

## Evolution of the STAR Framework OO model for the Multi-Core era

---

### Valeri Fine<sup>1</sup>

Brookhaven National Laboratory  
PO Box 5000, NY 11973, USA  
E-mail: [fine@bnl.gov](mailto:fine@bnl.gov)

### Jérôme Lauret

Brookhaven National Laboratory  
E-mail: [jlauret@bnl.gov](mailto:jlauret@bnl.gov)

### Victor Perevoztchikov

Brookhaven National Laboratory  
E-mail: [perev@bnl.gov](mailto:perev@bnl.gov)

With the era of multi-core CPUs, software parallelism is becoming both affordable as well as a practical need. Especially interesting is to re-evaluate the adaptability of the high energy and nuclear physics sophisticated, but time-consuming, event reconstruction frameworks to the reality of the multi-threaded environment. The STAR offline OO ROOT-based framework implements a well-known "standard model" composed of chained modules, where input for each module is the output of the other modules. At its basic principle, modules do not communicate with each other directly and act as consumers and providers of data structures. They use the framework via the special "query" / "publish" API to query the presence of the input data and publish the output results the modules produce. As a proof of principle, we will show that by complementing the base framework with the ability to start several modules in parallel and synchronize the global data access between the "consumer" modules and "producer", one can transparently enhance the existent packages to leverage the multi-core hardware capability. Such conceptual design modification should allow re-using the existing offline software, initially built with single thread architecture and embarrassingly parallel processing in mind, in the multi-threaded environment if needed. However, we realize that the "query"/"publish" paradigm is not sufficient to run the multi-threaded application effectively. It should be complemented with an API to "register" the module output to notify the framework members about an output dataset "to be produced soon". With such addition, the receiving module thread can be automatically suspended if the data it has requested is not ready yet. We will explain how the STAR Offline Framework was modified to test the present approach through building sophisticated interactive real time applications.

*XII Advanced Computing and Analysis Techniques in Physics Research  
Erice, Italy  
3-7 November 2008*

---

<sup>1</sup> Speaker

## 1. Introduction

The Solenoidal Tracker At RHIC (STAR) [1] is a large acceptance collider detector which started data taking at the Brookhaven National Laboratory in summer 2000.

At the time, STAR developed a single all-purpose yet modular ROOT-package-based Object-Oriented framework for simulation, reconstruction and user analysis in offline production also providing the capability to perform interactive physics analysis or online monitoring [2]. It implements a well-known architecture model composed of chained modules, where input for each module is the output of the other modules. Such scheme did fit well the single thread batch environment and every unit of work was organized in a tree-like sequence. In recent times, the evolution of the hardware and CPU architecture has made multi-core and multi-CPU node not only affordable but a response to a growing computing power demand facing limitations in power consumption, cooling or even environmental aspects such as floor space. Under such new reality, it is especially interesting to re-evaluate the adaptability of the STAR framework and similar high energy and nuclear physics sophisticated, but time-consuming, event reconstruction applications to the reality of the multi-threaded environment.

Base principles for a proven and production-mode framework build over a development cycle of the order of ten years implies that any appealing solution requiring changes of the core code and algorithms by STAR scientist are not acceptable. In other words, any solution for mature experiment such as STAR must be backward compatible and “transparent” to minimize development cost, further time investment and possible disruptions in productivity.

## 2. STAR Framework OO Model

The STAR framework was designed to support chronologically chained components, which can themselves be composite sub-chains, with components called “makers” (data object owners) managing named “datasets” they have created and are responsible for [3].

“Makers” and “datasets” inherit from the “TDataSet” class which supports their organization into hierarchical structures for management.

“TDataSet” also centralizes almost all system tasks:

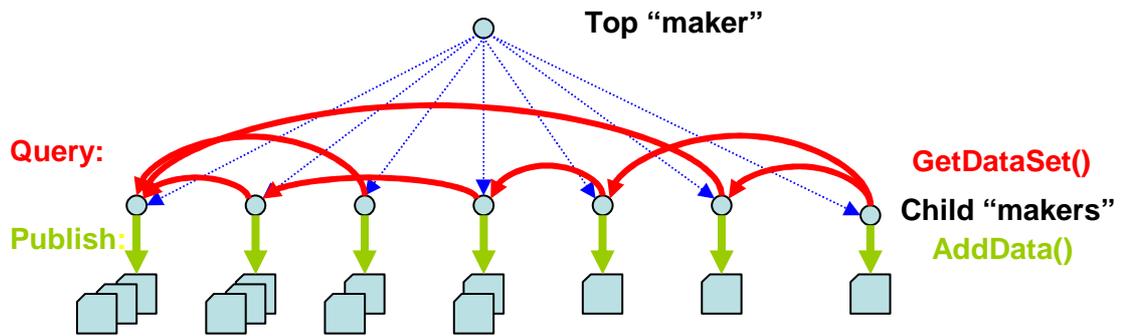
- data set navigation,
- I/O, database access,
- inter-component communication.

