

How should the online analysis program interpret this during the analysis of a particular event? There are 3 sensible interpretations, they are outlined in the following sections.

7.1. All possible combinations of incrementers

The online analysis program understands that the user wants to increment points defined by every combination of the x and y incrementers for each event.

$$\begin{array}{llll} P(x_1, y_1) & P(x_1, y_2) & \dots & P(x_1, y_7) \\ P(x_2, y_1) & P(x_2, y_2) & \dots & P(x_2, y_7) \\ \dots & & & \\ P(x_7, y_1) & P(x_7, y_2) & \dots & P(x_7, y_7) \end{array}$$

This is the most general solution. If a user wants this, the 'Always' option on the set of radio buttons in Fig. 8 should be selected.

7.2. Incrementers from the same detector

In the case of the user defining the matrix "γ-energy versus γ-time"—they place the incrementers representing γ-energy in the table Y and the incrementers representing corresponding γ-times in the table X. For this type of matrix the user only wants combinations of the γ-ray energy data coming from the same detector as the γ-time data. Generally there is no physical sense in using the combination of energy data from one detector, with the time data from another, so the user is interested in the combinations

$P(x_k, y_j)$ when x_k, y_j are incrementers from the same detector (i.e. $k \equiv j$).

Note. This does not mean that the incrementer from row 3 of the X incrementer table will be used together with the incrementer from row 3 of the Y table. Such a solution would not be user friendly, because it requires that the user places his incrementers in a strictly defined order—this is a potential source of errors and time consuming.

The user friendly approach is different; the user should not think about the rows in the table, but about the meaning of the incrementers. To facilitate this, the online analysis program ('spy') does not care about the numbers of the rows. The user can place their incrementers in the tables in any order. The 'spy' program is able to recognise which two incrementers belong to the same detector, and can use them to increment the matrix.

If the user wants this interpretation, he should select the 'when X & Y are from the SAME detector' option on the set of radio buttons shown on Fig. 8.

7.3. Incrementers from different detectors

If the experimenter is defining the coincidence matrix "γ–γ energy", they place the same incrementers representing the γ-energy in both tables X and Y. Now they are interested in a different combination of listed incrementers.

If, continuing the example, it is going to be a γ–γ energy coincidence matrix of the germanium crystals belonging to the cluster called 'B' they choose 7 energy incrementers for the X-axis, and the same 7 incrementers for the Y-axis. But now the desired combinations are different. In this case, the user wants to increment their matrix only for the points:

$P(x_k, y_j)$ where x_k, y_j are incrementers from different detectors (i.e. $k \neq j$).

Again, here the k and j do not mean the row in the incrementer tables, but rather the detector which delivers this data.

In other words, in the case of coincidence matrix γ–γ energy, the user is interested in all combinations of energies of the γ quanta, except the situation where the x and y incrementer is exactly the same. (This would create on his matrix a diagonal line.)

If the user wants this interpretation—they should select the second option on the set of radio buttons shown on Fig. 8.

8. "ALL…" collective-incrementers, a list of other incrementers

Very often the user would like to create the sum spectrum of many variables/incrementers, for example, a sum spectrum of γ

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006

energies registered by all of the 105 germanium crystals. Such a "total" spectrum can be easily defined by placing on the list of X incrementers, the 105 incrementers representing the desired (calibrated) variables. However this can be inconvenient. To make such a task easier, there are some special, so-called, *collective-incrementers*, which are equivalent to the list (collection) of incrementers of the same kind. So, for instance, the long list of incrementers:

cluster_crys_A_1_energy_cal,
cluster_crys_A_2_energy_cal,
cluster_crys_A_3_energy_cal,
...
cluster_crys_R_7_energy_cal

can be substituted, by just one collective-incrementer called:

ALL_cluster_crys_energy_cal

The collective-incrementers (users like to call them: "ALL..." incrementers) save a lot of work when defining a spectrum. They are useful, when there are many detectors of the same kind (for example: many germanium cluster detectors, many Miniball detectors [1], many Hector BaF detectors [5]).

When using the "ALL..." collective-incrementers the definition of the γ–γ energy coincidence matrix is very simple, we just:

- place one collective-incrementer ALL_cluster_crys_energy_cal on the list of X-axis incrementers,
- place the same collective-incrementer ALL_cluster_crys_energy_cal on the list of Y-axis incrementers,
- we select the option that we are interested only in combinations where the particular incrementers (hidden inside these two "ALL..." collective-incrementers) are from different germanium crystals.

9. Manager of the user-defined spectra

The experimenter can create many user-defined spectra and all of them are on a special list handled in 'Cracow' GUI by a user-defined spectra manager. This manager allows us to modify existing definitions, remove, clone, or create brand-new. After any such change, the user has the option to immediately send his request to the currently running 'spy' (online analysis program). If 'spy' is currently not running, the changes will wait on the disk until the 'spy' has started next time.

10. User-defined conditions

The experimenter often needs to increment his user-defined spectrum only under specific condition. In the case of the RISING experiments, the most obvious condition is the selection of fragments which come out of the Fragment Separator and hit the target. The experimenter wants to increment his gamma-ray energy spectrum only when the target was hit by the desired projectile (fragment).

Of course there are also less obvious conditions, some of them are invented ad hoc just to see what is wrong with the experiment. So it is very important to give the user freedom over creating conditions. The user can define his condition swiftly online. The user-defined condition, once created by the user, is stored on the disk as a file, and can be used in all future analysis.

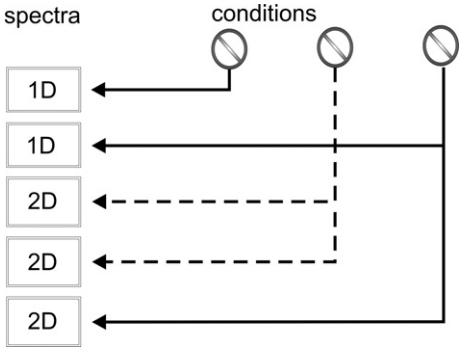

Fig. 9. User-defined spectra and user-defined conditions are separate objects in the C++ program. The user can assign any user-defined condition to affect any user-defined spectrum. The same condition can be assigned to more than one spectrum.

