The STAR offline framework

V. Fine, Y. Fisyak, V. Perevoztchikov, T.Wenaus

Brookhaven National Laboratory, USA

Abstract

The Solenoidal Tracker At RHIC (STAR) is a large acceptance collider detector, commissioned at Brookhaven National Laboratory in 1999.

STAR has developed a software framework supporting simulation, reconstruction and analysis in offline production, interactive physics analysis and online monitoring environments that is well matched both to STAR's present status of transition between Fortran and C++ based software and to STAR's evolution to a fully OO software base. This paper presents the results of two years effort developing a modular C++ framework based on the ROOT package that encompasses both wrapped Fortran components (legacy simulation and reconstruction code) served by IDL-defined data structures, and fully OO components (all physics analysis code) served by a recently developed object model for event data.

The framework supports chained components, which can themselves be composite subchains, with components ('makers') managing 'data sets' they have created and are responsible for. An *St_DataSet* class from which data sets and makers inherit allows the construction of hierarchical organisations of components and data, and centralises almost all system tasks such as data set navigation, I/O, database access, and inter-component communication.

This paper will present an overview of this system, now deployed and well exercised in production environments with real and simulated data, and in an active physics analysis development program.

Keywords: OO, Fortran, C++, ROOT, dataset, hierarchy

1 Introduction

The new generation of HENP experiments such as those at RHIC and LHC must contend with processing and mining heretofore unprecedented amounts of data with highly complex analysis software developed and used by large worldwide communities of physicists. Object oriented (OO) programming has been identified and adopted by these communities as an efficient and powerful approach to developing capable, robust, maintainable software in this environment.

Two years ago STAR started to develop a software framework supporting simulation, reconstruction and analysis in offline production, interactive physics analysis and online monitoring environments to support STAR's transition from Fortran to C++ based software and to support the fully OO software base STAR is migrating to[1].

The framework dictates [2] the architecture of the application. It defines the overall structure, its partitioning into classes and objects, the key responsibilities thereof, how the classes and objects collaborate, and the thread of control. The framework predefines these design parameters so physicists can design their solutions using a proven programming model and can concentrate on the specifics of their applications.

We understood that a framework would not simply materialise by using object-oriented techniques. The framework requires a lot of attention if it is going to be successful.

This paper presents STAR's C++ ROOT-based class library [3] and STAR production framework as "a set of cooperating classes that make up a reusable design for a specific class of software" [2].

2 STAR C++ ROOT-based class library

The general view of STAR reconstruction framework as "a set of cooperating classes that make up a reusable design for a specific class of software" is present on **Figure 1**. OO model of STAR reconstruction chain is described in terms of the St_DataSet C++ class. This class is used to describe the hierarchy of the data as well as the hierarchy of the program flow control of the reconstruction code functional modules.

St_DataSet **object ::= the "named" collection of** *St_DataSet* **objects,**

where the "collection" (the pointer to ROOT collection object) may contain no object.

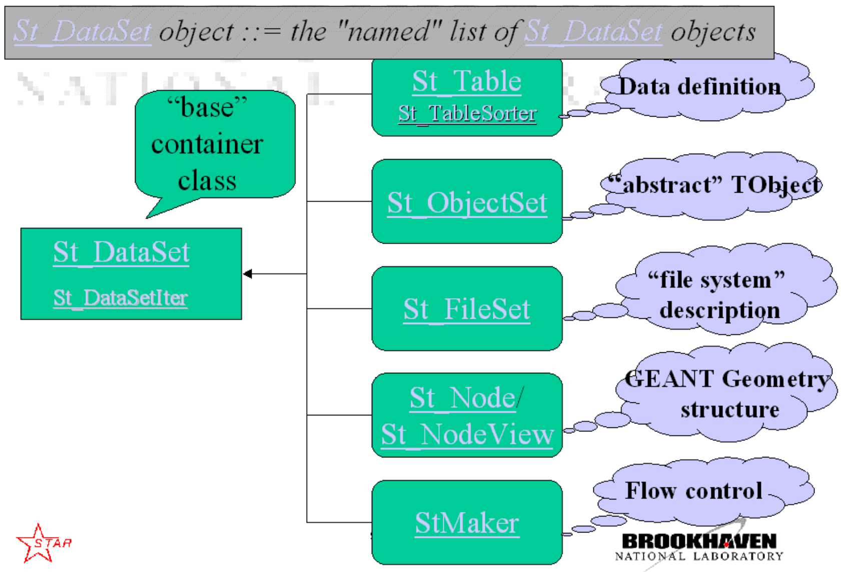


Figure 1: OO model of STAR reconstruction / simulation chain

St_DataSet class is a base class to implement the directory-like data structures and maintain them via the *St_DataSetIter* class iterator.

