The STAR “Plug and Play” Event Generator Framework

J. Webb, J.Novak, J. Lauret, V. Perevoztchikov

Abstract

The STAR experiment pursues a broad range of physics topics in pp, pA and AA collisions produced by the Relativistic Heavy Ion Collider (RHIC). Such a diverse experimental program demands a simulation framework capable of supporting an equally diverse set of event generators, and a flexible event record capable of storing the (common) particle-wise and (varied) event-wise information provided by the external generators. With planning underway for the next round of upgrades to exploit ep and eA collisions from the electron-ion collider (or eRHIC), these demands on the simulation infrastructure will only increase and requires a versatile framework.

STAR has developed a new event-generator framework based on the best practices in the community (a survey of existing approach had been made and the “best of all worlds” kept in mind in our design). It provides a common set of base classes which establish the interface between event generators and the simulation and handles most of the bookkeeping associated with a simulation run. This streamlines the process of integrating and configuring an event generator within our software chain. Developers implement two classes: the interface for their event generator, and their event record. They only need to loop over all particles in their event and push them out into the event record. The framework is responsible for vertex assignment, stacking the particles out for simulation, and event persistency. Events from multiple generators can be merged together seamlessly, with an event record which is capable of tracing each particle back to its parent generator. We present our work and approach in detail and illustrate its usefulness by providing examples of event generators implemented within the STAR framework covering for very diverse physics topics. We will also discuss support for event filtering, allowing users to prune the event record of particles which are outside of our acceptance, and/or abort events prior to the more computationally expensive digitization and reconstruction phases. Event filtering has been supported in the previous framework and showed to save enormous amount of resources – the approach within the new framework is a generalization of filtering.

Design Goals

Design goals were developed after analyzing our existing framework, studying use cases in STAR and anticipated for eSTAR, and surveying existing frameworks from Alice, Atlas, CMS, and FairRoot, and considering the HepMC event record.

(1) Provide a streamlined framework in which developers can integrate their event generators using a minimum number of clearly documented base classes.

(2) Present end-users with a single-pass “plug-and-play” framework. Users add the generator to their chain, configure it through a uniform interface, and run their simulation in a single pass. Support adding event generators to our embedding framework for, e.g., adding simulations on top of zero- or minimum-bias triggered data.

(3) Leverage existing STAR infrastructure, maintaining support for *starsim* – STAR's legacy (FORtran) MC application – while establishing the basis for the development of the *starvmc* virtual Monte Carlo application.

(4) Implement persistent event records which stores particle information from the event generator, and provides a uniform treatment of event-wise data (e.g. parton-level kinematics) for the case of polarized *pp, ep, pA,* and *AA* collisions. Users should be able to analyze the persistent ROOT files w/out loading in additional libraries.

(5) Permit end-users to mix results from multiple event generators, replay events stored in ROOT files, modify particle lists before stacking particles out for simulation.

(6) Provide the same capabilities for event filtering currently present in starsim.

Design Overview

The design which we settled on is based around three main classes which will be run in the STAR Big Full Chain**.**

StarPrimaryGenerator
- Interface to the STAR Big Full Chain – steers simulation
- Aggregates particles from user's event records
- Handles bookkeeping, i.e. keeps track of which generated particle corresponds to which particle in the geant stack
- Pushes particles out to the geant stack for simulation

StarGenerator
- Abstract Base Class which developers use to implement their interface to the concrete event generator
- Establishes a common set of methods for base configuration of event generators
  - SetFrame(…) // The interaction frame
  - SetBlue(…) // ID of the particle in RHIC's “blue” beamline
  - SetYellow(…) // ID of the particle in RHIC's “yellow” beamline

StarGenEvent
- Base class for event record
- Provides list of generated particles, list of stable particles stacked for simulation
- Provides event-wise information, e.g. process ID, event kinematics, etc...

