Integrating with BioGears

A. Baird[1], J. Carter[1], L. Marin[1], M. McDaniel[1], N. Tatum[1], S. White[1]

1. Applied Research Associates Inc.

Agenda

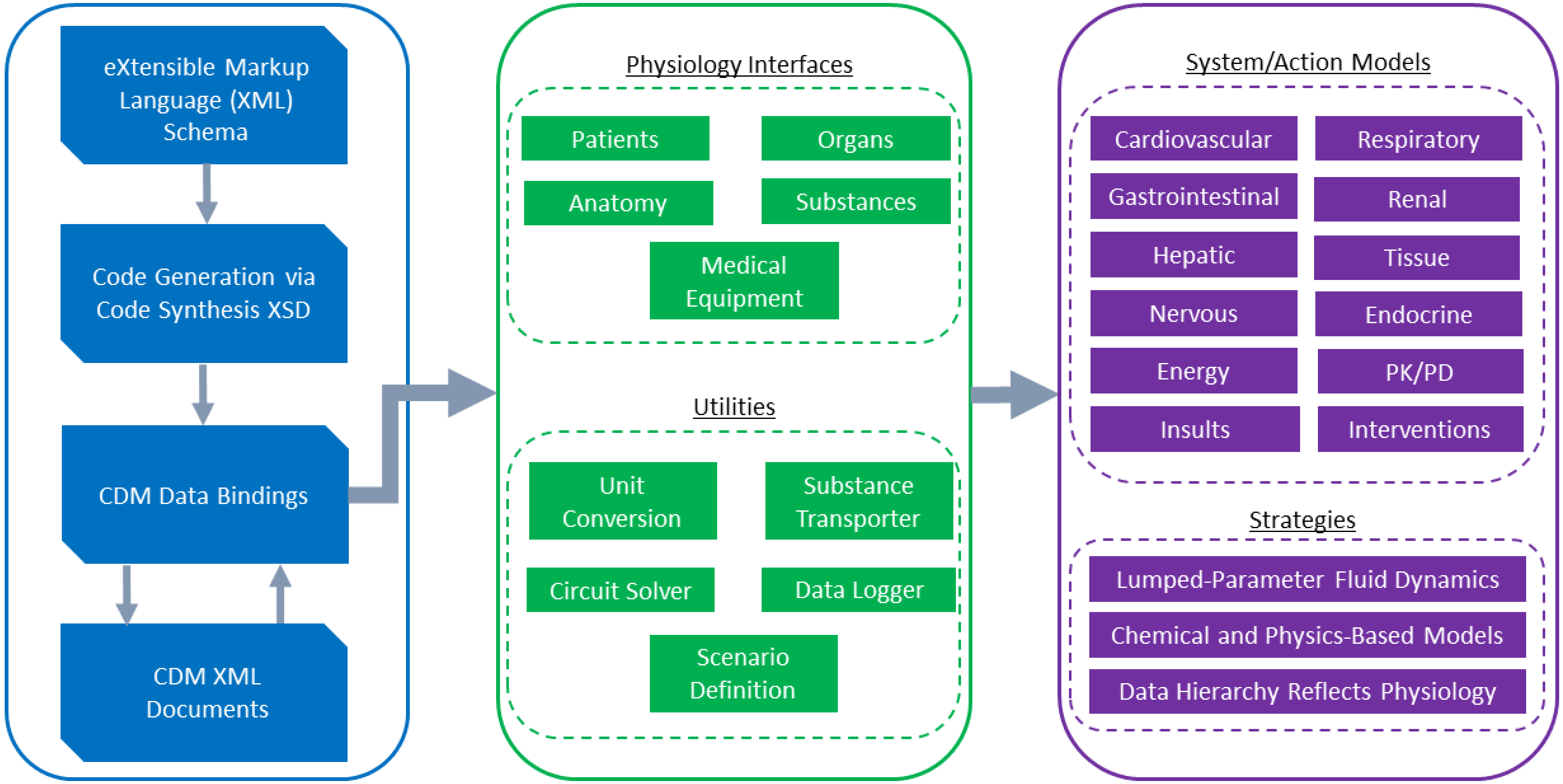
- Introduction to the BioGears Architecture
- BioGears library layout.
- Examples
  1. Synthetic Environment Review
     1. Scalar & UnitScalar
     2. Common Interfaces
  2. Basic API Integration
  3. Applying Actions
  4. Programmatic Data Request
  5. Receiving Patient Assessments
  6. Asynchronous Integration (Multi-Threading)

BioGears Architecture

Common Data Model (CDM)
Communication standard for physiology data transfer

Synthetic Environment (SE)
C++ API that implements an abstract Physiology Engine class

BioGears Engine
Physiology Engine instance developed by ARA



SEScalar and SEUnitScalar

- 74 Headers of various dimensional analysis and value validation.
- Integrators should just include *SEScalarTypes.h* which includes the most common 54 scalar units.