The *St_DataSet* object has a back pointer to its "parent" *St_DataSet* object if any, and the "character" *name*, and "character" *title*.

The service which this class does provide is to help the user to build and manage the hierarchy of his/her data. For example the method St_DataSet::Update allows use some St_DataSet object to update another St_DataSet. This is useful when only a fraction of the entire structure has to updated within production chain looping over "events". Method "Shunt" allows change the relationships of some particular St_DataSet object with others etc.

Figure 1 shows the main classes derived from the St_DataSet one to build the 00 model of STAR off-line framework.

St_DataSet can be iterated using an iterator object (see *St_DataSetIter)* or by *St_DataSet*::Pass method (see below) and allows us to introduce and change the following relationships between its components:

- *Dataset Member.* Any object from the list above is called **"DataSet Member"**

- *Structural member.* The “Dataset Member” is its **“Structural member”** if its “back pointer” points to this object
- *Dataset Owner (parent).* We will say this *St_DataSet* object **“owns”** (or is an **owner / parent** of) another *St_DataSet* object if the last one is its “Structural Member”
- *Associated member.* If some object is not “Structural member” of this object we will say it is an “Associated Member” of this dataset
- *Orphan dataset.* If some dataset is a member of NO other *St_DataSet* object it is called an “orphan” dataset object

This schema is not ideal but instead of fighting for “purity”, we created a framework that could be used.

2.1 Classes St_Table, St_TableSorter, St_TableIter

St_Table class is a base class-wrapper to maintain the various arrays of the plain C-structures - tables.

The initial purpose of this class was to provide a tool to migrate from the StAF (Standard Analysis Framework) employed by STAR on the early stage of the project [1].

The table is very convenient object to save the huge amount of experimental data and manipulated. All known databases can manipulate with tables. It is much more simple to resolve “schema evolution” problem for the “plain” tables (“plain” stands for the arrays with no pointers to another object / structures). It can be used in mixed code environment with Fortran / C / C++.

Class St_Table is generated for each table and is supplied with two class-companions, namely with St_TableSorter class to sort the table rows by various foreign keys, and St_TableIter to loop over the sorted table rows.

The present STAR framework provide several I/O formats to save/restore these objects. There are xdf, root, plain ASCII and MySQL formats. The I/O format is chosen by the steering code. Usually the users code is not aware about I/O format and is not affected when some format is changed or new I/O schema is introduced.

Since St_Table’s are subclasses of St_DataSet it is easy to combine them in the various hierarchical structure. This way we compensate lacking of pointers within St_Table objects (**Figure 2**).

2.2 Class St_FileSet

Class St_FileSet is to map a real “file directory tree” of the platform we are running on to the St_DataSet instance in memory. Each “pure” St_DataSet object is related to the real directory of some file system (UNIX or Windows for example). The extra data-members of the the subclasses of St_DataSet (St_Tables for example) can be converted into the “plain” ASCII ROOT macro with the regular TObject::SavePrimitive method. Such approach allows people to get access to either piece of the “naive” data-base, create, edit and play with the same way people do with the software code. In other words any user can create a prototype of the database he keeps in mind. The user of the STAR framework “by definition” has skill to use the “regular” shell commands and C++ classes like St_DataSet and St_Table and designing this “naive” database doesn’t require any extra knowledge or a special skill.

Every one is free to use for his / her particular task either the “central” data base - “CVS repository” or one can “check out” CVS stuff and adjust it for the current needs with the trivial UNIX / Windows commands like “mkdir” / “rmdir” and a “plain text editor”.

This class with the “regular” CVS and AFS facilities allows to created a “naive” but powerful database. AFS provides word-wide access to this “database” and CVS supplies the “house-

