COMPOSITE FRAMEWORK FOR CMS APPLICATIONS

V. Innocente, CERN, Geneva, Switzerland,
G. Eulisse, S. Muzaffar, I. Osborne, L. Taylor, L.A. Tuura,
Northeastern University, Boston, USA

Abstract

We present a composite [1] framework [2] which exploits the advantages of the CMS data model and uses a novel approach for building CMS simulation, reconstruction, test-beam visualisation and future analysis applications. The framework exploits LCG SEAL [1] and CMS COBRA [2] plug-ins and extends the COBRA framework to pass communications between the GUI and event threads, using SEAL callbacks to navigate through the metadata and event data interactively in a distributed environment.

We give examples of current applications based on the framework, including CMS test-beams, geometry description debugging, Geant4 [3] simulation, event reconstruction, and the verification of reconstruction and higher level trigger algorithms.

INTRODUCTION

CMS users wishing to contribute to physics analysis face a steep learning curve due to the complexity and the ammount of the software written in CMS as well as the number of external packages they need to incororate. The community has long discussed the idea of creating a coherent graphical user environment where a user would have the same interface to all CMS applications whether it is on-line or off-line, simulation or reconstruction, DAQ or test-beams.

The first step toward such an environment has been done by creating a new IGUANACMS [4] project based on the CMS build sysyem, SCRAM [5]. IGUANACMS by definition depends on all other CMS projects. Thus a user working within a developer's area of this project enjoys a consistent cross-project run-time environment. The distribution of IGUANACMS includes consistent set of versions of both the CMS and external software.

From the user point of view IGUANACMS is seen as a composite framework providing coherent interface to the majority of the CMS applications and frameworks neither of which predominates over the other. Later in this paper IGUANACMS is referred to as a visualisation framework.

One of the challenges in implementing a coherent environment is the integration of the frameworks which are not a priori designed to collaborate. We give successful examples of such integration and describe various ammount of effort required to accomplish the task.

[1] composite [adj] - consisting of separate interconnected parts;

[2] framework [n] - a set of classes that embodies an abstract design for solutions to a number of related problems, Computer Dictionary.

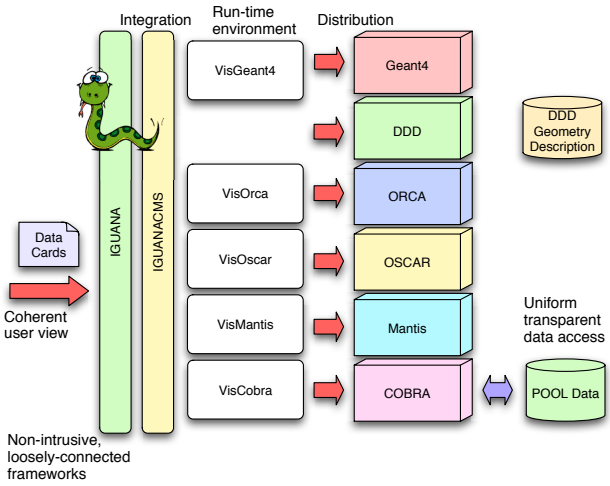

Figure 1: Uniform user view of the CMS visualisation applications for ORCA, OSCAR, COBRA, Geant4, Mantis, DDD, and test-beams.

VISUALISATION FRAMEWORK

The presentation layer of the visualisation framework (see Fig. 1) - the layer which user sees and works with - is based on IGUANA object model [6]. This layer is known to the user as iguana. IGUANA is a SCRAM-based project, it is a generic visualisation framework that provides a command "iguana" and a default driver "IGUANA". The architecture and design of IGUANA is described elsewhere. [6].

The default mode of running iguana is a GUI (graphical user interface) application. GUIs are generally recognized as being very good for visual tasks, especially document editing and some common kinds of file management, and helping the user perform uncommon or unfamiliar tasks. Alternatively the user may run iguana and its drivers via a command line interface or in a batch mode. Nothing prevents such an implementation since the GUI dependent layer is well separated from the rest of the framework.

On startup iguana presents a list of available at run-time session types to the user. Such session types are SEAL plug-ins that correspond to a shared library. These plug-ins are described in detail elsewhere [6]. The user selects one of the session types with a predefined configuration for the iguana studio, the graphical user environment. Depending on the configuration iguana studio loads the relevant plug-ins and becomes one of the CMS visualisation applications for CMS reconstruction - ORCA [7] (CMS reconstruction project), simulation - OSCAR [8] (CMS simula-

tion project), or iguana examples.