In the software described here, the condition is an independent object. The user creates the condition and may assign it to a user-defined spectrum. One condition can be assigned to more than one spectrum, see Fig. 9.

In this figure we see that a user-defined spectrum can have at most one condition assigned to it. (This is not a limitation, because conditions can be created, which contain a nested logic expression of other conditions.)

11. Condition wizard

The experimenter can create a new condition, or clone an existing one using the condition manager provided by 'Cracow' GUI. As conditions can have very complicated logic, the process of creating the condition is supported by a special instrument called a condition wizard. Let's look at it.

The first page of the wizard contains the simple question about the name of the object representing the condition. The user will refer to this name later when assigning the condition to spectra. To understand better the following, more difficult pages of the wizard, let's try to predict what elementary situations the user would like to test as a condition.

- Sometimes the user just wants to set a simple gate on one variable. For example, he wants to test if some energy is in the channel range 500–520. This kind of elementary condition we will call a *one-dimensional elementary condition*.
- Sometimes the user observes a matrix (created by two incrementers X and Y)—and he would like to have a condition selecting a fragment from this matrix, marked by a polygon with the shape of a banana, or a cloud. This kind of elementary condition we will call a *two-dimensional elementary condition*.

Actually, all we need are these two types of conditions. However, experience shows that the conditions the RISING experi-

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006

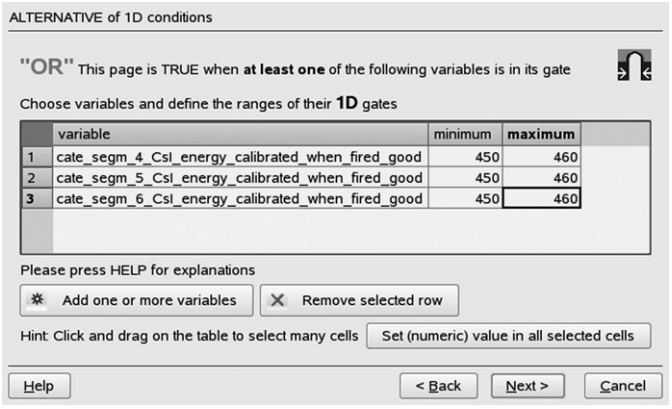

Fig. 10. A page of the condition wizard, which allows the creation of an alternative (OR) of 1D elementary conditions.

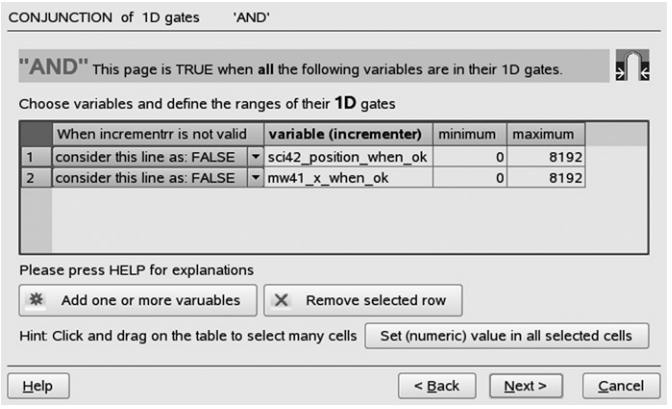

Fig. 11. If the user wants to create the conjunction of 1D elementary conditions, he should also specify how to precede if some of the incrementers do not contain the valid value. In the case of the alternative (OR) gating conditions it was not important, such a row with the elementary condition could be treated as false. With conjunction it is different; sometimes we need to treat it as true, sometimes as false.

menters need, are more complex. Every complicated condition though, can be constructed from these two elementary types, but... with a lot of clicking. So this approach would be too elementary and cumbersome. This is why the conditions available in 'Cracow' GUI are more complex. They allow more complex combinations of elementary conditions to be defined.

The conditions supplied by 'Cracow' GUI can be combinations of the following expressions:

- alternative of some 1D elementary conditions,
- conjunction of some 1D elementary conditions,
- alternative of some 2D elementary conditions,
- conjunction of some 2D elementary conditions,
- conditions of other conditions.

These five ways of defining conditions are available on the five following pages of the wizard. If the user does not need one of them—they simply leave this page empty.

11.1. "OR list" of the 1D elementary conditions

Fig. 10 shows a page of the wizard, where we can place a one-dimensional elementary condition.

To specify this part of the condition we should choose a variable and set a gate on it. It is easy: in the first row of the table there is a place for the variable name (the desired variable can be selected from a list of incrementers). Then, in the next columns, we can type two values: lower and upper limits of the gate. This is enough to create a definition of a simple condition—but very often the users need more.

If we want to have another gate on another variable, then it can be placed in the next row of this table. We can place in the rows of this table as many elementary conditions as we want. By this we create a list of elementary conditions. This list of elementary conditions has a logical value *true*, when at least one of them is *true*. More formally speaking, here the elementary conditions are creating the alternative (OR) of all of them.

$$\mathrm{row1} \vee \mathrm{row2} \vee \mathrm{row3} \cdots .$$

11.2. "AND list" of 1D elementary conditions

If the user prefers not the alternative, but the conjunction of his elementary conditions—the wizard offers this possibility on the next page, see Fig. 11.

At first glance, this page looks like the previous one, but there is an important difference. If we want to check the following conjunction

$$\mathrm{row1} \wedge \mathrm{row2} \wedge \mathrm{row3} \cdots$$

we should remember that sometimes the variable (incrementer), used in a row of this table, may not contain the meaningful value (because, for example, its detector did not fire). How should one make a conjunction of rows in such a case?