StarPrimaryGenerator

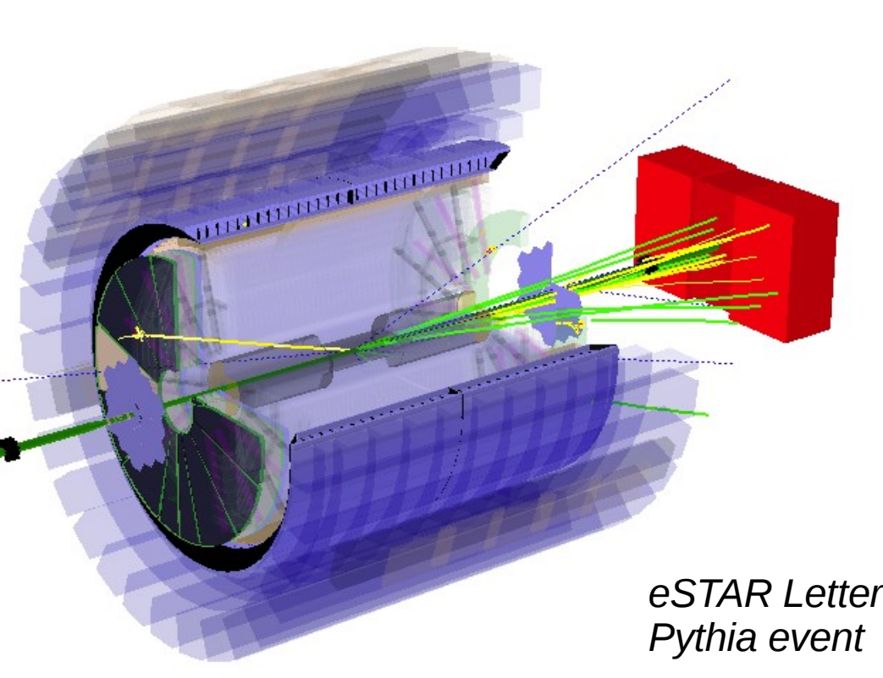
```
StarPrimaryGenerator
/// Add generator, optional vertex
Add(StarGenerator*)
/// Set global vertex and sigma
Vertex( v[3], s[3] )
/// Initialize
Init()
/// Generate event
Make()
/// Prepares for the next event
Clear()
/// Adds track to the particle stack
private PushTrack( id, px, py, pz, … )
/// Sets the particle stack
Stack( StarParticleStack *s )
/// Accept/reject particle based on
/// user-provided cuts
bool Accept( StarGenParticle *p )

/// Set the filename for TTree IO
SetFilename( name )

vector<StarGenerator *> mList
vector<array> mVert, mSigma
StarGenEvent      *mEvent
StarParticleStack *mStack
double mVertex[3], mSigma[3]

string mFilename
TTree mTree
```

StarPrimaryGenerator inherits from StMaker, the class which executes and organizes“tasks” within the STAR framework. It is responsible for steering the entire event-generation process, accumulating particles from all event-generators, pushing them out to the Monte Carlo particle stack, creating the persistent record of the event. Support for rejecting particles based on user-defined filter conditions is also provided.

The particle stack is derived from ROOT's TVirtualMCStack interface, which will enable us to integrate our event generators into the STAR VMC application as the project matures. The interface with our extant FORtran-based simulation package is provied by the AgStarReader class.

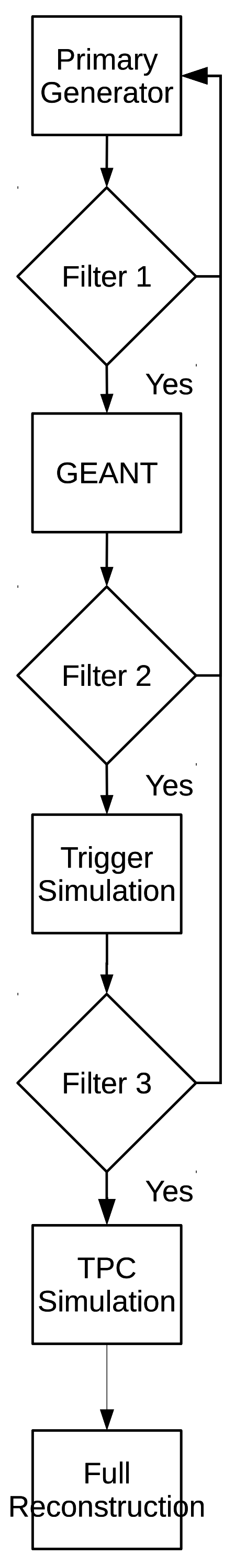
```
StarParticleStack : TVirtualMCStack
  /// Add a track to the event
  PushTrack( todo, id, px, py, pz,
        vx, vy, vz,...)
  /// Methods specified by the VMC
  /// interface
```

```
AgStarReader
/// Returns the singleton instance of
/// this class
&Instance()
/// Add a track to the event
SetStack( StarParticleStack *s)
/// Read event from the stack and push
/// tracks out to starsim
ReadEvent()
extern “C” {
  void agusread_{ AgStarReader::ReadEvent(); }
}
```



