Analysis Environment Challenges

Lassi A. Tuura*

Northeastern University, Boston, Massachusetts 02115

Abstract. This paper discusses some of the challenges modern high-energy physics analysis tools are or will be facing shortly. We take a brief look at the environment in which the tools will need to operate and the requirements this implies. We then explore a handful of concepts that may prove helpful in responding to the challenges. Some general ideas on implementation are also given.

INTRODUCTION

Experimental particle physics has high data processing demands due to the need to transform large volumes of data into highly distilled statistical analyses. To present results to others, large samples of data must be scanned, drilled into, reduced, viewed in different ways, and fitted to models. This process involves much of the experiment software. Sometimes reconstruction needs to be redone with changed parameters and improved algorithms. It is not uncommon to find that a part of the "final analysis" needs to be migrated back the chain, for example to the offline reconstruction or the triggers.

With the growing volume and complexity of the data gathered as well as the rising user-interface expectations, pressure on the analysis tools is increasing. Existing tools no longer perform satisfactorily. This paper takes a look at some of the key challenges from the software technology point of view, concentrating on the more important issues and requirements. These are followed by a discussion of concepts that may prove useful to improving the tools, including a few ideas on implementation.

The purpose of this paper is to motivate discussion on where future analysis tools should be going. Hopefully this discussion will lead us to better understanding of the problems we are trying to solve, and hence better tools for physics analysis.

ANALYSIS ENVIRONMENT CHARACTERISTICS

It could be said that physics analysis is largely an iterative process of reducing data sets to more interesting subsets and deriving new quantities from the data at hand, as illustrated by Figure 1. The outcomes are highly distilled statistical analyses intended for human interpretation and discussion, usually histogram plots in two or three variables of a large number of measurements fitted to a distribution function. New quantities are usually derived either to increase the abstraction level ("this trajectory is a muon particle track"), to summarise data ("the highest momentum track in this event has p_T of 65.5 GeV"), or to derive statistical quantities from repeated measurements ("classify the momentum of these million tracks into a range from 50 to 100 GeV in steps of 0.5 GeV"). Much of this work can be done in an interactive analysis and presentation tool on very high-level entities such as simple scalars and 4-vectors. This tendency is clearly manifested by the past focus on tools operating on simple summary information such as DSTs, N-tuples and tag databases.

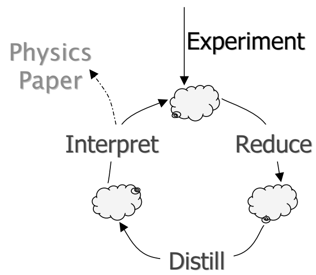

FIGURE 1. Generic physics analysis process

However, it could also be said that the above description is, if not incorrect, at least an immensely limited view of physics analysis. Analysis involves *a lot* more than just the interactive tool. We can learn from the "PAW revolution": PAW (1) provided a new user interface and new, more powerful ways of working with data through N-tuples. Better access to data did not stop with PAW, only today we can do much more and better than just N-

* This work is supported by US National Science Foundation

tuples. For example, ROOT (2) has added trees, and CMS uses a complete object model (3).

Consequently experiments are making their data accessible in new ways by exploiting object models that have proved themselves very powerful. Besides this advances in networking, distributed computing and mass-storage systems have opened new possibilities of automation and integration to the collaborations. It is now quite reasonable to assume that one can in many cases navigate transparently from one part of the data to the other even when the destination objects will need to be first fetched from tape or transferred from a remote site. For example, we are now growing to expect that one can navigate from a N-tuple back to the data it describes. Similarly, we are no longer taking for granted that an analysis job should run on a single computer, or even any particular computer at all.

Where we are particularly lacking is new user interfaces to match these advances. The new object and computing infrastructure models require corresponding advances from analysis tools. To accomplish this, the next-generation analysis tools will need considerably deep links with the rest of the infrastructure in the experiments. This is bad news as it will be much harder to develop experiment-independent tools.

ANALYSIS TOOL REQUIREMENTS

Let us now turn to specific challenges new tools will need to address. Here we explicitly assume we are reaching *beyond* the present functionality: all the existing capabilities in data analysis and presentation are expected to be retained in one form or another, including familiar operations on N-tuples, histograms, fitting and plotting.