- All Scalars implement a common interface
  - SetValue(1.0, *UnitType*)
  - GetValue*(UnitType*) -> double

- SESystem interfaces always return a PTR to a SEScalar that is good for the life time of the parent object. With this you can call GetValue after each advanceModelTime and avoid fetching the PTR unless a state load occurs.

Synthetic Environment Common Interfaces

- SESystem
- SEAction
  - Patient Action ← Insults/Injuries to BioGears Patient
  - Environment Action ← Changes to Environment including active heating.
  - Anesthesia Machine Action ← All Anesthesia machine operation.

Simple BioGears Integration

```
g++ -std=c++17 -c -o bigears-agent bg-agent.cpp \
   -I${BIOGEEARS_INCLUDE_PATH} \
   -L${BIOGEARS_LIB_PATH} -l biogears -l biogears-cdm \
   -lxercies-c -l log4cpp
```

```
int main(int argc, char* argv[] )
{
  using namespace biogears;
  // Create the engine and load the patient
  auto logger = std::make_unique<Logger>{ "biogears-agent.log" };
  auto bg = CreateBioGearsEngine(logger.get());
  bg->GetLogger()->Info("BioGears Agent");
  if (!bg->LoadState("./states/StandardMale@0s.xml"))
  {
    bg->GetLogger()->Error("Could not load state, check the error");
    return;
  }

 bg->AdvanceModelTime(1, TimeUnit::hr);
}
```

Most of BioGears functionality bundled in two libraries

1. libBioGears (SE + Engine classes)
2. libBIoGears-cdm (cdm)

BioGears has two linker dependencies

1. Xercies-C
2. Log4Cpp

Custom Creation Function

```
auto create_engine(Logger * logger) -> std::unique_ptr<biogears::BioGearsEngine>
{
  using namespace biogears;
  // Create the engine and load the patient
  auto bg = std::make_unique<BioGearsEngine>(logger);
  bg->GetLogger()->Info("BioGearAgent");
  if (!bg->LoadState("./states/StandardMale@0s.xml"))
  {
    bg->GetLogger()->Error("Could not load state, check the error");
    return;
  }

 return bg;
}
```

Physiolgy Engine vs BioGears Engine

- Physiology Engine
  - Abstract Interface
  - All const accessors
  - Interface assumes no changes a subsystem are possible after first time advance,
  - Integrators fight lack of const correctness in subsystem interfaces(We are working on this)
- BioGears Engine
  - Concreate Engine Class
    - Derives from Physiology Engine and BioGears
    - Marked for merging with BioGears
  - Can be cast to BioGears when you need a non const accessor.
    - BioGears merge will allow BioGears Engine to access non const accessors.
- With out multiple Physiology Engine derived Concrete Implementations it doesn't make sense to use Physiolgy Engine

Performing an Action

```
auto apply_pain_stimulus(std::unique_ptr<biogears::BioGearsEngine>& bg) -> bool
{
auto hemorrhage = biogears::SEHemorrhage();
     hemorrhage.SetCompartment(compartment);
     hemorrhage.GetInitialRate().SetValue(10.,   biogears::VolumePerTimeUnit::mL_Per_min);
  if (hemorrhage.IsValid()) {
    engine->ProcessAction(hemorrhage);
    return true;
  } else {
    return false;
  }
}
```

See HowTo-Integration

Most BioGears class lack copy constructors

We plan on unifying the ownership model across the Synthetic Environment as we move towards 8.0, but when using 7.4.X you must consider that most copy and move constructors in BioGears are implicitly deleted.