The main features of the STAR framework design:

- its “makers” and “datasets” share a common object model
- they derived from one and the same base “TDataSet” class
- the entire STAR reconstruction chain is a single instance of the “StChain” class (subclass of “TDataSet”)

At its basic principle, modules do not communicate with each other directly and act as consumers and providers of data structures. They use the framework via the “query” function implemented by the method “StMaker::GetDataSet” and the “publish” function implemented by

the method “StMaker::AddData()” which is intended to query the presence of input data and publish the output results that the module has produced. The details of STAR implementation were published several times [2-4]. Figure 1 shows the two levels schema of STAR production chain. The parent – child makers (gray circles) relations are indicated with the blue dotted arrows. The figure shows that each child “maker” is provided with dedicated private data-structure (gray folders) to publish (green arrows) the result and each maker can query (the red arc arrows) the private result of the other members of the chains via “GetDataSet” interface.



Each maker can **query** the data **published** by the other makers.

Figure 1. STAR framework "Query" (in red) / "Publish" (in green) modules communications [3].

### 3. The “transparent” steps towards multi-core era

“By design” each framework module can work independently of any other modules as soon as its query and demands for data-sets are satisfied. This suggested that it was possible to “transparently” enhance the existent packages to leverage the multi-core hardware capabilities. To do that it should be sufficient to

- complement the “TDataSet”-base framework with the ability to start several modules in parallel
- provide some facility to synchronize the global data access between the "consumer" modules and "producer“

#### 3.1. Evolution – main directions

The average “reconstruction chain” comprises of four main steps namely, “Init”, “Clear”, “Make” and “Finish” corresponding to respectively a main initialization step, an event loop (a method which will be called for each “event”), and a garbage collection or cleanup needed to be done upon completion of all event processing. Since the first and last steps of the chain are invoked once during a regular production session and the “Event Loop” is called as many times as we have events, one should expect a significant performance gain mainly from the parallelization of the “Clear/Make Event Loop”. For the time being, the first and the last steps can be left unchanged and single-threaded.

To go further we are concentrating our effort to implement and test the crucial components so that they can offer “transparent”

- parallelization, synchronization
- registration

### 3.1.1 “Transparent” parallelization and synchronization

To introduce the parallelization and synchronization of the “Make()” methods called from for the different “makers” of the STAR production “chain” and this in a “transparent” manner, we found it is enough to

- re-implement the methods “Init()”, “Clear()”, “Make()”, and “Finish()” in STAR’s StMaker base class;
- add the extra thread-related private data-members to the base classes. Especially, an additional data-member for the thread ID is needed to distinguish the object owned by the different threads;
- make methods “atomic”;
- allow generating or using as many threads as there are “makers” in the “next level of hierarchy” of the “StChain” class instance instead of calling the methods in loop.

Even though the average STAR chain consists of hundreds [3] of different “makers”, the proposed solution does not lead to an application with hundreds of concurrent threads. Within the STAR framework, the change of the hierarchical level does not change the ability to query the data and does not require changing any low-level implementations either (Figure 2).

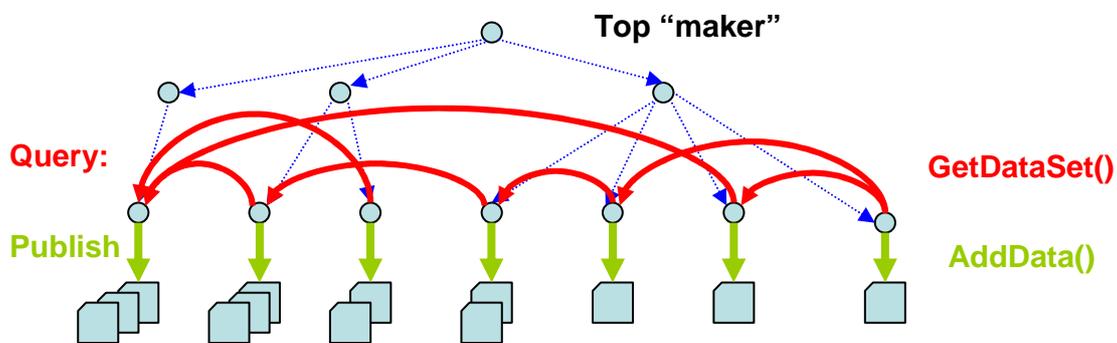


Figure 2. Adding the extra level of the “maker” hierarchy to minimize the number of the threads.