We observe that the range of analysis activities is great, with no clear boundaries. They include for example:

- Batch analysis;
- Interactive analysis;
- Setting up software configuration management tools, application frameworks and reconstruction packages;
- Data store operations such as replicating entire data stores; copying runs, events, and event parts across stores; not only copying but also doing something more complicated (filtering, reconstruction, . . .);
- Browsing data stores, selections and other collections down to object detail level;
- 2D and 3D visualisation;
- Moving code across final analysis, reconstruction and triggers.

Each of these has considerable range in itself. People have different preferences for interactive analysis style, and those tastes vary according to the task at hand. Sometimes a “pointy-clicky” graphical interfaces is called for, at other times an interface more powerful if also more cryptic in the spirit of Emacs (4), yet at other times a command line interface is the best. As another example, analysis often develops from a quick interactive cut to a more elaborate expression to some quite complicated code that is best ran in the reconstruction framework. By this time one needs to know how to compile and incorporate code into the reconstruction framework—for example, how to check out a workspace package from the source code management system, copy the sources into it, build it, and tell the framework to load and run it at an appropriate time.

Today all this is largely possible but involves far too many tools and the physicist has to painfully learn to master each one. This usually results in a collection of “magic incantations”—commands and code fragments not understood by the author but somehow causing the system to carry out the desired task—not a coherent environment. The transition from one environment to another usually implies pain (the all too familiar “Oh dear, I must now convert this KUMAC into a FORTRAN reconstruction package and start using data in a completely different format” scenario). The other extreme, a single environment that solves all the problems, is not very appealing either as experience has too often showed that such environments fall short on quite a few important accounts and escaping outside the environment is often painful. The concept of a coherent, seamless and well-integrated environment *is however extremely appealing.* In our opinion one of the greatest challenges is how one can develop such analysis tools without dictating a single environment to the experiments.

A Concrete Example

Let us consider a relatively simple concrete example to illustrate these points. Let us assume I wish to replicate and share some of my experiment’s data for analysis tasks and GRID-style[1] use. While I have tools to do so, I immediately stumble on the following:

- Do I understand my experiment’s world-wide configurations well enough to use the tools confidently? (That is, can I be sure that I am not doing something utterly silly?)

[1] Please refer to other talks in the conference and references in them.

- How do I find the data store nearest to me in the first place?
- If I want a private working store that shares the experiment data at the same time, exactly what should I do?
- What if I do not want just a plain file copy, but want only a copy of the reconstructed data for the calorimeter from a certain sample that includes events in tens of files?
- What if I want to share my analysis settings and results with my colleague for verification?

There are answers to all these questions in each experiment. Some of the answers are even well documented. The challenge to us is: is there a *coherent* answer? To what degree can the answer be applied from one experiment to another?

CONCEPTS TO GET US THERE

We will now suggest a couple of concepts that might prove helpful in responding to the above challenges. These are simply ideas; the author certainly has not yet tested all of them in analysis tools. We do however include some examples below how existing (mostly unrelated) applications are already employing these ideas. Some analysis tools, such as JAS (5), are also pursuing some of these concepts already. In other words, these concepts are not novel. Most of them can be considered industry-standard software design practice. As we apply the concepts to our domain, hopefully something useful will emerge.

First we observe that a *uniform integrated user interface* to the whole task range would be most useful. It should encourage a coherent look and feel within reasonable limits, both for command line and graphical use. It could take a form of a *tool suite* or a work bench, together with a sufficiently large set of common building blocks. It should be possible to host tasks that are most naturally done via a graphical user interface[2] inside a coherent, standard framework—a document-centric model such as those found in most of today's office applications would probably feel most natural to many users.

Secondly, a cohesive solution to a whole slew of the tasks, including the more complex ones, is required. *Wizards* seem to be emerging as the popular solution: most find them easy to use as they guide one through each task by explaining the choices, giving sensible defaults and by making the steps clear. In our environment most of the wizards ought to leave behind a repeatable "trail" such as a script so that someone preferring to see actual code can replicate the wizard's actions once the steps are more familiar. It would also allow the wizard's actions to be postponed and carried out in a batch environment.