The quick answer: "let's treat this row as *false*" may not be a good solution. Imagine we want a condition telling us that all the registered γ quanta have their times in the channel range 500–600. Creating this condition we put the corresponding 105 γ-time variables (incrementers) in this table and we specify range 500–600. This would be wrong. By doing this we are creating a conjunction, which is almost never *true*, because it highly unlikely that all 105 detectors fire in the same event.

So treating a row with an incrementer from a detector which did not fire—as *false*—was not a good choice. Another possible answer: "the row with the variable (incrementer), which does not have a meaningful value, should be treated as *true*"—may also not be a good solution. Imagine, we want the conjunction of the elementary conditions related to the position of the ion as registered by the multiwire chambers.

$$\begin{aligned} &\mathrm{mw41\_x\_when\_ok} \quad \text{is in range } -5, +5 \\ &\mathrm{mw42\_x\_when\_ok} \quad \text{is in range } -5, +5 \end{aligned}$$

This condition should be *true*, when the horizontal position given by the multiwire chamber mw41 *and* the horizontal positions given by the multiwire mw42—are in a certain, small range. Unfortunately, in some events, one of these positions may be impossible to calculate (because the related multiwire

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006

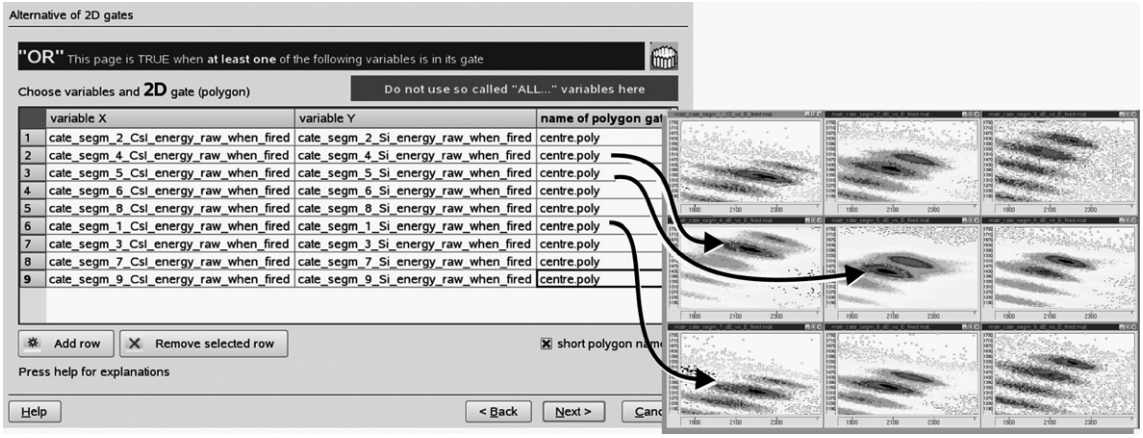

Fig. 12. The condition wizard offers also a page with an alternative (OR) of 2D elementary conditions. In the RISING experiment this was useful for the CATE detector, where the ion could hit only one of nine segments of the detector. As we see, in every row the elementary condition refers to a polygon gate called 'center'. However, there is no confusion, because the real, full name of each polygon also contains the string describing the name of the matrix where the polygon was created.

chamber did not fire in this event). So, how should one treat the row with such an elementary condition? The last example showed us that the solution "treat it as *false*" was not good. What to do? Treat it as "*true*"? No, it is obvious that if one of the multiwire chambers did not deliver the data—the elementary condition with corresponding incrementer should not be *true*. And the whole conjunction should be *false* as well.

As we see—sometimes we need one approach, sometimes the other. This is why the first column in the table contains the combo box, offering the choice.

What should we choose in a particular situation? The answer: "nobody knows this better than the experimenter himself"—is not realistic; especially in an experiment like those from RISING, where we have many new people working on each new experiment. So, to avoid confusion at this point in the condition wizard, the 'Cracow' software helps in making this choice. When the user is trying to use a variable (incrementer), which has a validator, a special wizard appears and suggests to the user what option he should (most probably) use for the incrementer in question.

11.3. Two-dimensional elementary condition on the "OR list"

The next page of the condition wizard deals with two-dimensional elementary conditions. When do we need such a condition? For instance, if the 'Cracow' GUI displays some matrix on the screen, we may draw on this matrix a polygon marking the interesting region. Such a polygon is not yet a condition; it is just a polygon defined by a set of vertices. However, we can use this polygon to create the elementary condition, just by specifying the names of two variables (x and y) and the name of the polygon.

The online analysis program during the analysis of every event will take the current values of variables x and y and check if the point $P(x, y)$ lies inside the polygon. If yes, such an elementary condition is considered as *true*.

On the corresponding page in the condition wizard of 'Cracow' GUI—we have a chance to define not just one 2D elementary condition, but the whole list of them, see Fig. 12. The logical value of the whole page is evaluated as the alternative (OR) of all the rows of this page.

In the first row of the table we can see the names of two variables, and then the name of the polygon. This example is taken from the data analysis made during the Fast Beam Campaign, where we were using the CATE detector [1]. The CATE detector is a chessboard of nine telescope detectors. The ion can hit one of these nine segments. If it does hit exactly one segment, we want to check if the values of dE and E registered by the corresponding segment are in the "banana gate" polygon, drawn on a related dE vs. E matrix.

This is why all nine elementary 2D conditions are on the list, and the logical value of this page is evaluated as the *alternative* (OR) of all elementary conditions (rows)

$$\text{row1} \vee \text{row2} \vee \text{row3} \vee \cdots \vee \text{row9}.$$

11.4. Two-dimensional elementary conditions on the "AND list"

The next page of the condition wizard is very similar to the previous one, but here all the listed elementary conditions are creating the conjunction. This is the most frequently used method of conditioning: we want some values on the X and Y variables to occur inside one polygon gate, AND we also want the current values of some other variables to occur inside another polygon related to other X and Y variables.