To get the optimal number of the threads (Figure 2 shows three “maker” threads) and avoid having as many threads as makers on a given chain (with potential adverse due to resource consuming), a simple solution is to have an additional layer of makers handling a “next level of the hierarchy” reducing the demand for simultaneous threads. The communication via this additional layer does not require changing any user “maker” code hence “transparent”.

### 3.1.2 “Transparent” registrations

It is noteworthy to realize that a "query"/"publish" paradigm is not sufficient to run the multi-threaded application effectively. Such model should be complemented with an API to “register” the module’s output data sets, allowing for automatically suspending or resuming other modules waiting for the registered “datasets” to be produced. As illustration of this principle, let us study Figure 3, which shows a simple two “makers” chain communication in a single thread model. “Maker 1” produces data that “Maker 2” needs to complete its job. Since they are executed in sequence, there are no synchronization issues and all provider/consumer relations are implicitly self-consistent.

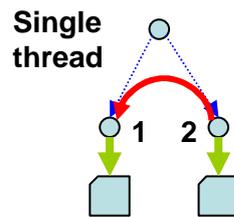


Figure 3. The “maker” commucation within single threaded chain.

Figure 4 shows that the same “chain”, if parallelized, may produce the wrong result since “maker 2” can query “maker 1” for its consuming data before “maker 1” had the time to provide it (generate it). Whenever “maker 1” finally publishes its result, “maker 2” has already provided the wrong result that is, its own provider’s data set not based on nor built from the expected output and information from the first “maker”.

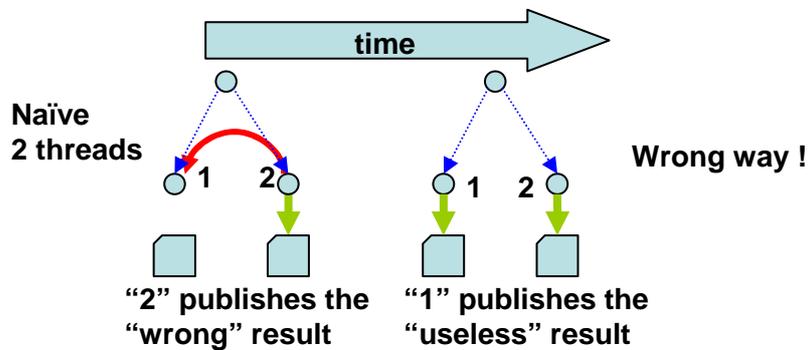


Figure 4. An example of the race condition within a multi-threaded chain.

Figure 5 illustrates that the introduction the additional facility and API to register the data set within the “Init()” method resolves the ambiguity. This is because the thread belonging to “maker 2” can be automatically suspended and then resumed as soon as the “maker 1” produced the “registered” data.

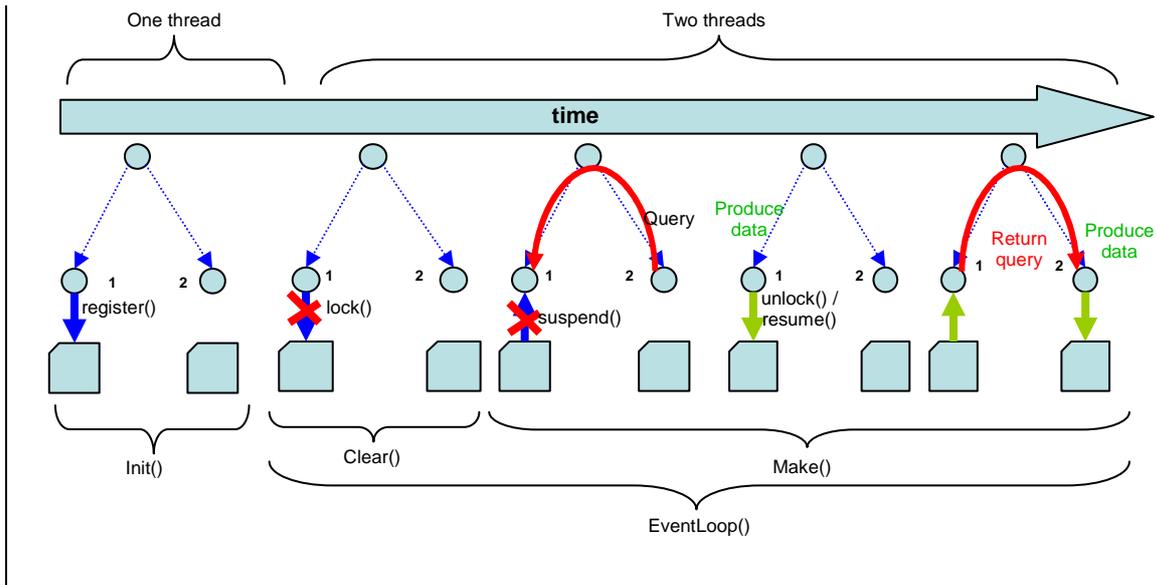


Figure 5. Synchronization between two threads using the registration API.