eSTAR Letter of Intent
Pythia event

Simulation Workflow



Run one or more generators and merge event records

Does the event pass a user-defined filter?

Stack the particles for simulation.

Do the geant hits pass a user-defined filter?

Digitize hits in trigger detectors.

Do the simulated hits in trigger detectors pass a user-defined filter?

Proceed with the time-consuming portions of the chain.

The STAR Primary Event generator is responsible for running the concrete event generators and assembling the event record. The event is passed to GEANT for full simulation, if it satisfies appropriate filter conditions.

After GEANT simulation, we can apply additional filters which can abort the event before the more expensive trigger- and TPC-simulations and event reconstruction.

StarGenerator

```
StarGenerator
/// Set vertex and sigma
Vertex( v[3], s[3] )
/// Initialize
Init() = 0
/// Prepares for the next event
public Clear() = 0
/// Generate event
Generate() = 0
/// User code executed before event
virtual PreEvent(){ }
/// User code executer post event
virtual PostEvent(){ }
/// Hook to handle PreEvent
private PreGenerate()
/// Hook to handle PostEvent
private PostGenerate()
/// Hook to handle Generate
private GenerateEvent()

/// Sets the frame of the interaction
SetFrame( frm=“CMS” , ene=510 )
/// Sets the particles in RHIC's “blue”
/// and “yellow” beams
SetBlue( part=“p” )
SetYell( part=“p” )
/// Sets the impact parameter
SetImpact( min, max )

/// Fills various event record types
FillPP(), FillEP(), FillEA(), FillAA()
StarGenEvent *mEvent;
```

StarGenerator defines the interface between event generators and the STAR framework. In order to integrate a new concrete event generator, developers only need to implement one class which inherits from this abstract base class. In general, this means the developer must

1) Implement set methods to configure the event generator, allowing the generator to be steered from w/in an interactive ROOT session or macro.
2) Implement the Init() method to configure the concrete event generator
3) Implement the Generate() method to exercise the event generation machinery within the concrete event generator
4) Provide code to fill the various supported event types (pp, ep, AA and eA)
5) Push particles out into the event record
   – translate concrete event generator particle ID codes and status codes into the PDG and HepMC standards
6) Create the interface between the event generator and the concrete event generator. In the case of FORtran event generators, this will involve exposing the common blocks and subroutines to C++.

Event-Wise Data

```
StarGenEvent

public:
  TClonesArray mParticles //StarGenParticle
  TRefArray    mSimulated

  int mEventNumber
  int mRunNumber
  int mDaqRunNumber
  int mGeneratorId
  int mBlueId, mYellowId
  double mCmsEnergy
  int mNRejected[3] // total, eg, filt

  int    mProcessId
  int    mFilterResult
  vec<float> mWeights // user defined

  int mNumParticles

public:
  StarGenEvent();
```