For this reason BioGears often assumes ownership of a pointer if it is passed or creates a clone through serialization if passed a reference. This is not universally true. As an Example BioGears does not assume ownership of the Logger used in its construction and all SESystems assume thee objects passed in there construction are owned by the instantiator.

This can cause problems for DataTracks in the logger which reference the SubstanceManager.

Cloning through Serialization

```
auto aStress = m_AcuteStress->UnLoad();
m_AcuteStress->Load(*aStress);
delete aStress;
```

Prefer Composition over Inheritance.

- Do
  - Build BioGears integration through composition
  - Inherit from an Synthetic Environment layer to extend the concrete implementations or implement your own
- Don’t
  - Inherit from the BioGears layer

Dynamic Data Request

- Do
  - Prefer Data Request caches to DataTracks
  - Use the Physiology Tree to inspect what data is available.
  - Reset Data Request Graph if the Physiology Engine exits scope.
- Don’t
  - Call GetScalar multiple times for the same request

Data Request Cache

```
biogears::SEScalar* _arterialBloodPH;
biogears::SEScalar* _arterialBloodPHBaseline;
biogears::SEUnitScalar* _bloodDensity;
biogears::SEUnitScalar* _bloodSpecificHeat;
biogears::SEUnitScalar* _bloodUreaNitrogenConcentration;
biogears::SEScalar* _carbonDioxideSaturation;
biogears::SEScalar* _carbonMonoxideSaturation;
```

```
 auto& bloodChemistry = _engine->GetBloodChemistry();
    _arterialBloodPH = (bloodChemistry.HasArterialBloodPH()) ? &bloodChemistry.GetArterialBloodPH() : nullptr;
    _arterialBloodPHBaseline = (bloodChemistry.HasArterialBloodPHBaseline()) ?
                               &bloodChemistry.GetArterialBloodPHBaseline() : nullptr;
    _bloodDensity = (bloodChemistry.HasBloodDensity()) ? &bloodChemistry.GetBloodDensity() : nullptr;
    _bloodSpecificHeat = (bloodChemistry.HasBloodSpecificHeat()) ? &bloodChemistry.GetBloodSpecificHeat() : nullptr;
    _bloodUreaNitrogenConcentration = (bloodChemistry.HasBloodUreaNitrogenConcentration()) ?
                                      &bloodChemistry.GetBloodUreaNitrogenConcentration() : nullptr;
    _carbonDioxideSaturation = (bloodChemistry.HasCarbonDioxideSaturation()) ?
                                &bloodChemistry.GetCarbonDioxideSaturation() : nullptr;
    _carbonMonoxideSaturation = (bloodChemistry.HasCarbonMonoxideSaturation()) ?
                                 &bloodChemistry.GetCarbonMonoxideSaturation() : nullptr;
```

Data Request Use

```
 _engine->AdvanceModelTime(5, TimeUnit::s)

struct PatientMetrics : QObject {
  PatientMetrics(QObject* parent = nullptr)
    : QObject(parent)
  {
  }
  //Blood Chemistry
  double arterialBloodPH;
  double arterialBloodPHBaseline;
  double bloodDensity;
  double bloodSpecificHeat;
  double bloodUreaNitrogenConcentration;
  double carbonDioxideSaturation;
  double carbonMonoxideSaturation;
};


 PatientMetrics* current = new PatientMetrics();

 auto& bloodChemistry = _engine->GetBloodChemistry();
  current->arterialBloodPH = (_arterialBloodPH) ? _arterialBloodPH->GetValue() : 0.0;
  current->arterialBloodPHBaseline = (_arterialBloodPHBaseline) ? _arterialBloodPHBaseline->GetValue() : 0.0;
  current->bloodDensity = (_bloodDensity) ? _bloodDensity->GetValue() : 0.0;
  current->bloodSpecificHeat = (_bloodSpecificHeat) ? _bloodSpecificHeat->GetValue() : 0.0;
  current->bloodUreaNitrogenConcentration = (_bloodUreaNitrogenConcentration) ? _bloodUreaNitrogenConcentration->GetValue() :
0.0;
  current->carbonDioxideSaturation = (_carbonDioxideSaturation) ? _carbonDioxideSaturation->GetValue() : 0.0;
  current->carbonMonoxideSaturation = (_carbonMonoxideSaturation) ? _carbonMonoxideSaturation->GetValue() : 0.0;

return current;
```