During the “Init” step, “Maker 1” registers and then locks its private data folder using the method “Clear()” leveraging thread synchronization techniques (see: Figure 5, blue arrow), for example “mutex”. As result, the query invoked from the concurrent thread by “maker 2” cannot be returned immediately. As soon as “Maker 1” produces and publishes its data it unlocks the “mutex” to awake the waiting “GetDataSet” query.

#### 4. Proof of principle implementation

To test the present approach, the STAR Offline Framework was modified to build an interactive real time online application. An online application was chosen as the mandatory real-time requirement (quick response and rapid interactivity) of such application justifies the additional effort needed to transition to a multi-threaded approach. Additionally, such application would be tested on a 24x7 basis during data taking.

The first version of the “parallel” chain concept was used in the context of the STAR “Online Event Display” (Figure 6). Its multi-threaded version allows the STAR shift crew to investigate the detector response interactively without data taking interruption. It has been deployed during the RHIC running year 8 and proved the approach can be implemented with a reasonable amount of time investment [4] . The enhanced version of our beta-application example is being used during Run 9 (2009). It contains all essential components one can find within the STAR full-fledged production “chains” and essentially

- a top level “maker” – “StSteeringModule”
- Input/Output “maker” – “StDataReadModule”
- a “worker maker” producing data: – ‘StDetectorGeoModule”
- a visualization module – “StDisplayModule”

Figure 6 presents the thread relation between “Event Display” modules: a top maker coordinates the sequence of events and handles the call to IO and Geometry related makers

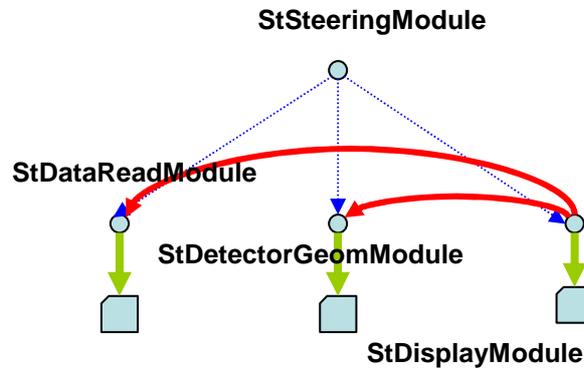


Figure 6. Hierarchy of the concurrent STAR online “Event Display” maker.

## 5. Conclusion

The analysis showed that the existing design of STAR single-threaded framework could evolve transparently to meet the Multi-Core Era requirement with little effort and no global re-design. While it is clear one should anticipate many problems arising from the onset of the multi-core era and much additional effort (at algorithmic level) may be needed to fully harvest the power of many core architectures, our approach is a first step toward a “soft” evolution of an existing framework which has proven its worth and stability through years of usage. In this paper, we concentrated our effort to identify, implement and test a crucial design component targeted to achieve “transparent” parallelization, synchronization and registration of data in a provider / consumer design model. Furthermore, and since it relies on the ROOT base class “TDataSet”, our approach could be made general for ROOT users.

Our proof of principle implementation in an “EventDisplay” context allowed testing the main design solution aimed to prepare STAR to enhance its production and analysis software frameworks with the ability to take some advantage of modern hardware. More work will certainly be needed to bring together the existing reconstruction framework and algorithms and the full potential power of the multi-core hardware for the offline “event reconstruction” and raise the level of usability of interactive data-analysis.

## 6. Acknowledgements

This work was supported by the Office of Nuclear Physics and High Energy Physics within the U.S. Department of Energy Office of Science, and the U.S. National Science Foundation.

## References

- [1] J. Adams et al, *Experimental and Theoretical Challenges in the Search for the Quark Gluon Plasma: The STAR Collaboration's Critical Assessment of the Evidence from RHIC Collisions*, in Nuclear Physics A, v. 757, p.102-183, 2005.
- [2] V. Fine et al, *Steps Towards C++/OO Offline Software in STAR*, in proceedings of *CHEP'98 conference*, Chicago, 1998
- [3] V. Fine et al, *OO Model of STAR detector for simulation, visualization and reconstruction*, in proceedings of *CHEP'2000 conference*, Padova, 2000
- [4] V. Fine, et al, *The Object Model to Construct the Mixed Open Inventor/ ROOT 3D scenes*, in proceedings of *ACAT 2007 Workshop, Amsterdam, 2007* [PoS\(ACAT\)023](#)