Środowisko programistyczne GEANT4

Leszek Adamczyk

Wydział Fizyki i Informatyki Stosowanej
Akademia Górniczo-Hutnicza

Wykłady w semestrze zimowym 2015/2016

Wyniki symulacji

Mając zdefiniowaną:

- geometrię;
- listę procesów fizycznych;
- kinematykę cząstek pierwotnych

GEANT wykonuje symulacje.
Użytkownik musi napisać kawałek kodu aby uzyskać dostęp do informacji dla niego użytecznej. Można zrobić to na cztery sposoby:

- Korzystając z klas G4UserTrackingAction, G4UserSteppingAction i G4UserStackingAction. Mamy dostęp do pełnej informacji ale sami musimy zadbać o jej przetwarzanie w trakcie procesowania przypadku.

Wyniki symulacji

- Implementując klasę G4VSensitiveDetector.
  GEANT automatycznie przetwarza informację, zapisując ją w postaci obiektów klasy G4VHit.
  Użytkownik ma dostęp do pełnego zbioru hitów (G4THitsCollection) po zakończeniu procesowania przypadku korzystając z klasy G4UserEventAction
- Korzystając z gotowej implementacji klasy G4VSensitiveDetector tzw. G4MultiFunctionalDetector opartej o obiekty G4VPrimitiveScorer.
  ”Proste liczniki” tworzą mapy, G4THitsMap, dostępne dla użytkownika po zakończeniu procesowania przypadku.
  W GEANT mapa jest tablicą indeksowaną liczbą naturalną, będącą najczęściej numerem kopii obszaru logicznego.
  HitsMap najczęściej stosuje się w aplikacjach nie wymagających informacji dla każdego przypadku a jedynie wysumowanej po wszystkich przypadkach.
- Korzystanie z wbudowanych w UI ”prostych liczników” tworzących HitsMap na podstawie geometrii niezależnej (równoległej) do geometrii detektora.

Wyniki symulacji

- Obiekty klasy G4VSensitiveDetector (G4MultiFunctionalDetector), przypisujemy obszarom logicznym.
- Tworząc geometrię detektora, czasami należy drobniej ”podzielić” obszar detektora niż by to wynikało z jego budowy fizycznej. Np. aby uzyskać informacje z mniejszych obszarów.
- Powinniśmy korzystać z wielokrotnego umieszczania tego samego obszaru logicznego (Parameterised, Replica), wtedy używamy jednego obiektu klasy SensitiveDetector a jego HitsMap w naturalny sposób indeksowana jest numerem kopii tego obszaru.