To provide additional configurability of the predefined session type the user can pass data cards defined in a configuration file as a command option. These cards can also define the session type and futher configuration of the desired application and algorithms; e.g. the list of the plug-ins to load at the startup and the input or output data he or she wants to work with.

EXTENDING COBRA FRAMEWORK

Visualisation applications do not need to know anything about the persistency of the data to be visualised. Indeed, the user may not even be aware of what is going on behind his request to visualise a certain physics quantity. For example, if the object is persistent, i.e. it has already been reconstructed and stored for future use, the request to COBRA will be converted to a read of this object from the data store, after which it is passed to the visualisation framework to be represented according to the user's request.

A more complicated scenario ensues when the user works with simulated data. In this case such a request will trigger a chain of actions: first digitization of the hits and then reconstruction. The transient result will be passed to the visualisation framework. Thus, as far as the user is concerned, the visualisation process is the same as the simple read request.

It is the responsibility of the visualisation framework to create a model from the object, put the model in a context, and assign one or more representations of said model in each of the contexts. More detailed description of the IGUANA object model architecture can be found at the IGUANA web site: http://iguana.cern.ch.

Active Configurables

The visual representation of the objects can be customized at run-time either by changing the model, by setting a different parameter for a reconstruction filter and then re-running the reconstruction, or by changing the representation itself, e.g. visualise only those reconstructed objects fitting certain criteria.

Active configurables are implemented based on the Observer pattern to set and change the data cards at run-time. The observed configurable does not know anything about the observers. Instead it "publishes" such changes, thus notifying the observers.

A GUI-based service to create new configurables or modify existing ones can be dynamically loaded at run-time for any COBRA-based application. This service presents the snapshot of all configurables at any given time.

Visualisation Context and Threads

To keep the GUI active most of the time, it runs in a separate thread. COBRA also exploits multiple threads to permit optimal use of resources. The communication layer keeps consistency between the visualisation model in the GUI thread and the COBRA objects in the event and other threads. The visualisation framework creates a context for each thread and defines mechanisms for updating the context. Once again, SEAL callbacks are used to maintain communication among the threads.

The visualisation framework processes the user actions and commands, such as a request for a new event or to visualise a physics quantity, and generates the appropriate callbacks, or threaded command objects, which are queued for further processing by COBRA.

As the threaded command objects are processed, the requested (reconstructed) objects or event proxy are dispatched and asynchronously visualised if thus requested. To keep the application thread-safe, the model functions are not called while the threaded command object is running.

Thread safety issues arise when trying to integrate non thread safe libraries and frameworks. OpenInventor is such an example. Any updates of the OpenInventor scene graph are therefore done within a lock.

MetaData Viewers

Information about the event and its structure is available either as a tree, with a hierarchy of the containers and collections, or graphically on a canvas. This information is dynamic and is automatically updated when the structure changes: from event to event or when new transient objects are created. Once more, Observer patterns proved to be very useful in the implementation.

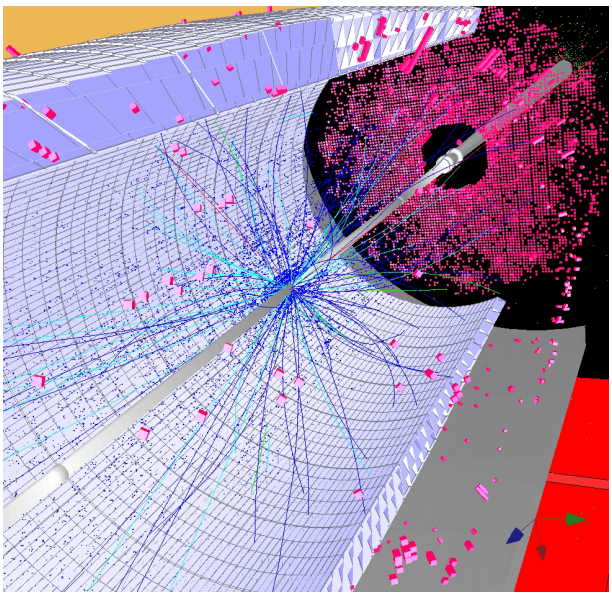

Figure 2: CMS detector and simulated event.

Observers and Other Patterns

The visualisation framework is based on a variation of the Model - View - Controller pattern; a layered MVC - a

triad modules linked by the Observer pattern. The MVC pattern decouples changes to how data are manipulated from how they are displayed or stored, while unifying the code in each component.