[2] Here the user preference matters just as much as the task itself.

Another quite evident idea is to represent the data store or parts of it[3] as a *directory structure*—and allow one to browse it as such, much like one views the file system with modern file browsers. It is not a long step from there to provide a conceptual *home directory* where one ends up upon "logging into" the system. It should be easy, if not the default, to put objects relating to one's analyses under this home directory, including setups for the framework and reconstruction packages, parameters and so forth. Data store browsers should be able to show, inspect and interact with any object in the directory structure, just like modern file system browsers can not only show the file system structure and basic file information, but can also spawn applications for any file type within themselves (6).

Finally, it should be easy to access and interchange analysis setups and objects with different individuals and physics groups. It should be possible for one to keep track of configurations, input and output data selections, and so forth. Following again prevailing user interface standards, perhaps a *desktop* would be most appropriate, allowing one to put on it shortcuts or symbolic links to the objects and "directories" of interest. Standard shortcuts for common things would naturally be part of this.

For all this we must bear in mind that one size never fits all. It is the analysis tools that must adapt, not the physicists and the experiments. Hence in all the above there is an implied requirement for the implementations to be modular.

Examples In Today's Applications

Figures 2 and 3 show some of these ideas in action. It is not difficult to extrapolate them to the physics analysis.

The left side of Figure 2 shows a recent version of Microsoft Outlook (7). On the left there is a quick access bar to common items. Next to the bar there is a hierarchical browser. On the right side are various levels of detail on the item selected in the hierarchical browser. In the figure that happens to be a mail folder and a message from the folder, but it could also be a several different views of the calendar—and many other things.

The right side of Figure 2 shows the Windows 2000 file save dialog above and an IGUANA (8, 9) database browser prototype below. The file save dialog presents

[3] Not just event data, but also other data and possibly even software components in the running program.

again the standard “root” locations on the left for an easy access to items of interest to me personally. One can imagine such a dialog to be offered when saving any analysis object such as a histogram, event collection or a N-tuple. The IGUANA prototype demonstrates how easy it is to work towards these concepts: this simple prototype can only browse histograms and doesn’t offer any of the conveniences a full-fledged browser ought to offer, but already has much to offer in that direction. Similar examples can be found in other recent analysis tools.

Figure 3 shows a combination of co-operating views. DDD (10) is a graphical debugger front-end—it only provides a graphical user-interface on top of command-line debuggers; exactly what it does is mostly irrelevant here. In DDD one can click around in the data to expand and collapse the view (top window), view data and issue commands in the source window (in the middle; note the tooltip when mouse hovers above a variable name), execute common commands from the floating command tool (on the right), or type in commands on the command line (bottom window). The command line reflects actions carried in the other windows and vice versa; the visualisation window reflects changes in the program being debugged. As far as the user is concerned, none of the views is more important than the others; one just uses whatever window is most convenient. What is not visible is that each of the views can be disabled and enabled separately, or split off into separate windows. DDD can also run in a command line mode without any of the windows—the requested windows will then pop up when needed.

Road Map

How might we implement such concepts? Few experiments can afford to develop a new interactive analysis tool, let alone coherent tool set for the entire range of analysis tasks. If as a community we want these tools, we need to partition the problem and co-operate. We need to define categories such as GUI, event and detector visualisation, and data analysis and presentation. In each category we need to use existing modules where possible, and write our own only when nothing suitable exists. All the components should be integrated to form a user-friendly and productive environment.

In this manner we can build a pool of components out of which we can draw to construct customised applications with relatively little effort. For this work out the pool needs to be *truly modular* which means we must consider *all* component dependencies, not just the obvious ones. We can get a good idea of the immediate dependencies of each component by analysing how all the features provided by that component can be *tested* (11). The result will be the union of the source and binary dependencies (the bare minimum required to build and run the test) plus logical dependencies from the ways the components must be used together.

Architectural Issues

The single key to a successful analysis environment we consider to be an architecture that is modular where it matters the most. Not everything needs an abstract interface—it may be better to make a strategic choice to use a particular product if it can be contained and completely replaced in something like 6–9 months.[4] In other words, the point is to *assess and bound risks*, not achieving total security of any change.