MultiFunctionalDetector

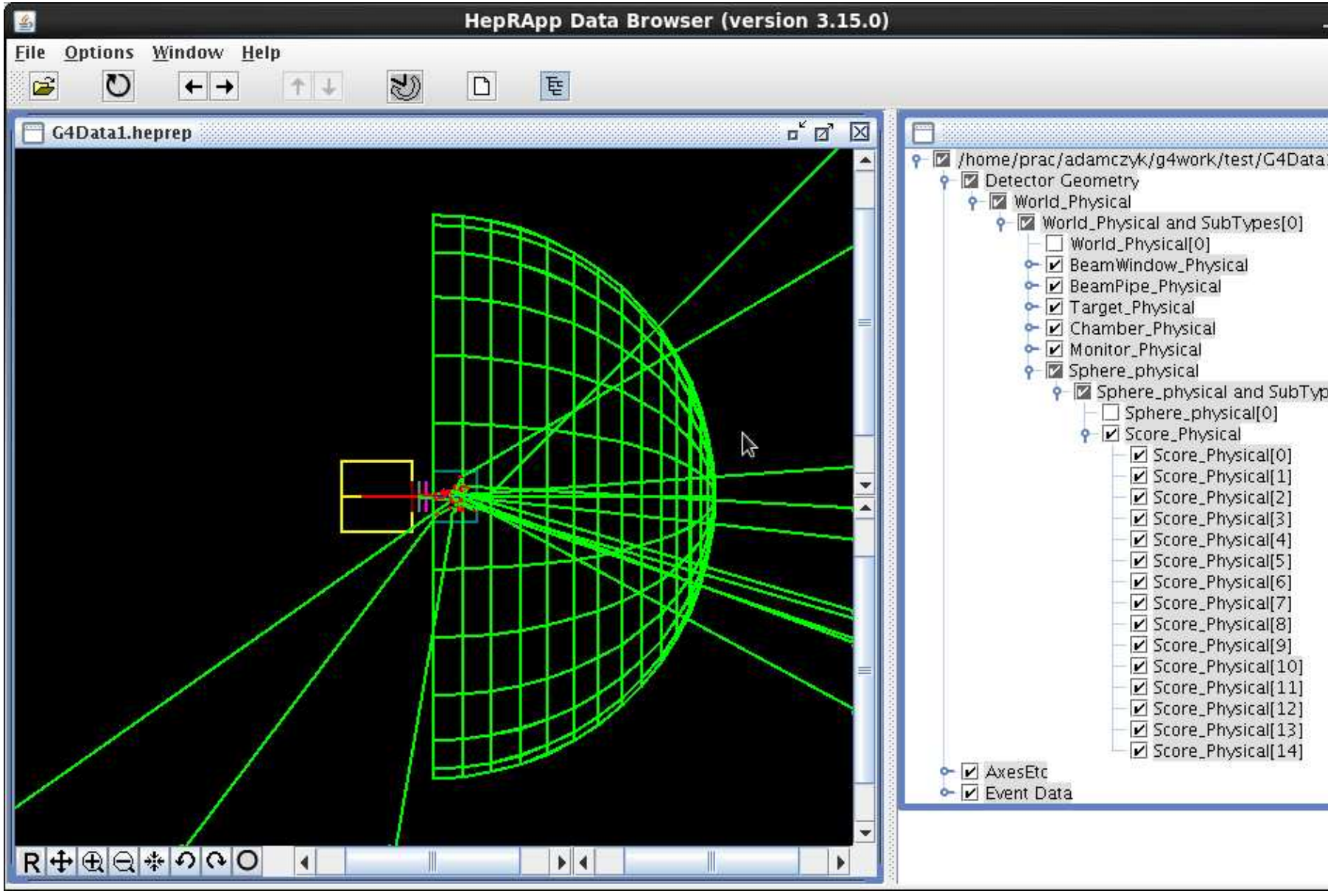
- G4MultiFunctionalDetector jest implementacją klasy bazowej G4VSensitiveDetector.
- Zamiast samemu implementować SensitiveDetector budujemy go rejestrując obiekty klasy PrimitiveScorer do MultiFuntionalDetector.
- G4VPrimitiveScorer jest klasą bazową z której wyprowadzamy konkretne klasy pochodne. Istnieje szereg gotowych klas np: G4PSEnergyDeposit lub G4PSTrackLength.
- Pełna lista w rozdziale 4.4.6 (Application Developers Guide)

MultiFunctionalDetector

Aby korzystać z funkcjonalności MultiFunctionalDetector należy:

- Utworzyć obiekt klasy G4MultiFunctionalDetector;
- Zarejestrować go do SensitiveDetectorManager'a (obiekt klasy G4SDManager);
- Przypisać MultiFunctionalDetector do jednego lub kilku obszarów logicznych;
- Utworzyć obiekt lub kilka obiektów klasy PrimitiveScorer;
- Zarejestrować PrimitiveScorer's do MultiFunctionalDetector

