The Design, Implementation and Deployment of a Functional Prototype OO Reconstruction Software for CMS. The ORCA Project.

V.Innocente[1]*, D.Stickland*[2]
For the CMS Collaboration

(1) CERN, Geneva, Switzerland
(2) Princeton University, Princeton NJ, USA

Abstract

Since September 1998 CMS has built and deployed an OO reconstruction program under the acronym ORCA. This software is built on the CARF architecture, based on action on demand. Using a pragmatic approach, initially wrapping and later replacing legacy code we have increased the team of active developers from a handful to more than 50 at this time. Using an iterative life-cycle model, with macro-cycles of about 3 months we have steadily improved the software quality as the experience of the developers grows.

In October 1999 ORCA was used in a production environment, writing 600k events into two Objectivity/DB Federations, and is currently being used by CMS physicists performing detector optimization and trigger studies. These databases can also be used to study realistic access patterns in user analysis and hence give useful input to LHC resource planning.

Keywords: CHEP, CMS, Reconstruction

1 Introduction

The transition to an Object Oriented (OO) implementation of CMS reconstruction software is now well established. The reconstruction software package has the acronym ORCA, for **O**bject Oriented **R**econstruction for **C**MS **A**nalysis

The long-term reconstruction planning is described in the CMS Computing Technical Proposal. [1]

This note describes the status of ORCA and its intended use in final detector optimization, trigger studies and global detector performance evaluation.

2 OO Transition Strategy

In September 1998 the core team developing software numbered about 6 people. The basic architectural scheme of CARF [2] had been tested in various prototypes and in test-beams. In investigating models for making this transition it seemed clear that a pragmatic approach was required. We did not have sufficient in-depth expertise to do a top-down design of the entire project, and indeed were convinced that this was also not an appropriate way to build a software project of this scale. Rather, it was decided to follow an "iterative spiral" model of development, whereby one implements initially a small part of the project in a rough, but complete way, repeating a cycle of Analysis, Design, Implementation, Testing as the product becomes more complete.

The transition was organized around four major release cycles, the first and last of which were also configured to meet CMS Software and Computing milestones:

- ORCA_1, the initial release of ORCA in December 1998. This implemented basic reconstruction in all detectors at least through to digitization.[1] Complex code, such as track

[1] Whereas digitization is often considered to be part of the simulation process, due to the complications of adding

finders that had been implemented in Fortran, was incorporated by "wrapping" the Fortran in C++ input and output interfaces. This release satisfied the December 1998 milestone requirement of "Proof of concept" for the Detector Reconstruction.

- ORCA_2, released in Spring 1999. Initial OO/C++ implementations of track finders in the central tracker and in the muon system. In addition more detailed digitization, including for example the effects of calorimetric selective-readout and the construction of trigger primitives was implemented. Most of the wrapped Fortran code was removed for this release. The exception being that related to extracting the GEANT3 hit information, and in some cases the GEANT3 geometry. Initial implementations of physics objects, such as jets, were included.
- ORCA_3, released in October 1999. The first release with general persistency mechanism implemented for Calorimeter, Muon, Trigger and Generator information. This release has been used to make a large scale production of reconstructed data, stored in Objectivity/DB for the trigger studies.
- ORCA_4, planned for February 2000. Extension of persistency to all (required) reconstruction objects. In particular Tracker related objects (which were omitted in ORCA_3). A complete reconstruction package that will satisfy the milestone requirements for a "Functional Prototype" of the Reconstruction/Analysis framework and of the Detector Reconstruction itself.

Within each of these major release cycles, there were 2-3 minor releases required to bring in new features, and of course fix bugs.

A secondary emphasis of this plan was to gradually improve the C++, and more importantly, OO skills of an expanded core team. To date about 50 people have contributed code to the ORCA project. In addition, as described below, many of them have followed common courses in OO Analysis and Design. The growth in experience, diverse requirements and abilities of the core team has been an important side-effect of this first years development. This gives us more confidence to embark on the next stages of the ORCA project, optimization and increased formalization of the designs and implementations, in 2000.

