Maker and I/O Model in STAR

Victor Perevoztchikov

Why?

STAR framework(root4star or StarGate), based on maker schema was developed several years ago.

It is working well a long time. Why I reporting about it now?

5. Documentation unfortunately is still bad or absent;
6. We have a lot of new physicists in STAR who are not really familiar with STAR framework;
7. Because of reasons above , the new code more and more often contradicts to STAR framework “ideology” and to support it becomes more and more hard.

Main Concepts

- ROOT as a basic tool for framework;
- High level of modularity;
- Hierarchy, tree of modules;
- Data and methods separation;
- Data and modules in one structure;
- Structured data cash;
- Steering – ROOT macro;
- ROOT I/O with schema evolution;

Glue Class TDataSet

The main class supporting STAR framework is a TDataSet. It was developed in STAR with name StDataSet. Later accepted in ROOT as TDataSet. The class is simple, and very similar to UNIX directory/file system.

- Everything important for framework is inherited from TDataSet. Tables,StEvent,i/o branches and modules==makers are inherited from TDataSet.
- STAR framework structure is a TDataSet tree, where all framework objects are placed, in some specific order. This structure contains the data objects and the methods(makers) together;
- From the ROOT prompt, user can browse them and can invoke the methods of each object manually. It is very important for debugging and understanding the structure.

STAR Framework Tree Organization

The organization of this tree is very simple:

- The top level object is always maker;
- Under it the system directories (nodes) with names: ".make", ".data", ".const", ".garb", etc…
  - ".make" node contains subordinated Makers, which in their turn contain the same substructure as the top Maker, and so on;
  - ".data" node contains list of Data objects, belonged to this Maker and deleted automatically before each event;
  - ".const" same as ".data" but objects are deleted only by user decision;
  - ".garb" same as ".data" but deleted automatically after return from StMaker:Make() method;

The deep of this structure is unlimited. All the nodes in this Tree are equal, but the Makers are "more equal".

User Makers

User Maker must be inherited from StMaker (which in his turn inherited from TDataSet).
Methods:

- The main method is Make() which usually must be overloaded by user. When user in his Make() calls StMaker::Make(), this triggers the call Make() of all subordinated Makers, and so on.
- Methods Init(),Clear(),Finish() could be overloaded too. All these methods also trigger calls to the same method of subordinated Makers, and so on.
- Method GetDataSet("DataObjName") returns pointer to Data object produced by another Maker.
- Method AddData, AddConst and AddGarb add user objects to maker subdirectories ".data",".const",".garb" respectively. After this user objects are accessible by other makers by GetDataSet(…);
- Method GetDataBase("name") returns pointer to Data Base object, already updated for given time stamp;

Makers Rules

Any application is constructed from makers. As a result we have a Tree of makers and Data objects. Creation of this Tree is based on loading of shared libraries related to concrete makers. To do it in a flexible way, some limitations are imposed to Makers:

Independency of Makers:

- Maker is independent from other maker. It does not know which makers are now in the Tree.
- An exception in case of subordinated makers. Parent maker could be dependent of child makers. Not vice versa.
- Adding or removing of Maker to/from Tree, must not lead to code changes in any maker.
- Changing of code in one maker should not lead to recompilation or especially modification of another maker.

Maker Rules (Continued)

- Dependency of Makers:
  - Maker depends from Data objects. But again, Maker must not know, which maker produced the input Data objects, and which one will use Data objects made by him.
  - Maker depends from the third party classes. Like common for experiment or detector or group classes.
- Violation of rules: method GetMaker() allows to find a maker by name. Soon some makers started to do direct call of methods of other makers. Flexibility, modularity decreased. Debugging becomes more and more hard.

Data Cash

Data objects in STAR frame work are kept in Data Cash. Methods:

1. **StMaker::AddData(TDataSet *ptr) adds produced object into Data Cash;**
2. **TDataSet *ptr = StMaker::GetDataSet(name); requests object from Cash by name;**

Data Cash in STAR is structured and based on Maker Tree. Search by name is not arbitrary. The strategy is:

•Search data from sub-makers. If not:

•Search data from the same level makers, already processed.

•Search from upper processed makers;

•Etc…

This allows to resolve name clashes. Maker gets data from the nearest maker sub-tree. It allows to organize the different streams of data with the same name. Makers do not need to account it in code.

Data Object Name

Each Data Object has a name, which is used in search. It is desirable that this name is unique. But it is not always possible. There are problems with the name search:

- The search strategy gives the wrong object with correct name;
- The name was changed, but requested name in maker still the old one. To modify the code is a bad idea. This maker could be used in other Maker Tree with this name;
- User creates a new maker, but he does not know which name of object will be used in a future application.

All these problems are solved by Logical Name approach. The method GetDataSet(name) actually use not real name but a logical one. By default, these names are the same. But this could be changed by.

•StMaker::SetAlias(logName,actName); This method is called in steering part, where Makers are instantiated. So no need to change the code and all problems are solved.

System Makers

STAR framework is a simple framework. All the work provided by makers only. So there are user Makers and system makers. System makers:

- StBFChain is a top maker for reconstruction. It creates a maker tree using set of flags.
- Set of I/O makers: StIOMaker,StTreeMaker, StDAQMaker. StIOMaker is a top I/O maker. Depending of the file type it calls appropriate I/O maker.
- StGeantMaker (St_geant_Maker) reads GEANT data. It can also to run real GEANT;
- StDBMaker(St_db_Maker) request data from MySql DB or from set of UNIX directories;
- StEventMaker: collects data objects and creates the basic STAR Data Object - StEvent.

With except of StBFChain and StGeantMaker, all these makers could have several instantiations in the maker Tree.

Conclusions

- STAR framework is in good shape and working well;
- Documentation is still bad;
- There is a danger of violation it's basic principles. As a result could be decreasing flexibility, debugging problems and instability. This report is an attempt to improve the situation.
- The feedback from users is very appreciated.