StarGenEvent – the Event Record

- Provides persistence for generated particles
- Maintains a list of particles which are stacked out for simulation
- Keeps track of common event-wise information, such as event number, run number, etc...
- Optimized for ROOT I/O... particle class avoids the
- “backwards pointer” problems of HepMC.
- Four use cases in STAR/eSTAR – pp, ep, AA and eA have four different implementations of the event record.
- Developers can further customize for specific generators

```
StarPPEvent : StarGenEvent

  int    mPartonId[2]
  double mX[2]
  double mxPDF[2]
  double mPt
  double mCosTheta
  double mPhi
  double mRapidity
  double mQ2
```

```
StarHIEvent : StarPPEvent

int    mNucleonId[2]
int    mNPartProt[2]
int    mNPartNeut[2]
int    mNWounded[2]
int    mNBinary
double mImpactParam
double mImpactPhi
int    mNumJets
...
```

```
StarDISEvent : StarGenEvent
int    mPartonId
int    mLeptonId
double mX
double mxPDF
double mPt
double mCosTheta
double mPhi
double mRapidity
double mQ2
double mW2
double mNu
```

```
StarDISAEvent : StarDISEvent

int    mNucleonId
int    mNPartProt
int    mNPartNeut
int    mNWounded
int    mNBinary
double mImpactParam
//?double mImpactPhi
int    mNumJets
… TBD
```

Particle-Wise Data

StarGenParticle provides a lightweight particle class which is essentially a facade to the well-established HEPEVT standard, with support added for merging events from multiple generators.

Below is a simplified printout of an event record from a pythia 8 single-diffractive simulation at 510 GeV CMS, with a toy event generator adding muons to it. mKey is a unique ID assigned to every particle. mIndex tracks the order in which each event generator added the particle. And mStack keeps track of the order in which each particle was added to the GEANT stack. Highlighted in bold are two special particles which are inserted to provide information about each generator.

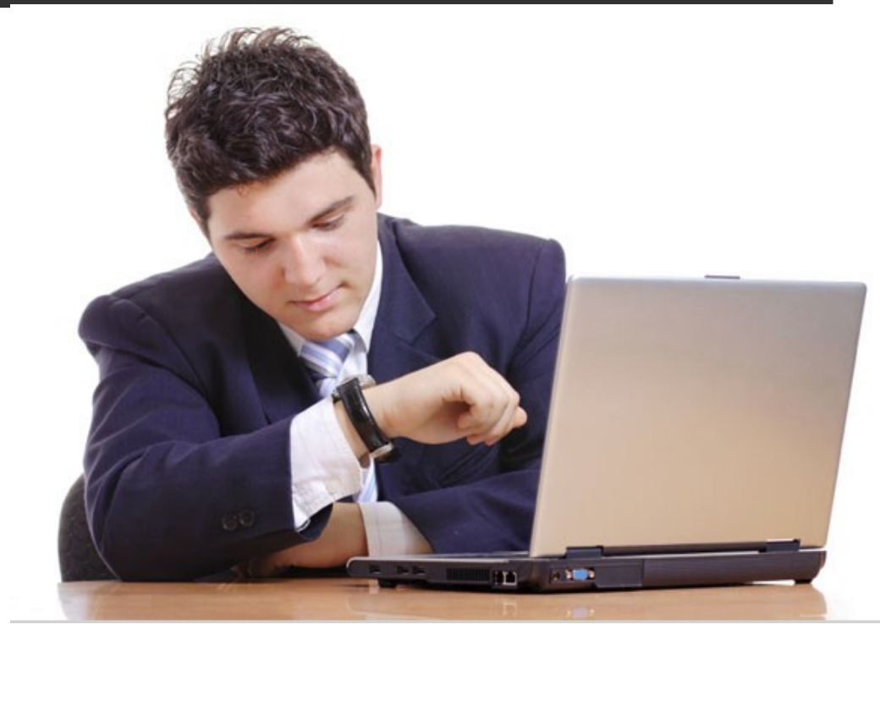
```
StarGenParticle
private:
  // /HEPEVT/ compatible data block
  int mStatus;
  int mId;
  int mMother[2];
  int mDaughter[2];
  float mPx, mPy, mPz;
  float mEnergy, mMass;
  float mVx, mVy, mVz;
  float mTof;
  void *get_hepevt(){ return &mStatus; }

  // Bookeeping for multiple generators
  int mIndex;
  int mStack;
  int mKey;    // primary key

  int mGeneratorId;

public:
  // Setters and Getters
  SetStatus(s); GetStatus(); etc...
```

```
mKey mIndex mStack   Particle Id      Staus    4-momentum                              mass    MOMS  KIDS
---------------------------------------------------------------------------------------------------------
[   0|    0|   -1]        0    Rootino  -201 (   0.000,    0.000,    0.000,    0.000;  510.000) [0 0] [0 0]
[   1|    1|    1]       13        mu-    01 (   1.464,    2.619,   10.447,   10.870;    0.106) [0 0] [0 0]
[   2|    0|   -1]        0    Rootino  -201 (   0.000,    0.000,    0.000,    0.000;  510.000) [2212 2212] [1 8]
[   3|    1|   -1]     2212     proton    04 (   0.000,    0.000, 254.998, 255.000;    0.938) [2 2] [5 2]
[   4|    2|   -1]     2212     proton    04 (   0.000,    0.000,-254.998, 255.000;    0.938) [2 2] [6 2]
[   5|    3|   -1]  9902210               15 (  -0.415,    0.118, 254.997, 255.001;    1.392) [3 2] [7 8]
[   6|    4|   -1]     2212     proton    01 (   0.415,   -0.118,-254.997, 254.999;    0.938) [4 2] [2 2]
[   7|    5|   -1]        2          u    23 (  -0.013,    0.004,    7.999,    8.003;    0.232) [5 2] [9 10]
[   8|    6|   -1]     2101       ud_0    63 (  -0.402,    0.114, 246.997, 246.998;    0.464) [5 2] [9 10]
[   9|    7|    2]      211        pi+    01 (  -0.115,   -0.193,   18.287,   18.289;    0.140) [7 8] [2 2]
[  10|    8|    3]     2112    neutron    01 (  -0.300,    0.310, 236.710, 236.712;    0.940) [7 8] [2 2]
```