3 Project Management

A management group was formed and charged with the implementation of the CMS reconstruction project as defined by the Software/Computing Technical Board, SCTB. This group goes by the acronym RPROM, standing for the **R**econstruction **Pro**ject **M**anagement group. It meets on a weekly basis and RPROM meetings are video-conferenced to reach a wider audience. The use of "free" video-conferencing using the Virtual-Room [3] system is vital to maximize the ability of a widely geographically dispersed group of developers to build a coherent product.

4 Code Repository

In the CMS Software workshop of September 1998 the structure of the code repository was defined. We have adopted a two-level system of:

- Sub-systems. Being code categories such as
  - Tracker
  - Calorimetry
  - Framework

pile-up events to simulated signals and backgrounds, it is more practical to see this as the initial stage of reconstruction of simulated data. Of course in real-data the digitization is *mostly* performed by the front-end hardware

  - Examples
- Packages. Being the modules typically corresponding to a single library, such as
  - Calorimetry/EcalClusters
  - CARF/G3PersistentEvent

  A clean structure is enforced by requiring non-cyclic dependencies between packages.[2]

We use CVS for version control and an in-house product SCRAM [4] for release and build management.

5 Release Strategy

After a number of trials, we have identified a working strategy for a major release that is reasonably efficient. The dependency structure of ORCA, being acyclic allows a layered release sequence. For example, basic sub-systems such as CARF, Utilities etc are released in an initial "pre-release". The code is collated by the librarian and a central release area for source, libraries and binaries is constructed. Developers in the next layer, say Calorimetry and Tracker, test their code against this release, feeding back any necessary corrections occasionally identifying new requirements on the base packages. The next layer of software is then released, until eventually the complete ORCA package is ready for a consistent release. (The actual efficiency of the process can be judged by the fact that whereas there are four independent layers of sub-systems, we actually underwent 6 pre-release cycles to reach the ORCA_3 release.)

A major release period is thus rather lengthy, typically a 1-2 weeks, but it is manageable and the number of iterations as developers break each others packages can be limited. Bug-fix releases, by contrast must be capable of being rapidly deployed, and here we have found it necessary to enforce that such changes are implementation-only, and that interfaces are unchanged.

6 CARF

The CMS Analysis and Reconstruction Framework, CARF is described in detail elsewhere [2]. The framework currently supplies the following functionalities:

- Persistent storage management for both event and non-event data.
- Reconstruction framework: the system that provides the users with the requested reconstructed objects (such as Track, Cluster, Jet...)
- It allows the selection of the Reconstruction Algorithm at Run-time, and the possibility that an event can contain the same type of objects reconstructed with different algorithms (or different parameters) to allow easy comparison.
- Analysis framework: the system that manages the chain of event filters that constitute an analysis cycle.

Communication with the persistent data store is handled by CARF rather than any explicit code in the user packages. CARF is used to manage the reconstruction utilizing a system of reconstruction on demand in which reconstructed objects are only created/accessed from the store, when they are needed.

6.1 Action on Demand

In the past years we have performed studies and evaluations of the architectures used in simulation, reconstruction and analysis programs of various HEP experiments at CERN, Fermilab, SLAC and DESY. This studies have convinced us that the traditional "main program and subroutines"

[2] Tools to identify such cyclic-dependencies have been developed in collaboration between CMS and xxx Institute

architecture and even a more elaborated “pipelined” architecture is not flexible enough to satisfy CMS requirements described above.

To achieve maximum flexibility CARF implements an “implicit invocation architecture”. Modules register themselves at creation time and are invoked when required. In this way only those modules really needed are loaded and executed. Implicit invocation requires some additional mechanisms to construct the required modules, to customize them and to manage dependencies.