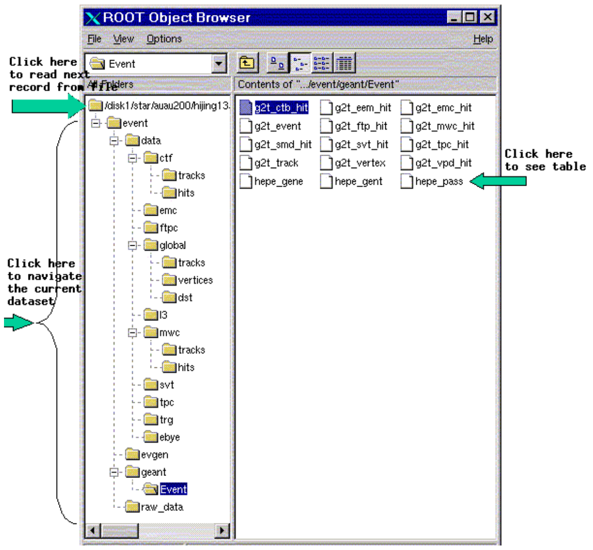
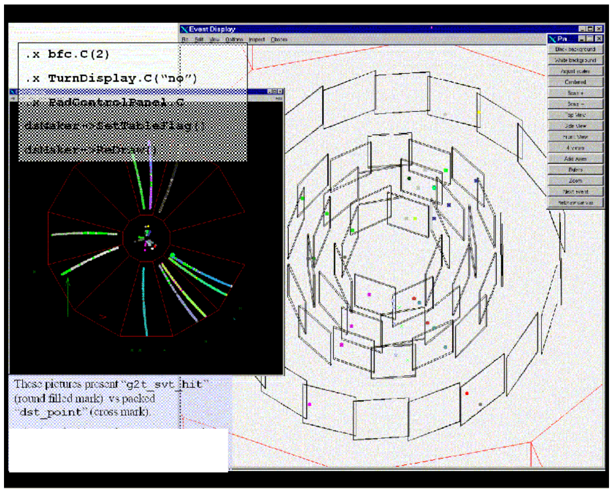


Figure 2: STAR data-structure via ROOT object browser.

keeping".

2.3 Class TVolume/ TVolumeView

These set of classes is to provide tools to access the detailed knowledge on detector implemented into GEANT3 simulation model [4]. The instance of this class are in use by various piece of the reconstruction and simulation chain and provide the detector geometry for "Event Display" classes (see: **Figure 3**)



```
.x bfc.C(2)
.x TurnDisplay.C("no")
.x PadControlPanel.C
dsMaker->SetTableFlag()
dsMaker->PadDraw()
```

These pictures present "g2t_svt_hit" (round filled mark) vs packed "dst_point" (cross mark).

Figure 3: A snap-shot of the TPad "view" of STAR dataset object combining TVolume ("detector" geometry) and St_Table ("event" geometry) objects

2.4 Class StChain/StMaker

StMaker is a base class to describe one single step (maker) of the chain of the makers controlled by the object of the StChain class. Where StChain is a subclass of StMaker. The instances of StMaker classes works out a dedicated branch of the general dataset, has an access to the rest dataset tree and can use the legacy Fortran / C simulation and reconstruction modules via special class ”module wrapper”.

3 Conclusion

Today the STAR reconstruction chain is defined with as a single instance of StChain class. It holds the instance up to 56 subclasses of StMaker class. Each instance of StMaker class included into StBFChain is to provide OO definition of the simulation /reconstruction /analysis domain. All together they create about 300 different instance of St_Table class involving 156 legacy Fortran /C modules. Thanks the framework either part of the chain can be splitted by fractions and saved / restored using different I/O formats by demand. The table shows how the usage of the C++ code vs Fortran has been changed for year.

Table I: Trend in usage of OO technology in the STAR off-line software since CHEP'98 (in kLoc (kLoc = 1000*(Line-Of-Code)))

Language	CHEP 98	CHEP 2000	2000/98
C++	**27**	**138**	**5.1**
FORTRAN	**90**	**68**	**0.75**
MORTRAN	28	34	1.2
C	20	24	1.2
IDL	10	13	1.3
ROOT macros	-	11	-
KUIP	22	4	0.2
Total kLocs	**197**	**292**	**1.5**

The present framework has being testing for the last year. It was used to produce 100 GBytes of DST from 3 TBytes of the GEANT-simulated data. It is proved it allows the construction of hierarchical organisations of components and data, and centralises almost all system tasks such as data set navigation, I/O, database access, and inter-component communication.

It has been adopted as the official STAR base framework.

References

1 V.Fine, et al. “Steps Towards C++/OO Offline Software in STAR”, CHEP'98, Chicago, Autumn 1998. (`http://www.hep.net/chep98/181.html`)

2 Erich Gamma,et al. “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley Pub Co, 1995.

3 V.Fine, “STAR C++ ROOT-based class library”, US HENP ROOT Workshop, Chicago, March, 1999.

4 V.Fine, P.Nevski, “OO model of STAR detector for simulation, visualisation and reconstruction”, CHEP'2000, Padova, 2000.

5 Y.Fisyak, “ROOT in STAR”, US HENP ROOT Workshop, Chicago, March, 1999.