We suggest the following as guiding design principles:

- Use Model-View-Controller and similar patterns to partition the domain;
- Use layering to separate front-ends and back-ends;
- Ensure a standard for visual components to facilitate integration.

Abstract interfaces are required for all data access, and very narrow interfaces are required to link the analysis and visualisation subsystem to the experiment’s core frameworks. Let us illustrate this principle with event selection using summary data: a concept that is implemented in many ways, such as tags, N-tuples, DSTs, and B-tree indices. N-tuple is both an access paradigm and a storage method, and historically the emphasis has been on the latter. It would be necessary to shift the focus to the *access and query interface* that continues to provide the look and feel of the proven access method with natural modern extensions. The implementation should be allowed to vary, and could be changed to take advantage of better ways of storing and organising events, indexing schemes provided by the data stores, or even computation on demand by the core framework and tape reads. Other interfaces should be provided to control the aspects outside querying, while the primary query interface should remain unaware of and unaffected by such concerns. As far as the client code cares, a physical N-tuple should not even need to exist.[5] For example, a database query result could be represented as a N-tuple: it could be iterated over one row at a time, and the previous row might vanish once it has been discarded.

[4] For example, it would be better to use OpenInventor (12) than to invent one’s own 3D library.

[5] Obviously the client *user* might care and might want to direct the system to make the query results persistent.

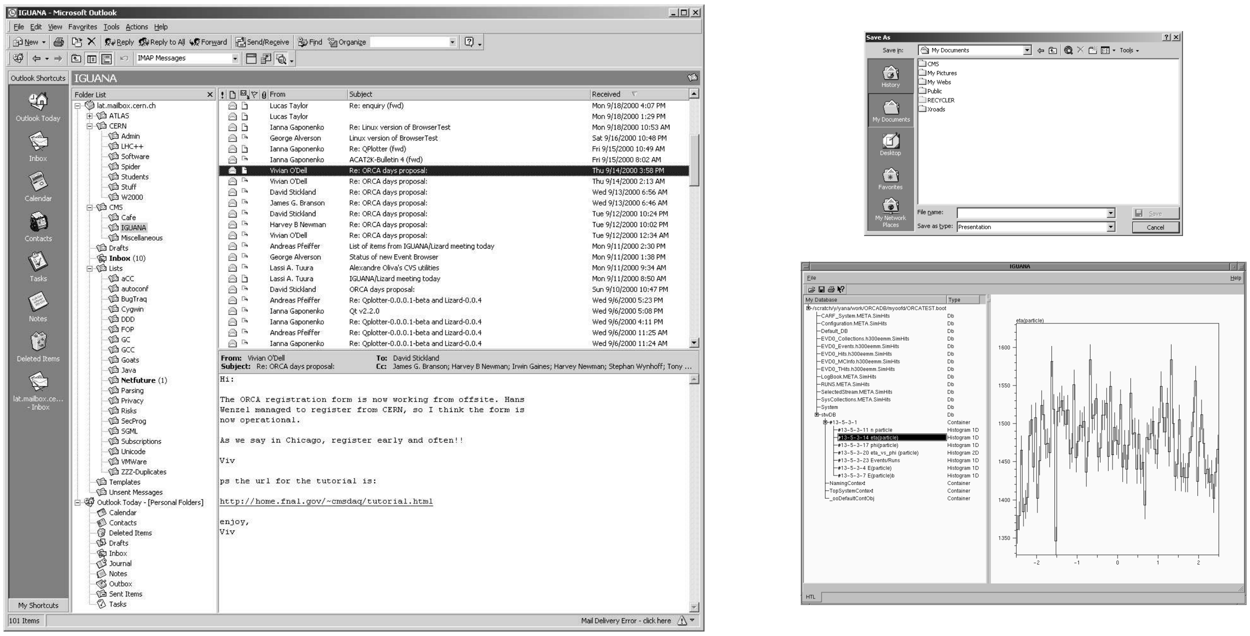

FIGURE 2. Modern user interface concepts: flexibility in Microsoft Outlook (left), and quick access to important items in the Windows 2000 file save dialog (top right). Also an IGUANA prototype on the way to such concepts (bottom right).