Aplikacja My



Aplikacja My

```
// Create a new sensitive detector named "MyDetector"
  G4MultiFunctionalDetector* detector =
    new G4MultiFunctionalDetector("MyDetector");
```

```
// Get pointer to detector manager
  G4SDManager* manager = G4SDManager::GetSDMpointer();
```

```
// Register detector with manager
  manager->AddNewDetector(detector);
```

```
// Attach detector to scoring volume
  scoreLogical->SetSensitiveDetector(detector);
```

```
// Create a primitive Scorer named MyScorer
  G4PSSphereSurfaceCurrent* scorer =
    new G4PSSphereSurfaceCurrent("MyScorer",  fCurrent_In);
```

```
// Register scorer with detector
  detector->RegisterPrimitive(scorer);
```

G4THitsMap

- Każdy G4VPrimitiveScorer generuje jedną mapę G4THitsMap<G4double> na przypadek.
- Mapa ta jest tablicą o wartościach G4double indeksowaną numerem kopii obszaru logicznego do którego przypisany jest MultiFunctionalDetector.
- Mapa może być indeksowana numerem kopii matki (babci, itd).
- Trzeci parametr konstruktora obiektu PrimitiveScorer określa którego przodka numer kopii będzie indeksem mapy.

```
G4PSSphereSurfaceCurrent(..., ..., G4int depth=0);
```

- W bardziej skomplikowanych schematach indeksowania można zaimplementować własną metodę GetIndex()

G4VSDFilter

- Obiekt klasy SDFilter może być przypisany do PrimitiveScorer w celu określenia na jaki rodzaj cząstek dany licznik jest czuły.
  Istnieją cztery gotowe implementacje klasy G4VSDFilter:
  - G4SDChargedFilter
  - G4SDNeutralFilter
  - G4SDParticleFilter
  - G4SDKineticEnergyFilter

```
G4VSDFilter* gammaFilter =
     new G4SDParticleFilter("gammaFilter");
gammaFilter->Add("gamma");
scorer->SetFilter(gammaFilter);
```

G4HCofThisEvent

- G4HCofThisEvent jest klasą która zawiera HitsCollection oraz HitsMap dla danego przypadku.
- HitsMap jest dostępna z HCofThisEvent poprzez unikalny numer identyfikacyjny danej HitsMap.
- Numer identyfikacyjny można uzyskać poprzez funkcję G4SDManager::GetCollectionID()

```
G4SDManager* manager = G4SDManager::GetSDMpointer();
G4int mapID = manager->
           GetCollectionID("MyDetector/MyScorer")

//   G4Event* evt
G4HCofThisEvent* hce = evt->GetHCofThisEvent();
G4THitsMap<double>*  eventGammaScorer =
        (G4THitsMap<double>*) (hce->GetHC(mapID));
```

G4THitsMap

G4THitsMap<G4double> ma wbudowane operatory:

- operator indeksowania map[index]
  zwraca wskaźnik do elementu mapy o indeksie index
  Wskaźnik o wartości zero oznacza że nie istnieje element mapy o indeksie index
- map1 $+=$ map2
  Dodaje wartości mapy map2 do odpowiednich wartości mapy map1.
  Usprawnia akumulacje danych po wszystkich przypadkach.

Przykład

```
G4SDManager* manager = G4SDManager::GetSDMpointer();
G4int mapID = manager->
           GetCollectionID("MyDetector/MyScorer")

//   G4Event* evt
G4HCofThisEvent* hce = evt->GetHCofThisEvent();
G4THitsMap<double>*  eventGammaScorer =
        (G4THitsMap<double>*) (hce->GetHC(mapID));
```

```
//   G4THitsMap<G4double> runGammaScorer

runGammaScorer +=  *eventGammaScorer;
```

```
for (G4int i=0; i<runGammaScorer.GetSize(); i++) {
   G4cout<< runGammaScorer[i] << G4endl;
 }
```