6.1.1 Implicit invocation

Modules whose state depends on the occurrence of some external “event” register themselves as “observer” to the “dispatcher” of such an event. In CARF applications typical external “events” are the arrival of a new physics event, of a new simulated event (trigger or pile-up), a new run, a new detector set-up etc. When a new “event” occurs the dispatcher informs all registered observers which update their state accordingly.

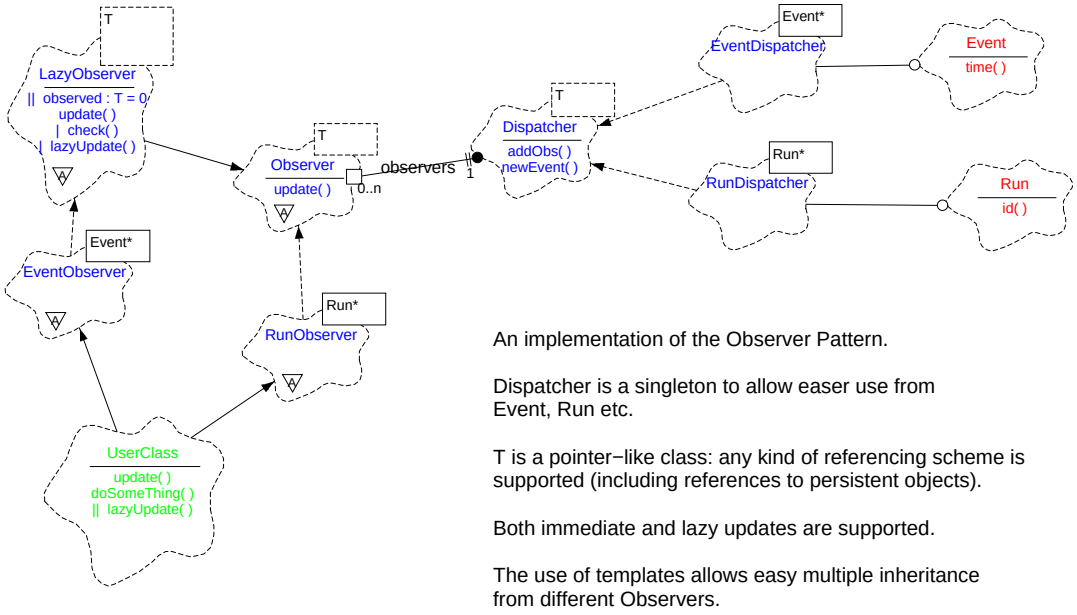

Figure 1: Class diagram showing the classes collaborating in the implicit invocation mechanism implemented in CARF.

In CARF this “Observer pattern” is implemented using template classes with, as template argument, a pointer to the event to dispatch. Figure 1 shows the class diagram of an example of a user class observing physics events and runs. In this example the user class is a “standard observer” of runs (it will take immediate action when a run changes) and a “lazy observer” of events (it will take action only when a user ask it for a service).

6.1.2 Module construction

CMS software is subdivided in “Packages”. A package groups all classes required to perform a given task. A package realizes itself into a shared library. CARF provides a PackageInitializer class which can be specialized in each package. One instance of this class can be staticly constructed when the corresponding library is loaded. Such an object can be used to construct and register default versions of the modules contained in the package. Additional “Singletons” can be used to publicize and export other more direct services.

6.1.3 Customization

A CARF application can be customized in several way:

- at loading time: the act of loading a shared library will automatically invoke the corresponding PackageInitializer. CARF will in future support run time dynamic loading to improve flexibility.
- at compilation time: CARF provides hooks where users can overwrite default initialization and register their own plug-in modules and define their own preferred configuration.
- at run time: through a user interface (at present a simple ASCII file, in future a database driven configuration control system) packages can be instructed to construct and register particular modules in some specific configuration.

