Decoupling navigation and stepping in ACTS

#3449

Motivation

- Stepping and Navigation is tightly coupled in ACTS
- Navigator will call stepper for position and direction
- Therefore Stepper and it's state must be available to the Navigator

(Stepper also depends on the Navigator but that is a story for another day)

Navigator.hpp#L304-L322

```
ACTS_VERBOSE(volInfo(state)
             << "Slow start initialization through search.");
// current volume and layer search through global search
ACTS_VERBOSE(volInfo(state)
             << "Starting from position "
             << toString(stepper.position(state.stepping))
             << " and direction "
             << toString(stepper.direction(state.stepping)));
state.navigation.startVolume =
    m_cfg.trackingGeometry->lowestTrackingVolume(
        state.geoContext, stepper.position(state.stepping));
state.navigation.startLayer =
    state.navigation.startVolume != nullptr
        ? state.navigation.startVolume->associatedLayer(
             state.geoContext, stepper.position(state.stepping))
        : nullptr;
if (state.navigation.startVolume != nullptr) {
  ACTS_VERBOSE(volInfo(state) << "Start volume resolved.");
}
```

Navigator.hpp#L254-L262

```
/// @brief Initialize call - start of navigation
///
/// @tparam propagator_state_t The state type of the propagator
/// @tparam stepper_t The type of stepper used for the propagation
///
/// @param [in,out] state is the propagation state object
/// @param [in] stepper Stepper instance
template <typename propagator_state_t, typename stepper_t>
void initialize(propagator_state_t& state, const stepper_t& stepper) const {
```

Navigator.hpp#L272-L276

```
/// @brief Initialize call - start of navigation
///
/// @param [in,out] state the navigation state
void initialize(State& state, const Vector3& position,
                const Vector3& direction) const {
```

Proposal #1

- Break the dependency Navigator→Stepper by providing position and direction to the Navigator explicitly
- This can be communicated by the Propagator
- Only provide the Navigator State not the full Propagation State

➢ The navigators job is just to give us a next step size

Navigator.hpp#L272-L276

```
/// @brief Initialize call - start of navigation
///
/// @param [in,out] state the navigation state
void initialize(State& state, const Vector3& position,
                const Vector3& direction) const {
```

Proposal #1

- Break the dependency Navigator→Stepper by providing position and direction to the Navigator explicitly
- This can be communicated by the Propagator
- Only provide the Navigator State not the full Propagation State

➢ The navigators job is just to give us a next step size

BUT

- MultiEigenStepperLoop breaks that assumption
- There the navigator only provides the next surface and the Stepper will run intersections to get the step length

Proposal #2

- Reduce the Navigator to give the next surface candidate
- The Stepper can run the intersection
- We just need to inform the Navigator when we reached or missed a surface

Navigator.hpp#L349-L362

```
/// @brief Navigator estimateNextTarget call
///
/// Call options
/// (a) there are still surfaces to be resolved: handle those
/// (b) there no surfaces but still layers to be resolved, handle those
/// (c) there are no surfaces nor layers to be resolved, handle boundary
///
/// @param [in,out] state the navigation state
/// @param [in] position the current position
/// @param [in] direction the current direction
///
/// @return the next target surface intersection
SurfaceIntersection estimateNextTarget(State& state, const Vector3& position,
                                       const Vector3& direction) const {
```

Navigator.hpp#L398-L400

```
void registerSurfaceStatus(State& state, const Vector3& position,
                           const Vector3& direction, const Surface& surface,
                           IntersectionStatus surfaceStatus) const {
```

Proposal #2

- The Propagator communicates between Navigator and Stepper

Propagator.ipp#L51-L74

```
SurfaceIntersection nextTargetIntersection =
    m_navigator.estimateNextTarget(
        state.navigation, state.position,
        state.options.direction * state.direction);
if (nextTargetIntersection.isValid()) {
  m_stepper.updateSurfaceStatus(
      state.stepping, *nextTargetIntersection.object(),
      nextTargetIntersection.index(), state.options.direction,
      BoundaryTolerance::None(), s_onSurfaceTolerance, logger());
}

// Perform a propagation step - it takes the propagation state
Result<double> res = m_stepper.step(state, m_navigator);
if (!res.ok()) {
  ACTS_ERROR("Step failed with " << res.error() << ": "
                                 << res.error().message());
  // pass error to caller
  return res.error();
}
// Accumulate the path length
state.pathLength += *res;
// Update the position and direction
state.position = m_stepper.position(state.stepping);
state.direction = m_stepper.direction(state.stepping);
```

Propagator.ipp#L89-L99

```
if (nextTargetIntersection.isValid()) {
  IntersectionStatus postStepSurfaceStatus =
      m_stepper.updateSurfaceStatus(
          state.stepping, *nextTargetIntersection.object(),
          nextTargetIntersection.index(), state.options.direction,
          BoundaryTolerance::None(), s_onSurfaceTolerance, logger());
  m_navigator.registerSurfaceStatus(
      state.navigation, state.position,
      state.options.direction * state.direction,
      *nextTargetIntersection.object(), postStepSurfaceStatus);
}
```

Summary

- Proposal #2 effectively decouples the Navigator from the Stepper and the Propagation State
- This makes the components easier to reason about and to test
- Changes should be transparent to the user as the Propagator interface stays the same
- Makes Navigators template free
- Potentially saves on one or more intersections while approaching surface

Questions and feedback are very welcome!

Now or GitHub or Mattermost