G4UserRunAction

- Po zakończeniu procesowania przypadku obiekt klasy G4Event posiada informację o HitsMap i HitsCollection dla danego przypadku.
- Użytkownik musi napisać kawałek kodu aby np. sumować wartości HitsMap lub zapisywać je do dalszej analizy.
- Można to zrobić na wiele sposobów;
- Na warsztatach zaimplementujemy własne klasy G4UserRunAction oraz G4Run .
- Klasa G4UserRunAction ma następujące motody wirtualne:
  - BeginOfRunAction()
  - EndOfRunAction()
  - GenerateRun() - służy do utworzenia obiektu samodzielnie zaimplementowanej klasy G4Run

G4Run

- Klasa reprezentująca zbiór przypadków.
- Obiekt tej klasy jest tworzony i niszczony przez RunManagera (bezpośrednio lub za pośrednictwem funkcji GenerateRun() klasy G4UserRunAction).
- Funkcja składowa G4Run::RecordEvent jest automatycznie wywoływana przez RunManagera po zakończeniu procesowania każdego przypadku.
- W tej funkcji mamy dostęp do HitsMap i HitsCollection po każdym przypadku.

G4VHitsCollection

Klasa G4VHitsCollection jest klasą bazową z której wyprowadzono klasy szablonowe G4THitsCollection oraz G4THitsMap

- G4THitsCollection jest szablonową klasą pojemnikową typu vector służącą do przechowywania wskaźników do obiektów jednej konkretnej klasy wyprowadzonej z klasy G4VHit (hitów)
- G4THitsMap jest szablonową klasą pojemnikową typu map służącą do przechowywania wskaźników do obiektów jednej konkretnej (ale dowolnej) klasy indeksowanej dowolną wartością.
  Mogą to być hity ale nie muszą np. PrimitiveScorer używają G4THitsMap które zawierają wskaźniki do wartości G4double indeksowane liczbą naturalną.

G4VHit

- Klasa bazowa G4VHit służy do wyprowadzania własnych klas hitów zawierających potrzebną nam informację np.:
  - pozycje i czas kroku (G4Step)
  - pęd i energie cząstek (G4Track)
  - depozyty energii podczas kroku (G4Step)
  - informacje geometryczne
- Hit przechowywany jest w dedykowanej kolekcji która jest instancją klasy szablonowej G4THitsCollection
- HitsCollection dostępna jest z obiektu klasy G4Event poprzez obiekt klasy G4HCOfThisEvent