6.1.4 Dependency management

To avoid a central management of dependencies all plug-in modules work in a "lazy-update" mode. In such a mode the actual code execution is deferred until a service is required to the object in question. A simple state machine ensures that code is executed once for each "event". A "lazy-update" approach achieves two goals:

- code is executed when and if it is really required
- dependencies are automatically taken into account

The major drawback of the absence of a central management of dependencies is the impossibility to perform a "static check" of the application configuration prior to run time: missing modules and/or circular dependencies will only be detected at run time. It is the responsibility of the application builder to avoid such conflicts. In the CMS reconstruction software ORCA this is done by a static check of the dependencies of the libraries prior to release.

7 Persistent Data Storage

Since the Simulation program we are using, CMSIM, is GEANT3/ZEBRA based, event data is currently read in from the fz format files produced by CMSIM. CARF encapsulates/copies CMSIM-ZEBRA banks in C++ objects managing, at the same time, signal events and minimum-bias pile-up from in-time and off-time interactions. Also the geometry data, created in the CMSIM step, is accessed via an intermediate rz format file. When the simulation migrates to GEANT4, these steps will be obviated.

With ORCA_3 we have implemented the possibility to make persistent most digits and detector objects. We are using Objectivity/DB for this storage. Most code contains no explicit reference to Objectivity classes. CARF itself controls the actual access to Objectivity and one package within each sub-system has the ddl classes to define the structure of the objects that can be made persistent. No user-code has any knowledge of the actual transient or persistent state of the objects it instantiates.

7.1 Persistent object management in CARF

Persistent object management has always been at the center of CARF development: It is considered to be one of the major task of the analysis and reconstruction framework. Indeed already the very first prototypes (notably 1997 test-beam prototype) had already an Objectivity database as key component.

Today, persistent object management is fully integrated into CARF, which manages, directly or through the utility toolkit:

- database transactions;

- creation of databases and containers;
- meta-data and event collections;
- physical clustering of event objects;
- persistent event structure and its relation with the transient one;
- de-referencing persistent objects.

CARF provides also a software middle-layer, mainly in the form of template classes, which helps developers of detector software in providing persistent versions of their own objects.

To avoid transaction overhead event objects are first constructed as transient instances and made persistent (by a copy to their persistent representations) only at the end of the event processing when a decision is finally taken about the fate of the event (send it to oblivion or save it in a particular dataset) and about its classification.

Such a copy is not performed in accessing persistent events: physics modules access directly persistent objects through simple C++ pointers. CARF takes care of all details to make sure that the required objects are actually loaded in memory and that their pointers do not become invalid while the event is processed.

This architecture avoids that detector software developers should become Objectivity experts. Indeed the goal is to make the use of Objectivity completely transparent to physics software developers without making compromises in efficiency. This will also guarantee a painless transition to a new persistent technology if Objectivity proves not to satisfy our requirements.

The present version of CARF offers persistent object management for:

- event structure and meta-data common to test-beams, simulation and reconstruction
- Raw data from test-beam
- simulated particles, simulated tracks and simulated digits
- reconstructed objects common to test-beams and simulation

8 ORCA Production 1999

The requirement to implement persistency now, has come from the Higher Level Trigger (HLT) studies. The processing time for high-luminosity ($10^{34}\mathrm{cm}^{-2}\mathrm{s}^{-1}$) events is too costly to repeat many times. (For example the ECAL pile-up simulation requires the reading of some 200 minimum bias events, about 70MB, for every trigger event)

In addition to deploying the software for ORCA_3 we have also built a production system in which we use 19 high-power CPUS (10 Linux PIII 450-MHz, 7 Sun 300MHz UltraSPARC-II and 2 Sun 400MHz UltraSPARC-II) working in parallel and writing into two Objectivity Federations. We were able to process about 30k events/day in this manner. The setting up of these systems has already given us valuable insight into the sorts of problems that can arise in such a complex (for now) environment.