Patient Assessments

```
Patient Assessment//-------------------------------------------------------------------------------
//!
//!  Pimpl Implementation Structure.
//!  demonstrates how to get the urine color
//---------------------------------------------------------------------------------

bool action_get_urine_color(std::unique_ptr<biogears::BioGearsEngine>& engine)
{
    auto urineAnalysis = biogears::SEUrinalysis(engine->GetLogger());

    engine->GetPatientAssessment(urineAnalysis);
    if (urineAnalysis.HasColorResult())
    {
      mil::tatrc::physiology::datamodel::enumUrineColor eColor = urineAnalysis.GetColorResult();

      switch (eColor) {
      case mil::tatrc::physiology::datamodel::enumUrineColor::DarkYellow:
         std::cout << "Urine Color: Dark Yellow";         return true;
      case mil::tatrc::physiology::datamodel::enumUrineColor::PaleYellow:
         std::cout << "Urine Color: Pale Yellow";         return true;
      case mil::tatrc::physiology::datamodel::enumUrineColor::Pink:
         std::cout << "Urine Color: Pink";         return true;
      case mil::tatrc::physiology::datamodel::enumUrineColor::Yellow:
         std::cout << "Urine Color: Yellow";         return true;
      default:
         std::cout << "Urine Color: Default";
         return true;
    }
}
return false;
```

Asynchronous Development

Do:

- Copy Data Between Threads
- Guard ProcessAction and GetPatientAssessments with Critical Sections.
- Use a Locking Mechanism for guarding AdvanceModelTime calls with DataRequest and Actions.
- Keep Memory Allocation to a Single Thread

Don’t:

- Introduce New Substances while model time Advances
- Modify any internal data while model time advances

Driving BioGears

```
//------------------------------------------------------------------------------
void UBioGearsEngineDriver::engine_thread_main(){
    auto current_time = std::chrono::steady_clock::now();
    std::chrono::time_point<std::chrono::steady_clock> prev;
    _engine_running = true;
    while (_engine_thread_continue) {
      prev = current_time;
      if (_engine && !_engine_paused) {
         while (_action_queue.size()) {`
                UE_LOG(BioGearsLog, Error, TEXT("Unable to process given action."));
             } else {
                 UE_LOG(BioGearsLog, Display, TEXT("Processed an action"));
         }   }
         _engine->advance_time(1);
         onStateUpdated.Broadcast(_engine->getState());
         onMetricsUpdated.Broadcast(_engine->getMetrics());
         onConditionsUpdated.Broadcast(_engine->getConditions());
         onTimeAdvance.Broadcast(_engine->getSimulationTime());
     } else {
        std::this_thread::sleep_for(16ms);
     }
     current_time = std::chrono::steady_clock::now();
     if (_engine_throttled) {
         while ((current_time - prev) < 1s) {
             std::this_thread::sleep_for(16ms);
             current_time = std::chrono::steady_clock::now();
    }    }
      _engine_running = false;
}
```

Driving BioGears 2/3

```
void UBioGearsEngineDriver::start()
{
   if (!_engineThread.joinable()) {
     _engine_thread_continue = true;
     _engineThread = std::thread{ &UBioGearsEngineDriver::engine_thread_main, this };
     runningToggled.Broadcast(_engine_thread_continue);

    } else if (!_engine_thread_continue) {
       _engineThread.join();
       _engine_thread_continue = true;
       _engineThread = std::thread{ &UBioGearsEngineDriver::engine_thread_main, this };
       runningToggled.Broadcast(_engine_thread_continue);

    }
}

void UBioGearsEngineDriver::pause_resume(bool state)
{
    _engine_paused = state;
    pausedToggled.Broadcast(_engine_paused);

}

void UBioGearsEngineDriver::stop()
{
   _engine_paused = true;
   _engine_throttled = false;
   _engine_thread_continue = false;

   pausedToggled.Broadcast(_engine_paused);
   runningToggled.Broadcast(_engine_thread_continue);
   throttledToggled.Broadcast(_engine_throttled);
}
```

Driving BioGears 3/3

```
void UBioGearsEngineDriver::join() {
  if (_engineThread.joinable())
  {
    _engineThread.join();
  }
}

void UBioGearsEngineDriver::set_throttle(int rate) {
  if (rate <= 1) {
    _engine_throttled = true;
  }  else if (rate > 1) {
    _engine_throttled = false;
  }
  throttledToggled.Broadcast(_engine_throttled);
}
```