Diagram klas

Diagram klas

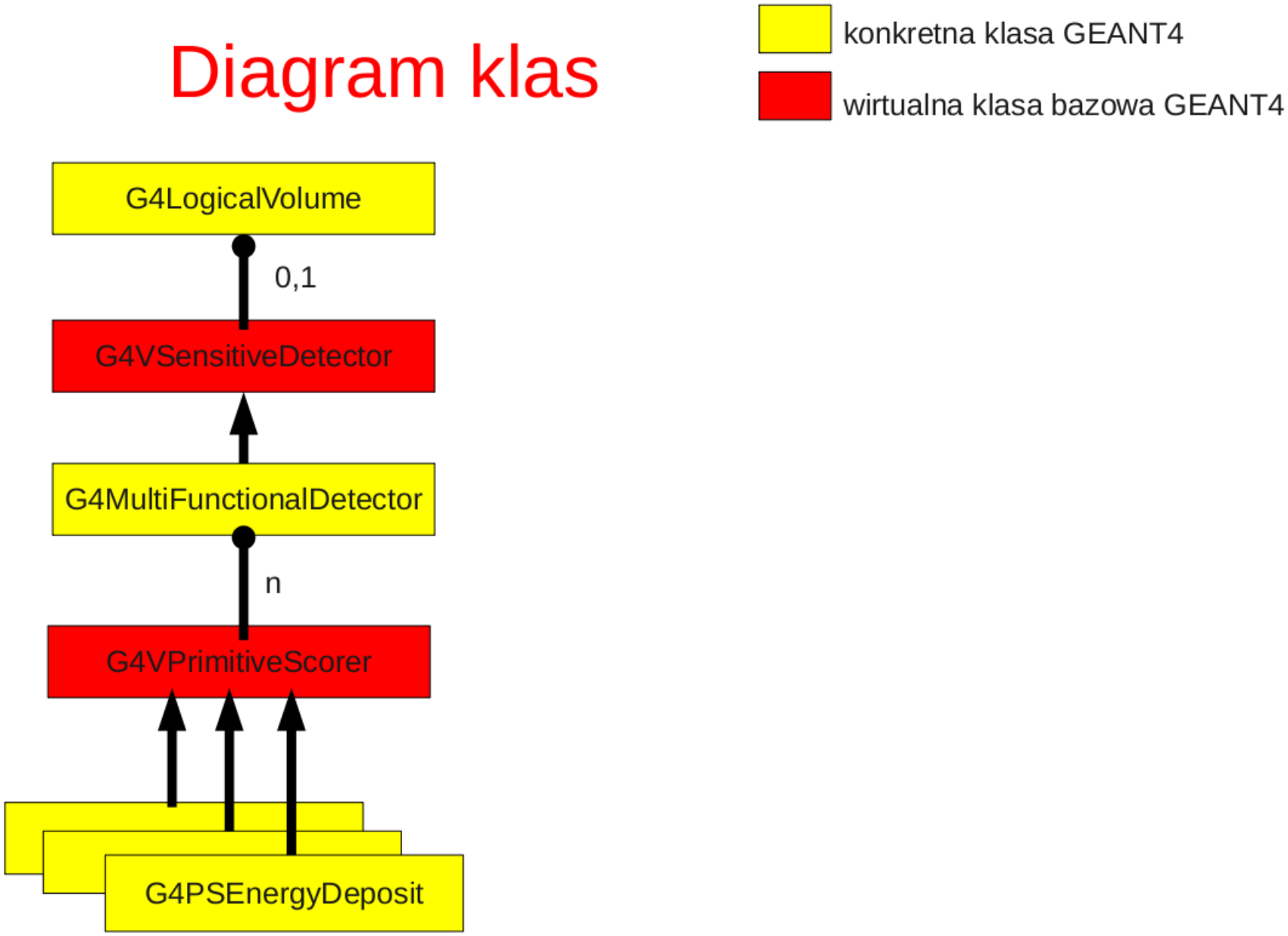


Diagram klas