More than 600k events, with pile-up equivalent to LHC operation at a luminosity of $10^{34}\mathrm{cm}^{-2}\mathrm{s}^{-1}$ were processed and stored in two Objectivity federations. These have been used for Trigger studies and this work is continuing. This and future ORCA production runs will be the basis for the validation of the Higher Level Trigger design and implementation throughout 2000

9 Analysis Tools

To make the results of ORCA usable to a large number of users as quickly and effectively as possible, we are using HBOOK4 objects such as Histograms and both row-wise and column-wise ntuples, which can then be analyzed in the standard way with PAW. This is by no means intended as a long term solution, but represents a perfectly usable and sufficient analysis tool for

our current requirements. Other tools from the lhc++ [5] project, such as HTL are also being used for histograming, both in analysis jobs and in test-beam applications..

10 Examples and Documentation

Most of the prospective users of ORCA have little OO, or C++, experience. It is therefore of paramount importance to supply clear examples showing how typical analysis tasks can be performed in this new environment. The Examples sub-system of ORCA is therefore one of the most important. It also serves as the integration test suite at release time.

Physicists are not renowned for supplying adequate documentation of their software, and the ORCA developers are no exception to this standard. However, the seeds of a documentation system have been implemented. An introductory user guide gives guidance on getting started. We have used the doxygen [7] tool to document the OO structure of the code, and both these and the user guides are deployed on the ORCA web site `http://cmsdoc.cern.ch/orca`

11 Training

The initial emphasis in training over the past year has been to enhance the skills of a much larger core developer group. To this end we have organized a series of OO and C++ related courses at CERN. Similar courses have been organized in Italy and at Fermilab. In addition to these CMS specific course many developers/users have followed regular courses at CERN and elsewhere.

The training requirements of users and of developers may be somewhat different. Developers must eventually have considerable expertise in OO Analysis and Design - as well as of course practical implementation abilities. The quality of the code they create will have direct bearing on the long term utility and maintainability of the code. There are very few developers within HEP with such a high level of expertise at present. Accordingly it is imperative that a well established plan of Software Improvement is established [6]. In addition, we have recognized the need to bring in software engineers at an early stage of the project, and throughout its development, both to take responsibility for some of the key underlying software foundations, and to offer expert advice and assistance to physicists developing code for specific applications. “Mentoring” is well established as probably the most effective way to enhance the skills of would-be developers.

The training of new users is in principle a different task. They may or may not come with some C++ experience, but even if they do, they need to understand the specific mechanisms, the architecture, of CARF and ORCA so that they can make effective use of them. We have experimented with tutorials in CARF/ORCA and in December 1999 almost 50 new users followed an interactive introductory tutorial.

Given the recent very rapid development of ORCA functionality it has been difficult to give such tutorials often enough and to keep them up to date. We believe that with the ORCA 3 release we have completed this stage of rapid change and achieved the relative stability required by users.

12 Remote Deployment

ORCA has been installed and is running at various remote sites in France, Italy, the USA and Russia. This is in addition to those users at remote sites who use the central CERN installation for their code development.

A key element of this remote deployment has been the development of the SCRAM [4] software release tool.

A further key to allowing effective remote deployment is the RPROM meeting which is transmitted on VRVS video and typically viewed by participants at 4-6 remote sites. Fermilab has

set up its own series of software and ORCA-user meetings so as to be better able to satisfy the requirements of US users. Since typically software development is performed not by large remote teams, but individuals at many different institutes the requirement for remote collaborative tools is significant.

13 Completing the Initial ORCA Project

The ORCA_4 release is intended to meet CMS requirements for a "Functional Prototype" reconstruction package. Within the ORCA code itself this means completing all sub-detector digitizations and candidate reconstructions for higher-level objects such as Calorimetric-Clusters, Jets, Tracks, Muons...

Compared to ORCA_3, there must be a substantial increase in sophistication of the persistent storage. In ORCA_3 the objects for "complete" events were stored within a database file, one run per file. For ORCA_4 a much more complex structure is envisioned, in response to more sophisticated requirements.