The Observer pattern is useful mostly for dynamic relationships between objects: one can hook up a new observer to an observable while the program is running (e.g. hook up a newly-opened viewing window to a domain object), and unhook it later (remove the window from the list of observers when the user closes it).

In the pull model, the observable broadcasts that it has changed, but does not indicate how it changed.

In the push model, the observable broadcasts a piece of information that reveals the nature of the change, e.g. 'a colour has changed to red' or 'added item at index 12'.

In general the use of patterns helps to simplify the development and understanding of the overall architecture. There are some limitations however, e.g. in using exceptions and Observers. The Observer should never throw exceptions. There is no one to catch them.

COMMUNICATION WITH GEANT4 FRAMEWORK

Geant4 visualisation system assumes a rendering pipeline view of graphics thus putting constrains on any other orthogonal to it interactive usage, i.e. incremental rendering.

The CMS visualisation framework needs to allow users to choose which parts of the detector and event are to be visualised, and with what parameters; each change of settings ought to allow for immediate change of the representation. With the current support in Geant4 it is not easy to process or re-process only a particular volume without disturbing the rest of the existing scene graph.

The information provided by the Geant4 framework about the volume processing is insufficient to allow the scene handler to map the representations it is creating to the original volume tree. Although it does provide information, the manner in which it is provided does not allow construction of the scene graph tree. As a consequence, picking is not possible. It also makes it difficult to reuse the representations like Geant4 does with logical volumes. This same issue applies also to the event content: hits and trajectories.

As a result, Geant4 integration into the CMS interactive graphical environment required to partially re-write the Geant4 visualisation framework: the visualisation manager, the Geant4 command line shell to synchronize the GUI and Geant4 threads. The latter runs the command line among other things. Consequently it also synchronizes the exit so that exit from either of the threads triggers the other to quit.

APPLICATION EXAMPLES

Currently several visualisation applications are available at iguana startup. They are divided into two major categories: visualisation for simulation and visualisation for reconstruction. As is desirable from a user's point of view, these two groups of CMS graphics applications are becoming more and more similar. Our goal is to eventually remove the remaining differences.

Visualisation for Reconstruction

Visualisation for the reconstruction project ORCA includes several applications: rec application, sim application and daq application. The type of the application can be set as a data card in a configuration file. Alternatively, one of the applications can be loaded from the GUI at startup. This feature is implemented as an application factory.

Visualisation for reconstruction can use either simulated data as an input collection or digitized data.

DST Visualisation

The visualisation application for the DSTs (Data Summary Tape) is similar to the reconstruction visualisation application with the difference that the input data collection contains only reconstructed objects which are in fact the DSTs (see Fig. 3).

Visualisation for Simulation

There are several types of applications for simulation: persistent ones that write an output of each event in a POOL [9] file and transient ones that do not. Generator types, physics tables, magnetic field, geometry source, etc. are configured at startup. CMS detector geometry described in XML comes from DDD [10] - detector description database. This geometry is converted to the Geant4 geometry and then visualised.

XML Geometry Visualisation

The geometry visualisation application is similar to the simulation application. Any DDD XML geometry description can be read and visualised. Two data cards define the name and the source of the geometry. Definition of the sensitive volumes is CMS specific and is needed in order to use the sensitivity filters.

Test-beam Applications

Applicability of the framework for the test-beams strongly depends on the modularity of ORCA packages. Since the test-beam setups normally have only a sub-set of the sub-detector geometry it is mandatory that corresponding packages from ORCA have clean dependencies, i.e. would not require linking against other sub-detector libraries.