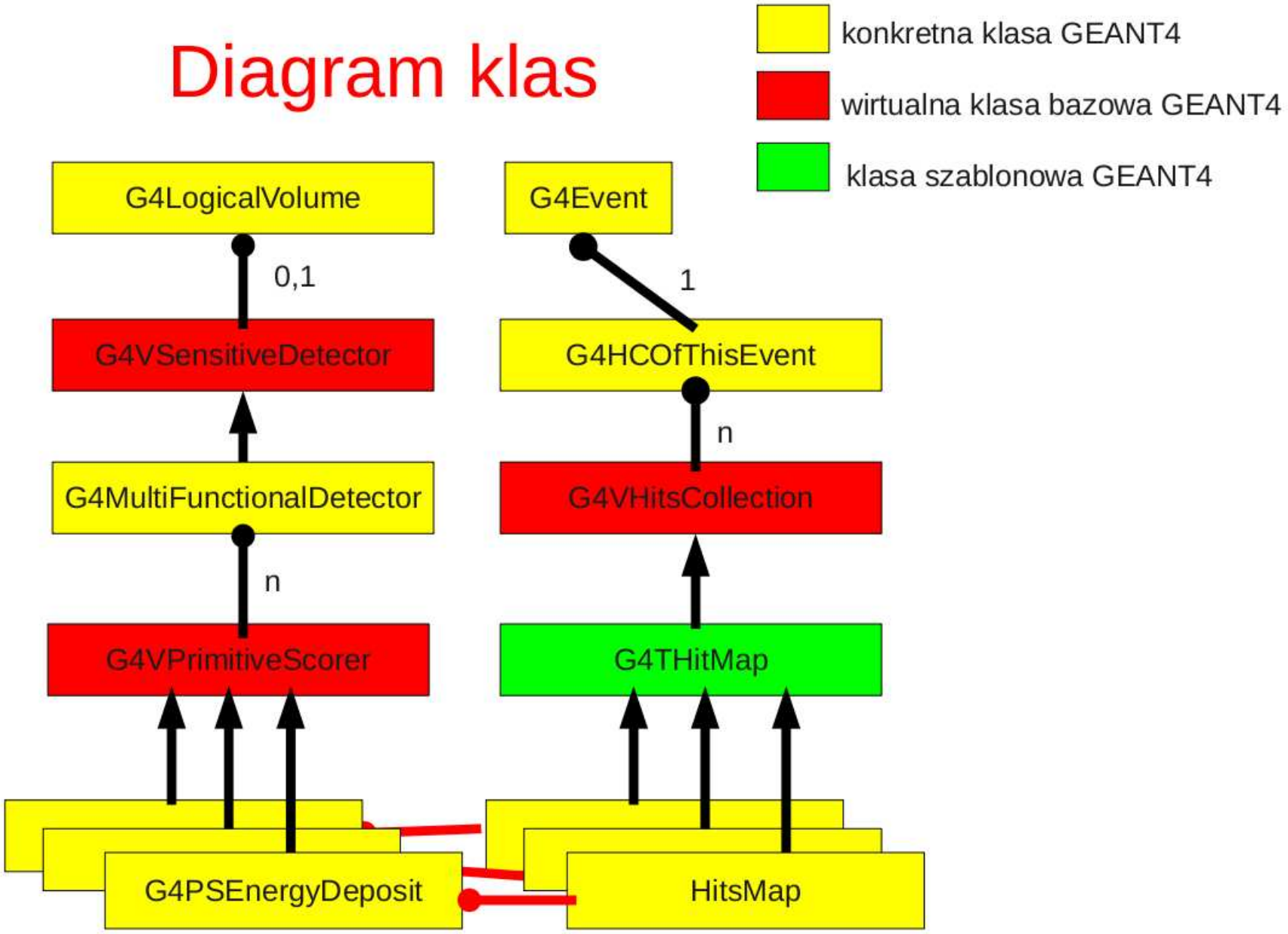
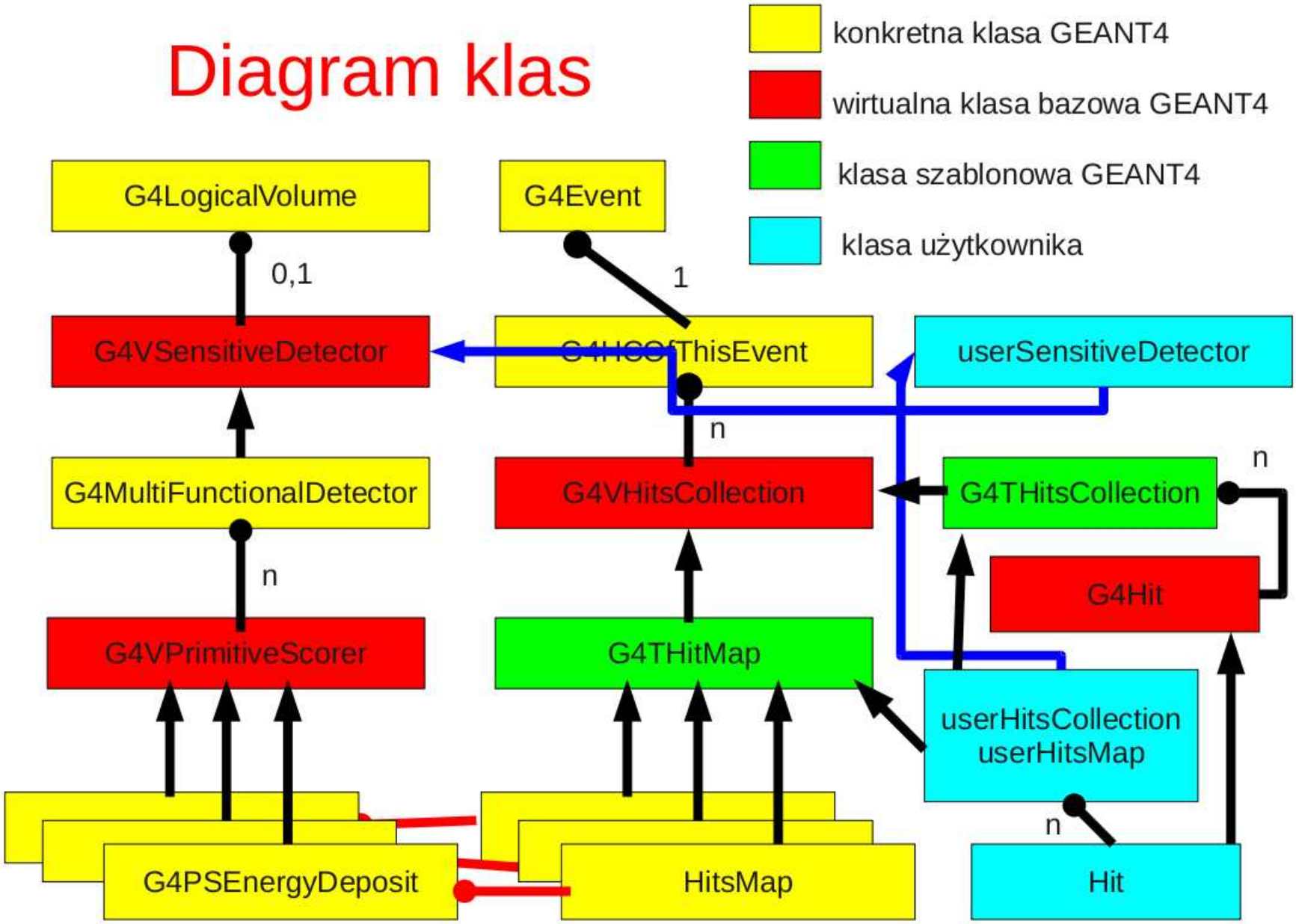