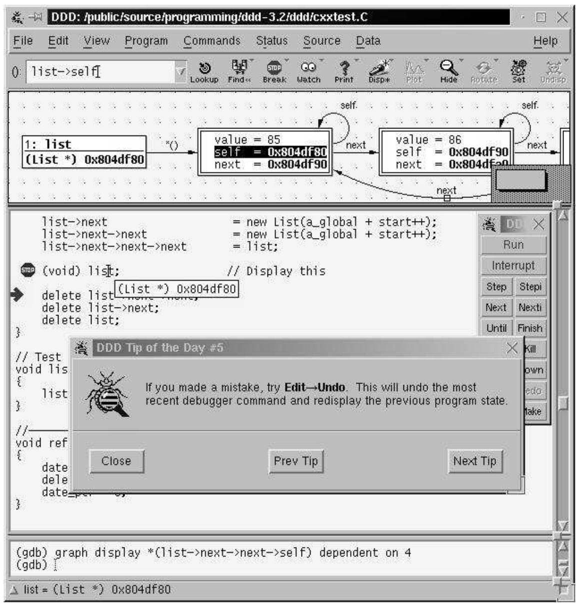

FIGURE 3. DDD illustrating how multiple different views can be combined for a flexible and productive interface to interact with the underlying system, ranging from very graphical (top) to a basic but powerful simple window (middle) to command line (bottom). Command line reflects actions in other windows.

CONCLUSIONS

Analysis environments include much more than just the interactive analysis and presentation tools. So far the focus has been largely on the latter, but as the experiment complexity grows we need to be able to drill down and interact with the data in new ways. Consequently a good, coherent, solid user interface for the whole range of tasks all the way from batch mode operation to the quick point-clicky jobs is required. Building such a tool suite from scratch is neither affordable nor wise. Instead we need to use existing components from our own community, open source and commercial packages.

For this to be practical, each component must be given a very clearly defined responsibility: a simple clear mission statement. Abstract interfaces are useful means in this to carry out some of the domain partitioning and layering. This way risks can be bounded should a particular technology or a component fail. Abstract interfaces will also provide natural points to hook in many of the exciting new ideas that we will see. Finally, they have the benefit of allowing people to co-operate without disturbing each other too much.

Some of these ideas are already being explored by many projects, but we still lack a comprehensive solution with a vision of catering to the complete range of analysis environment tasks.

ACKNOWLEDGEMENTS

I would like to thank my colleagues earlier in ATLAS and now in CMS, and IGUANA project in particular, for the many insightful discussions that have lead to the development of these concepts.

REFERENCES

1. Brun, R. et al, *PAW Physics Analysis Workstation*, 1989. CERN/CN Long Write-Up Q121.
2. ROOT Home Page, `http://root.cern.ch/`
3. Innocente, V., *CMS Software Architecture: Software framework, services and persistency in high level trigger, reconstruction and analysis*, 1999. CMS IN/1999-034.
4. GNU Emacs Home Page, `http://www.gnu.org/software/emacs/emacs.html`
5. Java Analysis Studio Home Page, `http://www-sldnt.slac.stanford.edu/jas/`
6. Microsoft Internet Explorer Home Page, `http://www.microsoft.com/windows/ie/`
7. Microsoft Outlook Home Page, `http://www.microsoft.com/office/outlook/`
8. Alverson, G., Gaponenko, I., Taylor, L., Tuura, L., *The CMS Interactive Graphical User Analysis (IGUANA) "Functional Prototype" Software*, 2000. CMS IN/2000-052.
9. Alverson, G., Gaponenko, I., Taylor, L., *IGUANA – Interactive Graphical User Analysis*, 1999. CMS IN/1999-042.
10. GNU DDD Home Page, `http://www.gnu.org/software/ddd/ddd.html`
11. Lakos, J., *Large Scale C++ Software Design*, Addison-Wesley, Reading, MA, 1996.
12. Wernecke, J., *The Inventor Mentor*, Addison-Wesley, Reading, MA, 1994.