Fig. 13 shows the example of using the two-dimensional elementary conditions.

This page defines the condition, which is *true* when all the rows of this table containing the elementary condition are *true* (so: it is a conjunction of 2D elementary conditions)

$$(\text{row1} \wedge \text{row2} \wedge \cdots).$$

As it is a conjunction, here again arises the same problem of what to do if one of variables does not contain a meaningful value (because some detector did not fire and the calculation of some value was impossible). Should the row of the conjunction

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006

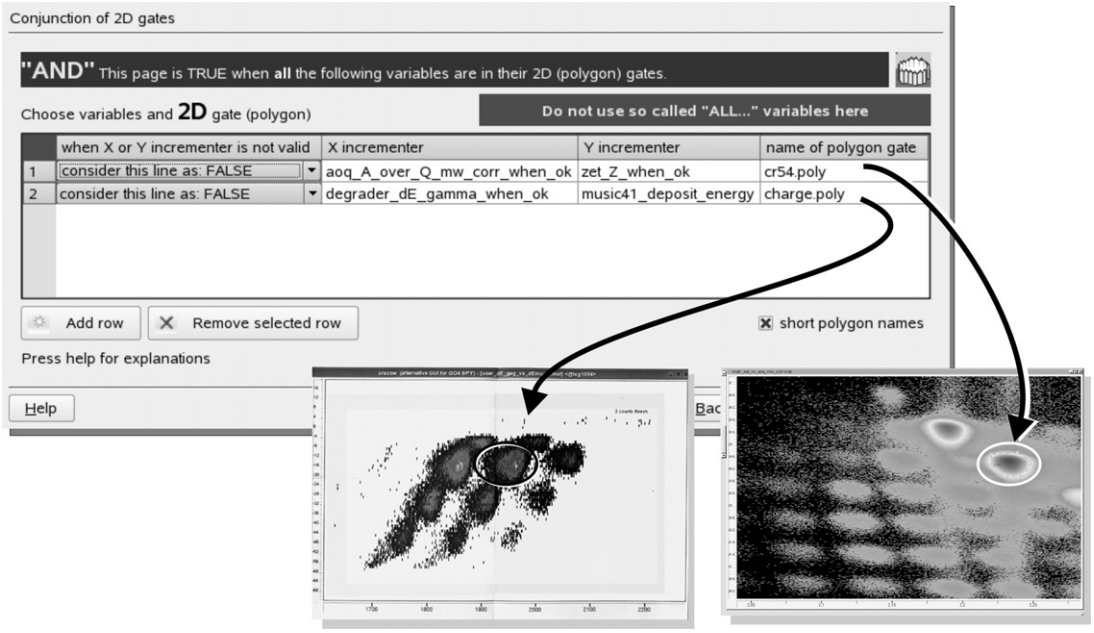

Fig. 13. The page of the condition wizard, where we can create the conjunction of 2D elementary conditions. Here, for example, this conjunction was used for better separation of the projectile coming from the fragment separator.

be in this case *false* or *true*? As there is no general answer—the user can specify case by case in the first column of the table.

11.5. Condition of conditions

The previous pages of the condition wizard can account for very sophisticated situations; with them the online analysis program was successfully used for months. However, the users demanded to have still another feature: the possibility to make logical expressions of other conditions.

When is this useful? If the user has already prepared the condition helping him to select the correct ion coming out of the fragment separator—he can use it in a more precise form, by cloning it and enriching it with additional elementary conditions. This is the correct procedure but, due to cloning, the same elementary conditions ("*is a point lying inside the polygon or not?*") will be evaluated twice for the same event, this is not efficient.

So now there is another, more economic solution. The user can say: *"I want a new condition which is true when the other condition is true, plus some extra condition. Here is this extra condition. . . "*. During the definition of his new condition—the user can also refer to the current values (*true/false*) of other conditions. This is very convenient, but is not only a matter of comfort or economy. The problem first arose when the users wanted a veto detector. There was then a need to build the condition which is *false* if some other condition is *true*.

From all these demands came the special page in the condition wizard, see Fig. 14.

On this page we can see four tables. In each of them the user can place the names of other conditions. (To be user friendly the user need only click and choose them from the list of already existing conditions.) The four tables represent the common logical operations. The first table is dedicated to the AND operator, this means that the user requires all the conditions placed here to be *true* (a conjunction of conditions). The OR table is similarly available (an alternative of conditions). For negation operations—tables with the operators NOR and NAND are provided.

We can see the four tables on this page of the wizard. We do not have to use them all. If some of them are left empty, they are considered to be non-existing. Those tables, which have some content, should be *true*. The logical value of this whole page of the wizard is a conjunction of these four tables.

11.6. Nesting of the conditions is done with care

The possibility of placing the names of other conditions is very powerful, especially as these other conditions can also refer to other existing conditions. So finally we can create a chain of conditions. The length of such a chain is not limited.

However there is a risk that some condition may appear twice in the chain, that would create an infinite loop of conditions to check. For instance: condition A refers to condition B, B refers to C, C refers to D, and D refers to B. By this, the infinite circle $B \to C \to D \to B \to \cdots$ is created. The analysis program will start analysing the first event and will spend the rest of its time jumping from condition to condition not knowing which of them should be evaluated first.

To avoid this error, during editing of the condition 'Cracow' GUI the wizard immediately checks to see whether the user has tried to create such an infinite loop. If such a loop has been created, the user is warned about the logical error.