Asynchronous Actions

```
void UBioGearsEngineDriver::action_apply_tourniquet(EBioGearsExtremity limb, EBioGearsTourniquet application) {
  _action_source->insert([&,limb,application](){return _engine->action_apply_tourniquet(limb, application); });
}

#include <biogears/cdm/patient/actions/SETourniquet.h>
bool UUE4BioGearsEngine::action_apply_tourniquet(EBioGearsExtremity limb, EBioGearsTourniquet application)
{
  auto tourniquet = SETourniquet();
  switch (limb)
  {
    case EBioGearsExtremity::LeftArm:
      tourniquet.SetCompartment("LeftArm"); break;
    case EBioGearsExtremity::RightArm:
      tourniquet.SetCompartment("RightArm"); break;
    case EBioGearsExtremity::LeftLeg:
      tourniquet.SetCompartment("LeftLeg"); break;
    case EBioGearsExtremity::RightLeg:
      tourniquet.SetCompartment("RightLeg"); break;
  }
  switch (application)
  {
    case EBioGearsTourniquet::Applied:
      tourniquet.SetTourniquetLevel(CDM::enumTourniquetApplicationLevel::Applied); break;
    case EBioGearsTourniquet::Misapplied:
      tourniquet.SetTourniquetLevel(CDM::enumTourniquetApplicationLevel::Misapplied); break;
    case EBioGearsTourniquet::None:
      tourniquet.SetTourniquetLevel(CDM::enumTourniquetApplicationLevel::None); break;
  }

  if (tourniquet.IsValid()) {
    _bg->ProcessAction(tourniquet); return true;
  } else {  return false;  }
}
```

Enums and Synthetic Environment Interfaces

Many development Platforms require inheritance from a platform object to allow types to seamlessly integrate with the platform.

Example
- Qt5 Qobject
- Unreal Engine 4 Uobject

For BioGears you have a few choices
- Modify BioGears and become incompatible with other BioGears Projects ☹

- Inherit from Synthetic Environment and BioGears Engine classes then splice objects when passing in BioGears 😐. All BioGears types have virtual deconstructors in general this is safe.

- Create Composite classes for features of BioGears your interface supports and try to limit using actual BioGears types to these integration objects.☺ Downside until serialization refactoring this does require implementing all BioGears Enum types

Platform Enum Adaptors

```
UENUM(Blueprintable, DisplayName="Biogears::Gender")
enum class EBioGearsGender : uint8 {
NONE,
MALE,
FEMALE,
OTHER,
};

UENUM(Blueprintable)
enum class EBioGearsExerciseState : uint8 {
SLEEPING,
RESTING,
SITTING,
STANDING,
WALKING,
RUNNING,
};
```

Platform Class Adaptors

```
USTRUCT(BlueprintType)
struct FBioGearsUrinalysisMicroscopic {
  GENERATED_BODY()

  public:
  UPROPERTY(BlueprintReadOnly)
    EBioGearsMicroscopicOpservationType type;
  UPROPERTY(BlueprintReadOnly)
    float red_blood_cells;
  UPROPERTY(BlueprintReadOnly)
    float white_blood_cells;
  UPROPERTY(BlueprintReadOnly)
    EBioGearsMicroscopicObservationAmount epithelia_cells;
  UPROPERTY(BlueprintReadOnly)
    float casts;
  UPROPERTY(BlueprintReadOnly)
    EBioGearsMicroscopicObservationAmount crystals;
  UPROPERTY(BlueprintReadOnly)
    EBioGearsMicroscopicObservationAmount bacteria;
  UPROPERTY(BlueprintReadOnly)
    EBioGearsMicroscopicObservationAmount trichomonads;
  UPROPERTY(BlueprintReadOnly)
    EBioGearsMicroscopicObservationAmount yeast;
};
```

Coming Soon

- UE4PLugin Public Release
- Biogears Visualizer Plugin Release

References

1. https://doc.qt.io/qt-5/qt5-intro.html
2. https://docs.unrealengine.com/en-US/index.html
3. https://github.com/BioGearsEngine
4. https://www.biogearsengine.com/
5. https://biogearsengine.com/documentation/annotated.html