Diagram klas



G4VHit - przykład

Przykład z warsztatów implementacji klasy wyprowadzonej z klasy G4VHit

- Wyprowadzimy klasę np. MyPhotonHit zawierającą informację o energii i pozycji każdego fotonu docierającego do sfery otaczającej tarczę berylową.
- Klasa G4VHit zawiera dwie czysto wirtualne metody Draw oraz Print które można zaimplementować jeśli chce się np. wizualizować hity

MyPhotonHit.hh

```
class MyPhotonHit : public G4VHit {

public:
  // Constructor
  MyPhotonHit();

  // Destructor
  virtual ~MyPhotonHit();

  // Methods
  // void Draw()
  // void Print()

  // Photon energy
  inline void SetEnergy(G4double energy) {fEnergy = energy;}
  inline G4double GetEnergy() const {return fEnergy;}

  // Photon position
  inline void SetPosition(G4ThreeVector position) {fPosition = position;}
  inline G4ThreeVector GetPosition() const {return fPosition;}

private:
  //Data Members
 G4double fEnergy;
 G4ThreeVector fPosition;
};
```

MyPhotonHit.cc

```
#include "MyPhotonHit.hh"

MyPhotonHit::MyPhotonHit()
  :fEnergy(0)
  ,fPosition()
{}

MyPhotonHit::~MyPhotonHit() {}
```

G4VSensitiveDetector

Aby zaimplementować własną klasę SensitiveDetector użytkownik musi:

- w konstruktorze dodać nazwy HitCollection's do wektora collectionName oraz dokonać instancji klasy G4VSensitiveDetector
- utworzyć i dopisać HitCollection do obiektu G4HCOfThisEvent w jednej z dwóch metod Initialize() lub EndOfEvent()
- zaimplementować metodę ProcessHits(). W niej tworzymy hity oraz dodajemy je do HitCollection

MyPhotonSD.hh

```
class MyPhotonSD : public G4VSensitiveDetector {

public:

  // Constructor
  MyPhotonSD(const G4String& name);

  // Destructor
  virtual ~MyPhotonSD();

  // Methods
  void Initialize(G4HCofThisEvent* hitsCollectionOfThisEvent);
  G4bool ProcessHits(G4Step* aStep,G4TouchableHistory* history);
  void EndOfEvent(G4HCofThisEvent* hitsCollectionOfThisEvent);

private:

  // Data members
  G4THitsCollection<MyPhotonHit>* fPhotonCollection;
  G4int fPhotonCollectionID;

};
```

MyPhotonSD.cc

```
MyPhotonSD::MyPhotonSD(const G4String& name)
  :G4VSensitiveDetector(name)
{
  collectionName.insert("PhotonCollection");
  fPhotonCollectionID = -1;
}

MyPhotonSD::~MyPhotonSD() {}

void MyPhotonSD::Initialize(G4HCofThisEvent*
                            hitsCollectionOfThisEvent)
{
 // Create a new collection
 fPhotonCollection = new G4THitsCollection<MyPhotonHit>
                  (SensitiveDetectorName, collectionName[0]);
 if( fPhotonCollectionID < 0)
   fPhotonCollectionID = G4SDManager::GetSDMpointer()
                       ->GetCollectionID(fPhotonCollection);
 // Add collection to the events
 hitsCollectionOfThisEvent->AddHitsCollection
                   (fPhotonCollectionID,fPhotonCollection);
}
```

MyPhotonSD.cc

```
G4bool MyPhotonSD::ProcessHits(G4Step* aStep, G4TouchableHistory*)
{
 // Create a new hit
 MyPhotonHit* aHit = new MyPhotonHit();

 // Get energy and position
 G4double energy = aStep->GetTrack()->GetTotalEnergy();
 G4ThreeVector position = aStep->GetTrack()->GetPosition();


 // Set energy and position
 aHit->SetEnergy(energy);
 aHit->SetPosition(position);

 // Insert hit into the collection
 fPhotonCollection->insert(aHit);
}
```

G4Step

Niektóre metody klasy G4Step umożliwiające dostęp do informacji o symulacji:

- GetTrack() - dostęp do informacji o cząstce która generuje krok;
- GetPre(Post)StepPoint() - dostęp do informacji geometrycznych o początku i końcu kroku (pozycja, czas, materiał, ...)
- GetStepLength() - długość kroku;
- GetTotalEnergyDeposit() - całkowity depozyt energii w materiale podczas wykonywania kroku;
- GetDeltaXXXXX() - informacja o zmianie pędu, energii, pozycji,... cząstki generującej krok
- IsFirst(Last)StepInVolume() - wskazuje czy aktualny krok jest pierwszym(ostatnim) w danym obszarze detektora.

G4HCOfThisEvent

- Obiekt klasy G4HCOfThisEvent zawiera wszystkie kolekcje hitów dla danego przypadku.
- Wskaźnik do danej kolekcji może mieć wartość NULL jeśli dana kolekcja nie została utworzona.
- Kolekcje hitów są dostępne za pomocą wskaźników do klasy bazowej G4VHitsCollection, zatem muszą być rzutowane na konkretny typ danej kolekcji.
- Numer identyfikacyjny danej kolekcji jest unikalny i taki sam dla każdego przypadku. Można go uzyskać za pomocą funkcji:
  SDManager::GetCollectionID(nazwaSD/nazwaKolekcji)
  SDManager::GetCollectionID(G4VHitsCollection*)