Status code -201 denotes a bookkeeping entry representing a new event generator.

Filtering

High energy physics cross sections feature dramatic variations in scale, for example falling many orders of magnitude within the transverse momentum range measured at STAR. Comprehensive sampling of these cross sections becomes prohibitively expensive: the majority of computational resources are expended drawing events from low pT. The majority of these events will be discarded by offline analysis as they try to compare the simulated data samples with real data,with high-pT thresholds imposed by the online trigger.



```
StarFilterMaker
  // User-defined filtering function
  virtual int Filter( StarGenEvent *event=0 );

  // Returns total number of events seen
  int numberOfEvents();
  // Returns total number of events accepted
  int acceptedEvents();
  // Returns total number of events rejected
  int rejectedEvents();
  // Returns total number of events rejected
  // since last accept
  int rejectedSinceLast();

  // Aborts the event and signals primary maker
  // to save event record
  void Abort()
```

Filtering enables us to examine the generated event at an early stage in the simulation, and stop processing it if it is unlikely to be of use in offline analysis. This saves us from passing unnecessary events to the more computationally demanding parts of the simulation chain.

Filters are derived from the StarFilterMaker abstract base class, which is responsible for aborting the event and keeping track of useful statistics about the run. Filters can be inserted at various points in the simulation chain, and can operate on the event record, geant hits and/or simulated trigger response.

Summary and Conclusion

The STAR experiment at Brookhaven National Lab continues to pursue a rich and diverse experimental program utilizing the Relativistic Heavy Ion Collider. To better support these programs, and with the plans for eSTAR at eRHIC in mind, we have developed a new framework for event generation which

- Leverages our existing simulation package and
- Lays the groundwork for the development of a new virtual MC application
- Establishes a simple, well-defined interface between event generators and the STAR software stack
- Streamlines for the process of integrating new event generators
- Provides end users with a single pass, plug-and-play workflow
- Supports complex use cases, including filtering and modifying the event records from standard event generators before simulation
- provides a complete, portable event record which covers most event generators, while allowing developers the flexibility to extend the event record to meet specific needs.

BROOKHAVEN NATIONAL LABORATORY

STAR