There is no such risk if we create a brand-new condition—it is new, so no other condition can be referring to it. But if we are modifying a condition created earlier, some other conditions can already be using the now modified one on their lists. We may not remember (or know) this, so we could inadvertently

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006

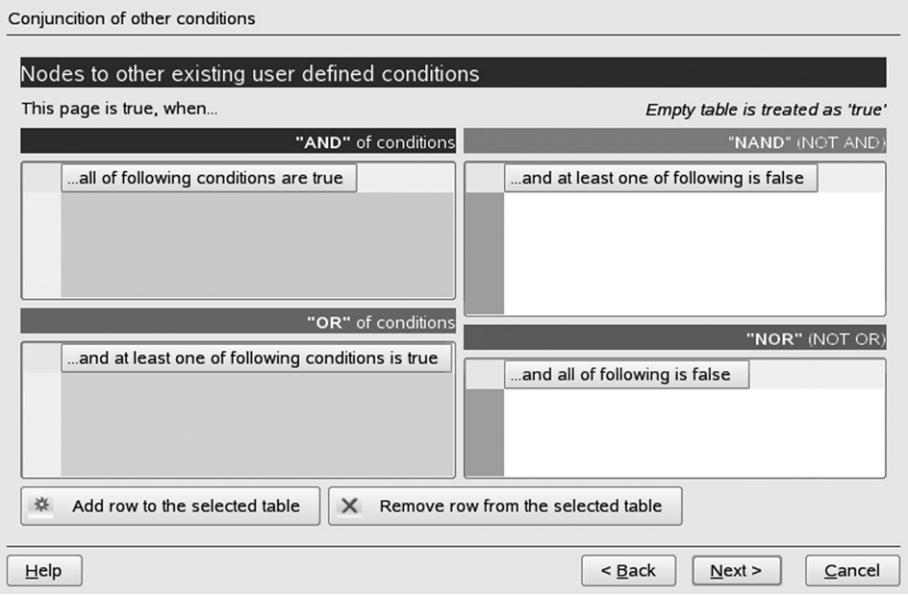

Fig. 14. Using this page of the condition wizard, the user can create any kind of logical expression from the other previously defined conditions.

created an infinite loop in our conditions. Fortunately—this error will be immediately signalled by the wizard. The wizard is even more careful here, as it checks not only the "direct" references, but also the indirect ones. An indirect reference is when we refer to a condition, which is at the beginning of a long chain and also features somewhere further along the chain.

Note that for simplicity here we speak about the 'chain', but as one condition can refer to many conditions at the same time (for example, there can be many names of conditions placed on the AND table—Fig. 14), so actually this system creates not a 'chain', but a 'tree' of conditions. This is no problem, the wizard is sophisticated enough to test all the branches of the tree. The wizard knows that the same condition can be referred to in different branches of the tree, but should never exist twice in the same branch.

This (at first glance) complicated algorithm is, of course, only a problem for the GUI programmer. The user does not have to think about the tree-structure of the condition. He will be only warned if he tries to create an invalid loop.

11.7. Veto conditions

As we have already mentioned, one of the reasons why the users demand the option to create logical expressions of conditions was for the experiments using veto detectors. The ability to implement the negation operator was very important.

If there is only one such detector in the experiment, it is not difficult for a beginner-user to prepare a veto-condition. At first he creates the condition pretending that he wants to accept events, when the veto detector not only fires, but also delivers the data in the forbidden range. Having done this, the user creates a second condition, goes directly to the last page of its wizard—condition of conditions—and here he puts the name of the first condition in the table related to the NAND, or NOR table. (If there is only one condition in the table, then it does not matter whether it is the NAND or the NOR table.) This negation is simple even for beginners.

With two veto detectors, some people have problems. This is not related to the software. Simply: many people do not remember the rules of logic. Here is an example: the user has already created two conditions specifying that his two veto detectors—det A and det B—have fired and registered forbidden situation. Now the user wants to create the condition which is *true*, when veto detector A was "not protesting", AND veto detector B was "not protesting" as well. Shortly speaking the user wants such a situation

$$(\neg \det A) \wedge (\neg \det B).$$

Unfortunately not everybody remembers De Morgan's law:

$$(\neg A) \wedge (\neg B) \equiv \neg(A \vee B).$$

This rule shows that the names of the two veto conditions should be placed on the NOR list. This list is represented by the table in the lower right corner, see Fig. 14.

Rules are rules, but we should understand that sometimes the user is creating his condition during a night shift, and is very tired. To be really user friendly, the wizard supplies the "human language" explanation on the top of the NOR table ("[true,] when all of following [conditions] are false").

11.8. Finishing the condition definition, and assigning it to a spectrum

As we have seen, the condition wizard has several pages where we can specify the lists of elementary conditions, which will be tested (by the online analysis program) every event. Very often users use only one of these pages to specify their wishes. The empty (unused) page has a logical value—*true*. The final logical value of the whole condition is a conjunction of all the 5 pages. This seems to come naturally for all users.

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006

When the user finishes the definition of the condition, the condition wizard saves this definition on the disk. So far the condition exists, but no spectrum uses it. Even if we apply this condition to the currently running 'spy' (online analysis program), 'spy' will not start testing this condition on the analysed events. The 'spy' knows that no spectrum (or other condition) waits for the result of such a test, so saves it time by not checking it. To be tested, the condition has to be assigned to a spectrum, or has to be used by another condition.

Assigning a condition to any user-defined spectrum is simple. We return to the definition of a spectrum, we open it using the spectrum wizard and on the wizard's last page (Fig. 6), there is a combo box with the list of all current existing conditions. By selecting one of them, we assign the chosen condition to this spectrum. From now on this is a conditional spectrum; it will be incremented only if the assigned condition is *true*.

After closing the spectra wizard, the GUI checks if the online analysis program ('spy') is currently running. If it is, the GUI asks if it should send this new definition to the 'spy'. If the definition is sent, the 'spy' opens the definitions of the spectra or conditions and then continues its normal work. The next events analysed by the 'spy', will be tested by the new condition and the new conditional spectrum incremented (or not). Once more we should underline the fact that: introducing even the most sophisticated conditional spectra can be done without the need to recompile the spy program; even without stopping it.