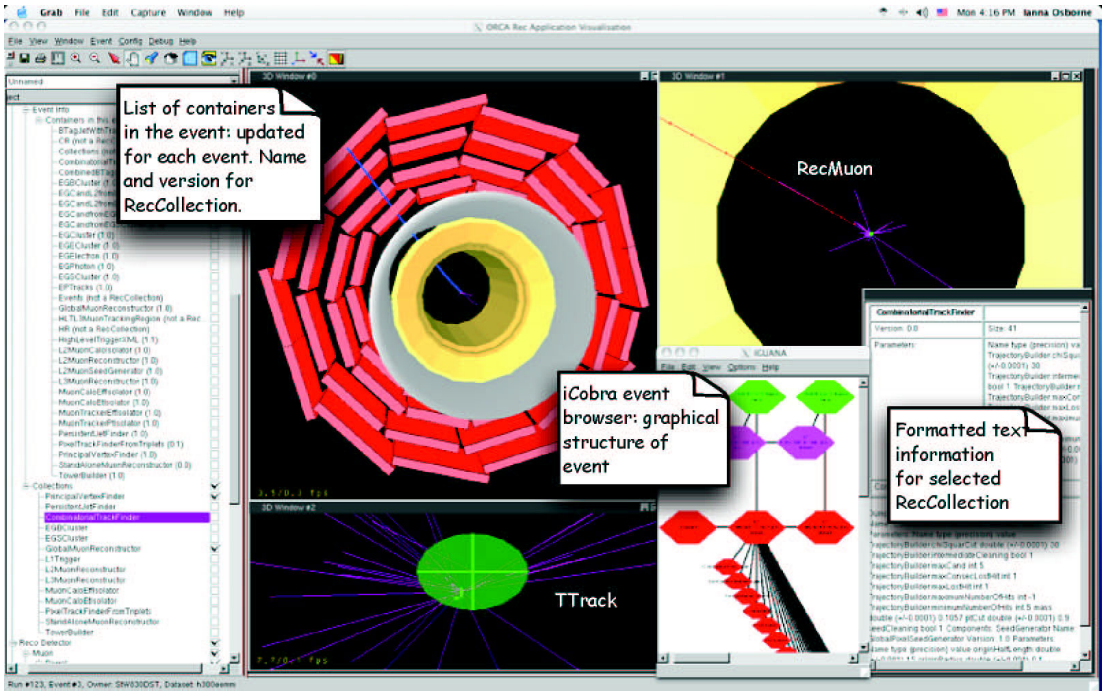

Figure 3: The IGUANACMS DST application with a hierarchical view of the event containers, reconstructed objects and detailed information about reconstruction algorithms.

Successful examples of the test-beam visualisation applications based on IGUANACMS are the muon drift tube test-beam, HCAL test bed, the tracker test-beams.

DAQ Applications

DAQ applications define their input data by adding extra packages in the configuration file. The rest is the same as for the test-beam applicatons.

CONCLUSIONS

The CMS visualisation framework is rapidly extending and the number of the specialized plug-ins is growing. As a result this solution becomes viable for different CMS visualisation needs. The variety of applications based upon the framework shows its inherent flexibility.

The limitations of the framework applicability are defined mostly by the lack of modularity of the external software, these issues have been identified and are being addressed by the corresponding projects.

ACKNOWLEDGEMENTS

This work is supported by NSF.

REFERENCES

[1] P. Mato, et al., “SEAL: Common core libraries and services for LHC applications”, Proceedings of CHEP’03, La Jolla, USA, March 2003.

[2] V. Innocente, et al., “CMS Software Architecture: Software framework, services and persistency in high level trigger, reconstruction and analysis”, Computer Physics Communications 140 (2001) 31-44.

[3] J. Apostolakis, et al., “An overview of Geant4’s recent developments”, Proceedings of CHEP03, La Jolla, USA, March 2003.

[4] http://iguanacms.cern.ch/iguanacms

[5] S. Ashby, I.Osborne, J.P. Wellisch, C. Williams, “Code Organization and Configuration Management”, Proceedings of CHEP 2001, Beijing, China, September, 2001.

[6] G. Alverson, G. Eulisse, S. Muzaffar, I. Osborne, L. Taylor, L.A. Tuura, “IGUANA Architecture, Framework and Toolkit for Interactive Graphics”, Proceedings of CHEP’03, La Jolla, USA, March 2003.

[7] S. Wynhoff, et al., “Using the reconstruction software, ORCA,in the CMS datachallenge 2004”, CHEP’04, Interlaken, Switzerland, September 2004.

[8] S. Abdullin, et al., “An Object-Oriented Simulation Program for CMS”, CHEP’04, Interlaken, Switzerland, September 2004.

[9] D. Duellmann, et al., “POOL Development Status and Plans”, CHEP’04, Interlaken, Switzerland, September 2004.

[10] M. Liendl, et al., “The Role of XML In The CMS Detector Description Database”, Proceedings of CHEP 2001, Beijing, China, September, 2001.

[11] G. Alverson, G. Eulisse, S. Muzaffar, I. Osborne, L. Taylor, L.A. Tuura, “IGUANA Interactive Graphics Project: Recent Developments”, CHEP’04, Interlaken, Switzerland, September 2004.