In order to keep the number of database files finite, each database file will contain objects from many different runs, and to allow efficient clustering of the data each event will have its objects distributed over many different database files. Information for a given "event" will be stored in 6 or more separate database files:

- (1) GEANT3/CMSIM Geometry structure stored as a "blob" for each setup
- For each minimum-bias and signal event:
  - (2) Generator information
  - (3) Simulated Hits for Calorimeter and Muon
  - (4) Simulated Hits for Tracker
- (5) Digitizations (being the result of Pile-up of up to 200 minimum-bias events and 1 signal event)
- (6..N) Reconstructed objects, Clusters, Tracks, Jets etc

For it to be possible to navigate this structure from within one ORCA job (or other tool) all these database files must appear to be members of a single federation, yet to manage the production and wide-area distribution within a single actual federation is unwieldy. We will simulate a single federation by requiring a unique schema for all actual federations and preassigning database-ID's to ensure that they are unique across the actual federations.

Since the production facilities and users are distributed throughout the world, we will implement catalog and database export/import with local mirroring.

Many of these features had been foreseen, in the CTP, for later implementation, however the actual requirements coming from a rapidly expanding user-base have increased the urgency of their deployment.

The next ORCA production, scheduled for Spring 2000, will populate ~ 2 TB of Objectivity data in a complex layered structure. Part of the federation will be mirrored at remote sites and indeed some production will also be carried out at remote sites. In addition to satisfying the collaborations requirements for trigger/physics studies, this dataset will also provide a realistic environment in which to test data access and other features of the computing model.

14 Future planning of the ORCA project

With ORCA_4, we will have deployed a functional prototype of CMS Reconstruction that will be meeting many of the requirements of the collaboration. It is intended that the basic functionality in all sub-systems will be present and the initial goals of the ORCA project will have been realized.

We have therefore decided to embark on a major program of appraisal, design improvement, design documentation and re-implementation where necessary.

In the 2nd quarter of 2000 we plan to hold sub-system design reviews both internally and between sub-systems, with the goal of accumulating the design and implementation documentation that would be required by an external formal review. We plan to undertake such formal review of the key elements of the CARF and ORCA software in the 2nd and 3rd quarters of 2000. This increasing formality is required to assure the long term performance and maintainability of ORCA. This will also have the effect of increasing the awareness of good design and implementation practices amongst the developers. It is hoped that the inter-subsystem design reviews will increase the coherence of the overall software package by exposing developers to solutions employed by other developers.

15 Summary

In one year the ORCA project has gone from being a plan to a reality. Almost every feature of the detector reconstruction is implemented at some degree of detail. In many cases the ORCA code contains much more correct and detailed implementations than earlier Fortran code. A steadily increasing base of developers have contributed code; about 50 people have been active in ORCA development to date. With the latest release of ORCA it has become a production tool. Undoubtedly much of the current implementation will be replaced as our understanding of the problems and solutions improves. Formality will have to play an increasing role, as will issues of optimization.

The ORCA project is well established, it is being used for detector optimization and trigger studies, it is testing our architectural models of the software and it is beginning to give us experience with the computing model.

References

1 CMS Collaboration: *Computing Technical Proposal*, CERN/LHCC 96-45, (Geneva 1996)

2 V.Innocente *CMS Software Architecture: Software framework, services and persistency in high level trigger, reconstruction and analysis* CMS IN/1999-034.

3 The Virtual Room Service `http://vrvs.cern.ch`

4 C.Williams `http://cmsdoc.cern.ch/Releases/SCRAM/current/doc/html/index.html`

5 *Libraries for HEP Computing – LHC++*, `http://wwwinfo.cern.ch/asd/lhc++`

6 J-P.Wellisch *Status of Software Process Improvement in CMS* CMS IN/1999-033.

7 The doxygen documentation system. `http://www.stack.nl/~dimitri/doxygen/index.html`