12. Self-gate

The user-defined spectra wizard and the user-defined conditions wizard described above—are very powerful tools. They give the experimenter the possibility to create any kind of spectrum which he needs. Unfortunately sometimes the cost is very high. In this case the cost is amount of clicking needed to create the desired conditional spectrum. If, for some chosen spectrum, we need to create (for example) 105 conditions—the solution is tedious and error prone; hence (from the logical point of view)—purely correct.

Let's take a good illustration from the everyday practice of RISING experiments: the total (sum) spectrum of all the energies registered by the germanium crystals. As these energies are gain-matched we can sum them and create the "total" spectrum. If the user wants such a spectrum all he needs to do, is to define it with a wizard. Using the wizard, on the page dedicated to the X incrementers, he needs to put

- either 105 incrementers
  cluster_crys_A_1_energy_cal
  ...
  cluster_crys_R_7_energy_cal
- or—even faster—a collective-incrementer called:
  ALL_cluster_crys_energy_cal.

After producing such a definition, the expected spectrum will be created by the online analysis. Of course, this definition is nothing special; such a total spectrum is already among the standard spectra supplied by default by the online analysis program ('spy'). The experimenter usually wants something more sophisticated; he wants this spectrum to be collected under some condition.

Let us assume that the user needs a condition where the time-of-flight value (measured by a set of scintillators) is in some particular range (of picoseconds). No problem, all the user has to do is to start the conditions wizard, which is an instrument designated to create such a condition. On the second page of the wizard, the user defines the 1D elementary condition—using variable (incrementer) called: tof_21_41_tof_in_picoseconds_when_ok. After creating such a condition and assigning it to the spectrum, the work is done. The logic of this is clear—even for a beginner.

Unfortunately some beginners tend to use the same logic in cases of some special kinds of conditions. For example: the user wants to collect the same total energy spectrum of gamma detectors, under the condition that the times registered *by the same* germanium *detectors* are in range, say, 4000–4020.

The user starts the condition manager and puts 105 incrementers related to the calibrated times of the germanium detectors in the table 1D AND. (If the user is clever, instead of 105 incrementers, he will use one collective-incrementer called "ALL_cluster_crys_time_cal" which gives the same effect.)

If the user assigns such a condition to his spectrum and starts the analysis of data, he is surprised that the spectrum remains almost empty. Seeing this, he checks the condition manager statistics list; here he can see that his condition is never *true*. No wonder, it is almost never the case that all the detectors that fire register the values of time in the same desired range. If one of them is outside, the whole condition is *false*.

The user realises his error: "*Perhaps it was nonsense to put incrementers on the AND list?*" The user opens the wizard to modify the condition, and he moves his incrementers from the page "1D AND" to the next page "1D OR".

After this modification—the conditional spectrum starts to grow quickly. It is incremented many times, but looking at this spectrum, an experienced user can easily tell that the condition does not work as it was expected. The spectrum is incremented by such γ quanta, which—for sure—were registered with times outside of desired region. So, what is wrong?

The shortest, but metaphoric answer would be: in the "logical expression" that the user creates by defining his conditional spectrum he put the parenthesises in the wrong place. Due to this error, if (in the particular event) we registered 22 γ quanta, and only one of them had a time value lying in the desired range, the condition will be *true* (because it is: OR). So the spectrum will be incremented. *Note*: it will be incremented by all of its incrementers; so by the one γ energy with "good time", and also by 21 "unwanted" γ energies, which were "out of the time range" (i.e. a total of 22 incrementers). Fig. 15 shows the scheme of such a defined conditional spectrum.

The correct way of producing such a conditional spectrum is shown on Fig. 16.

As we see here, every energy incrementer should be used to produce its own spectrum under its own, private condition (a condition, which is set on the corresponding time incre-

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006

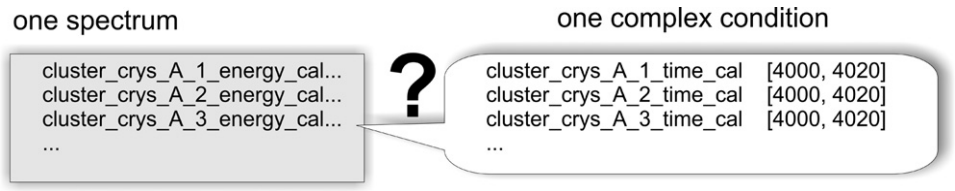

Fig. 15. A very common error, usually made by beginners during defining a conditional sum spectrum. It is not enough to create just one condition and depending of its current logical value, increment the spectrum (or not). The proper solution is shown on the next figure.

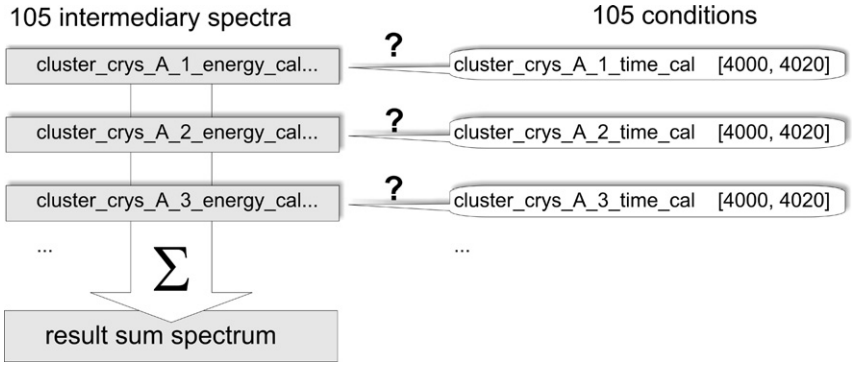

Fig. 16. The correct way of creating the conditional sum spectrum. Unfortunately this requires creating 105 intermediary conditional spectra. To avoid this—there is a better tool, called a self-gate.

menter). So, for example, we should make a definition of the energy spectrum of detector k, under the condition that the value of the time incrementer from detector k is *true*. We need 105 conditional spectra of this kind. Then, they all have to be summed to produce one 'total' spectrum. The summing of the spectra is not a problem; the 'Cracow' GUI provides this feature. The sum spectrum can be created easily and the sum in made not only once, but it is updated every 30 seconds, so the sum spectrum is automatically growing together with all the spectra which contribute to its sum.

The real problem is that the user has to create 105 spectra and 105 conditions. Even when taking advantage of the cloning option—this is a lot of clicking. This is not a user friendly solution; especially since during an experiment, such a spectrum is needed to be observed as soon as possible.

To make this task easy, a special feature has been provided. It is a small condition which we can assign to an individual incrementer placed in the spectrum definition. This specific type of condition is called a "self-gate", because *it checks other variables belonging to the same detector* to which the original incrementer belongs.

For example, if an incrementer represents the value of calibrated energy registered by the cluster crystal F4, we can assign to this incrementer a self-gate, and this self-gate will allow us to check if the calibrated time information in this same detector, F4, is in the desired range. In this case we do not have to specify that we mean the F4 detector. The self-gate looks at the incrementer to which it is assigned and recognises which detector is in question.

If the self-gate condition in a particular event is *true*, the "self-gated" incrementer is allowed to increment the spectrum. If it is *false*, the incrementer is treated (in this event) as non-existing.

The self gate has its own name, so we can apply the same self-gate to many incrementers. The self-gate can check much more than just the time value information. We can even use it to set a gate on the scattering angle or on the constant Θ and Φ angles—describing the geometric position of this detector (Fig. 17).

The self-gate is a tool which saves us from creating many conditions and many intermediate spectra, when we actually need only one. This tool is useful in cases where:

- we have many detectors of the same kind,
- each of them offers several kinds of incrementer (energy, time, scattering angle, angles of their position),
- one type of incrementer has to be summed to create a sum (total) spectrum...,
- ... only if some other variable (in the same event) of the same detector, fulfils some 1D condition.

In the case of the RISING experiments it was useful to support self-gates for:

- Germanium cluster detectors,
- Germanium detectors belonging to the Miniball array,
- BaF detectors belonging to the Hector array,
- the addback algorithm incrementers—used for all the above types of detectors (not discussed in this paper).

As we have seen, the self-gate simplifies the creation of the user-defined spectra. To prove its simplicity, let us look at an example. We need the total energy spectrum of cluster detectors under the condition that the corresponding times are in the selected channel region 4000–4020. To define such a spectrum we should:

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006

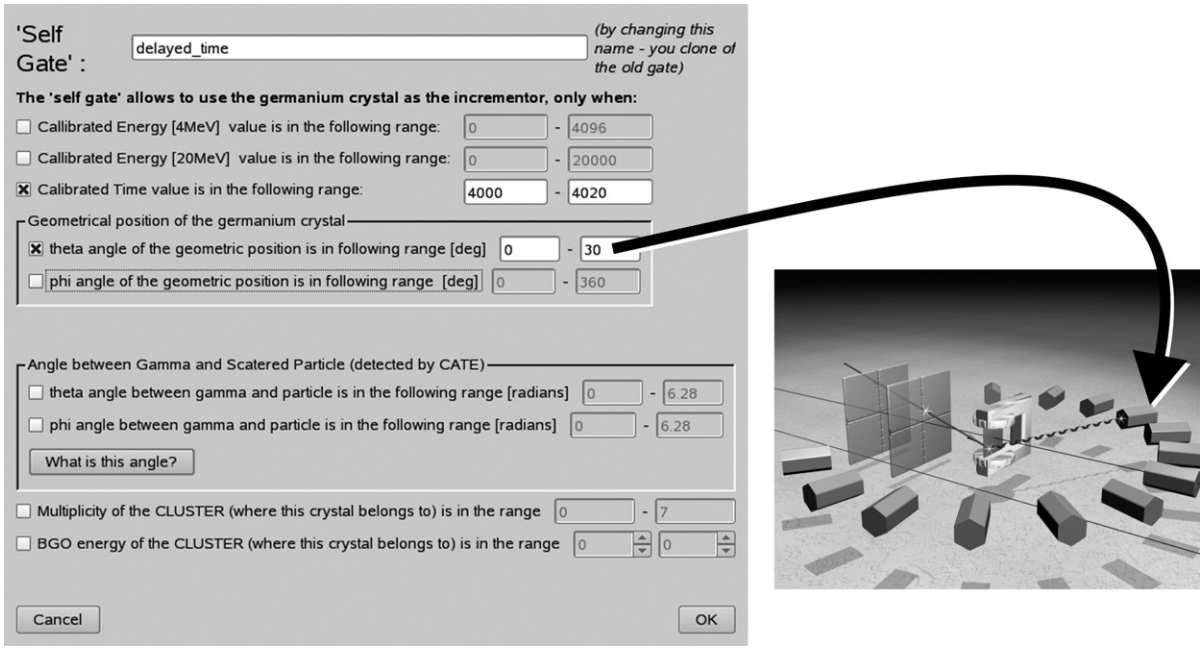

Fig. 17. A dialogue box for the definition of a self-gate used for cluster detectors (with VXI electronics). As we see, the self-gate can be set not only in the time, but also on many other variables belonging to the same germanium crystal. Here—for illustrative purposes—we can see that is was also set on the angular position of the detector.

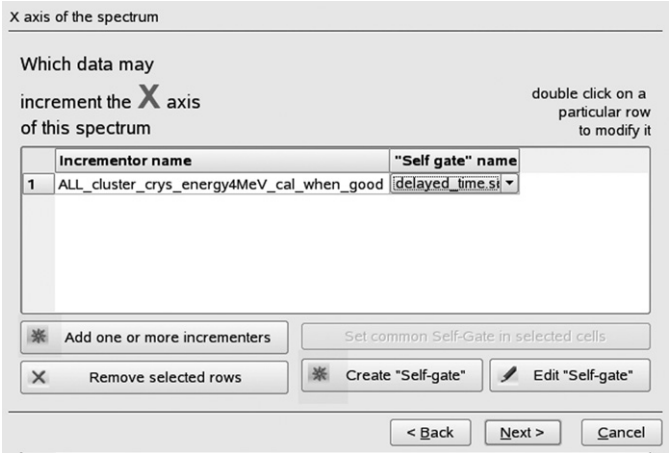

Fig. 18. Thanks to the self-gate, just on one page of the spectrum wizard we can create the conditional sum spectrum described earlier in Fig. 16.

- create the 1D user-defined spectrum, where there is one collective-incrementer ALL_cluster_crys_energy_cal_when_good,
- create a self-gate with the condition on the desired time range,
- assign this self-gate to the incrementer.

All this can be done on one page of the spectrum wizard (Fig. 18).

How does the online analysis program ('spy') proceed in the case of such a spectrum definition? For every event it will try to increment our spectrum. In the definition of this spectrum, the 'spy' sees one incrementer—the collective-incrementer (which is actually the list of 105 incrementers). The analysis program will take each of them and before using each, will go to the related detector to validate the corresponding time (checking the self-gate). Those incrementers, for whom their self-gate condition is evaluated as *true*, will be allowed to increment the spectrum.

To conclude this part, let's underline once more: from the logic the point of view, the self-gate is not necessary. All possible conditional spectra can be created by the spectra wizard and the condition wizard without self-gates. However, the self-gate allows making such definitions much simpler.

13. Online, near-line, offline analysis

The system described here makes the analysis swift enough so that it is really online. The events are coming directly from the data acquisition system in their raw form. Of course the analysis could be faster if the events were not raw, but already pre-sorted and stored on the disk in some optimised form (for example, as a so-called ROOT tree [6]). But an experiment like RISING—needs the analysis to be really online. This is especially true during the startup phase of the experiments, when the experimenter keeps one hand on a potentiometer, changing settings, while watching the expected immediate effect on the online conditional spectrum.

The 'Cracow' GUI strictly collaborates with the online analysis program 'spy' used in the RISING experiment. Here we were discussing the 'Cracow' code, but it is worth saying, that the 'spy' is based on the Go4 library [7]. Thanks to this, the events for the analysis can be obtained either online (from the DAQ system), or from the "event by event" (list mode) data file stored by the DAQ on the disk a few minutes earlier. This mode of work (a *near-line analysis)* has an advantage that 100% of events are analysed (while in the online mode—only some frac-

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006

tion, depending on a counting rate and complexity of the online analysis algorithms).

No matter what mode (online, near-line) of analysis the user chooses, he can use the same tools described in this paper for defining his analysis. The users like this fact, they often ask to install spy/Cracow code on their Linux laptops, because—after experiments—when they return to their home laboratories they would like to continue their analysis offline with the help of the spy/Cracow code.

Some users even ask if it would be possible to implement the Cracow code to analyse the data taken in different laboratories. Yes, it would, but—as the 'Cracow' code is strictly collaborating with the 'spy'—their local program for offline analysis (their local 'spy') should be modified, to:

- 'publish' its list of available incrementers (variables) specific to a particular experiment,
- implement the object oriented procedures responsible for handling the user defined definitions created by 'Cracow code' (spectra and conditions).

As most physicists nowadays understand that good analysis software [8] should be written using the object-oriented technique—the adaptation of such object-oriented software should not be difficult.

14. Final impressions of the author

The system of interactive creation of the user-defined spectra and user-defined conditions, was invented as a solution to make the experimenters independent (of me), by giving them a universal instrument to go through a long commissioning phase. When this phase was over, we were still profiting from this tool, so the system was constantly being improved. New users coming for the new experiments are learning this tool surprisingly quickly. Soon they forget that in their hand they have complicated software—they are now talking only about the physics seen on the screen. The physics, "extracted" by themselves—from millions of numbers.

Acknowledgements

The 'Cracow' GUI project would probably never exist without the enthusiastic experimenters and members of the RISING project. Also special thanks go to Joern Adamczewski, who patiently explained to me the mysteries of GO4 and to Steven Steer for fruitful discussions about this text.

I would like to express my gratitude to many unknown friends, the creators of the KDevelop project [9].

This work has been partly supported by the Polish Ministry of Education and Science (Grants Nr. 1 P03B 030 030 and 620/E-77/SPB/GSI/P-03/DWM105/2004-2007).

References

[1] H.J. Wollersheim, et al., Rare ISotopes INvestigation at GSI (RISING) using gamma-ray spectroscopy at relativistic energies, Nucl. Instr. Methods in Phys. Res. A 537 (2005) 637.
[2] J. Simpson, Z. Phys. A 358 (1997) 139; Achievements with the EUROBALL, in: W. Korten, S. Lunardi (Eds.), Scientific and Technical Report 1997–2003, 2003.
[3] H. Geissel, et al., Nucl. Instr. Methods B 70 (1992) 286.
[4] H. Essel, J. Hoffmann, N. Kurz, R.S. Mayer, W. Ott, D. Schall, IEEE Trans. Nucl. Sci. NS-47 (2) (2000) 337.
[5] A. Maj, et al., Nucl. Phys. A 571 (1994) 185.
[6] http://root.cern.ch/.
[7] http://www-w2k.gsi.de/go4.
[8] B. Jacobsen, Applying object-oriented software engineering at the BaBar collaboration, Nucl. Instr. Methods in Phys. Res. A 389 (1997) 1.
[9] http://www.kdevelop.org.

Please cite this article as: J. Grębosz, The Cracow code—an interactive method of sophisticated online analysis, Computer Physics Communications (2006), doi:10.1016/j.cpc.2006.09.006