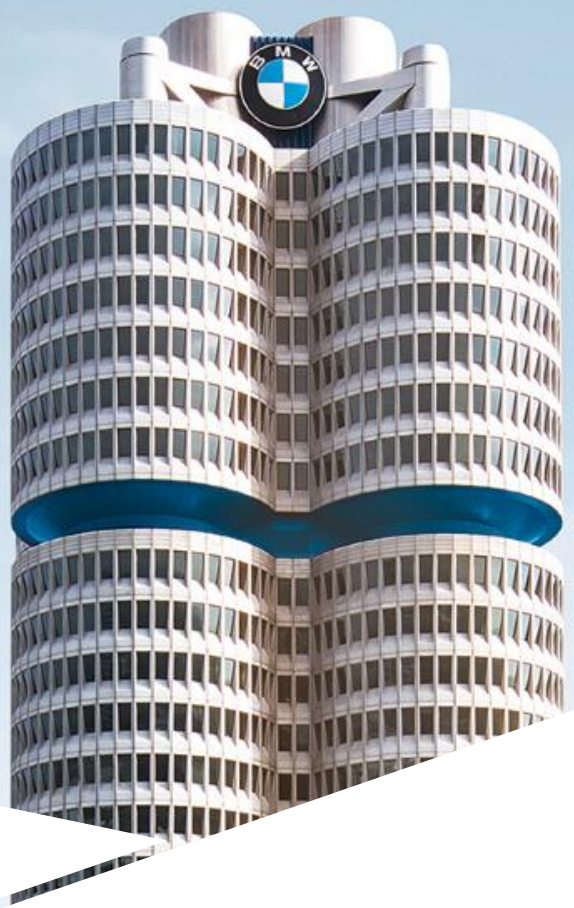
OPENPASS RELEASE V0.7
SPAWNER AND OBSERVER

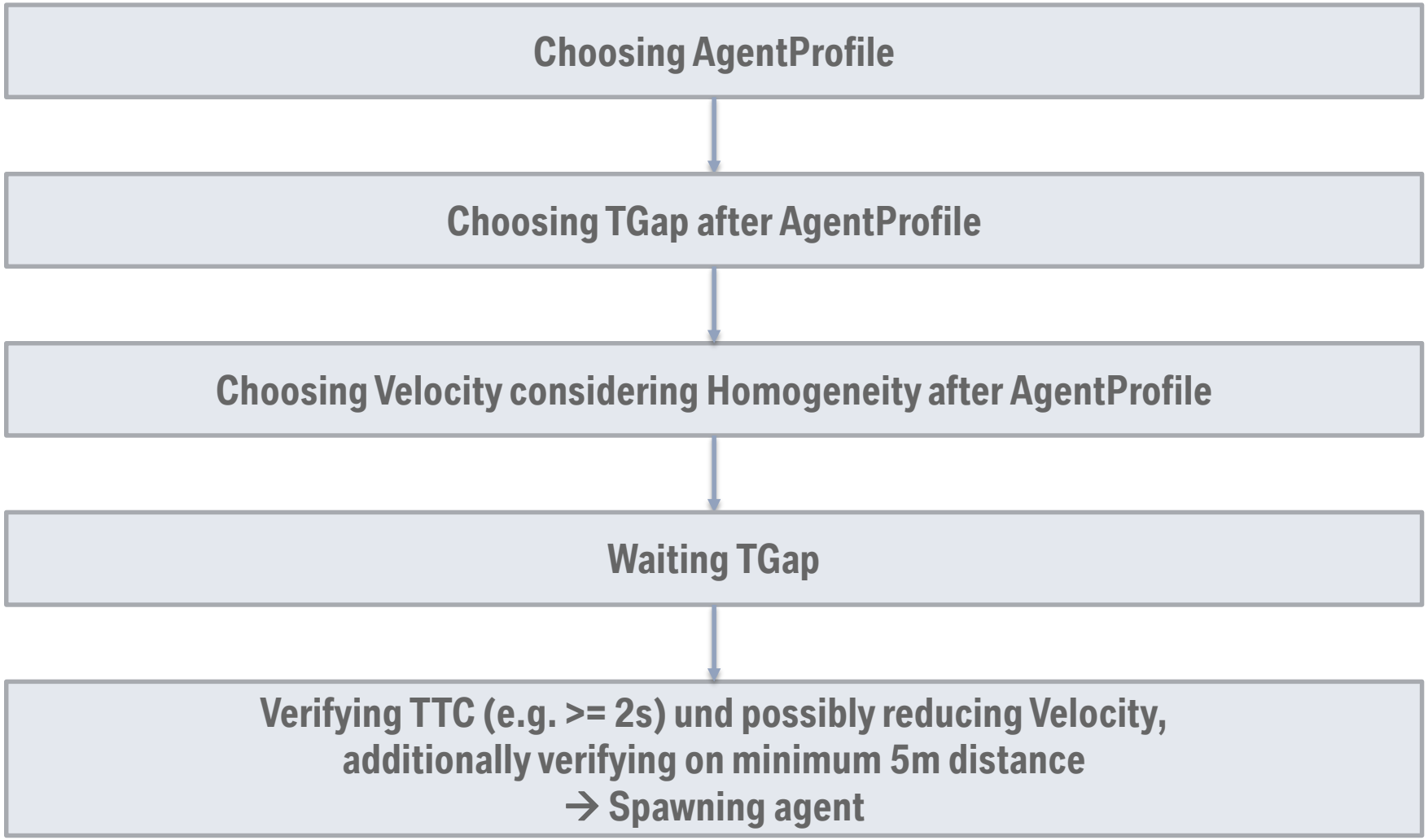


BMW | 06.07.2020

BMW GROUP THE NEXT 100 YEARS

Rolls-Royce Motor Cars Limited

NEW SPAWNER IMPLEMENTATION (I).



NEW SPAWNER IMPLEMENTATION (II).

- Spawn on multiple roads in urban environments
- Parameterize each spawn point individually
- Differentiate the spawn characteristics by agent type

slaveConfig.xml:

```
<Spawners>
  <Spawner>
    <Library>SpawnPointScenario_OSI</Library>
    <Type>PreRun</Type>
    <Priority>1</Priority>
  </Spawner>
  <Spawner>
    <Library>SpawnPointPreRunCommon_OSI</Library>
    <Type>PreRun</Type>
    <Priority>0</Priority>
    <Profile>DefaultPreRunCommon</Profile>
  </Spawner>
  <Spawner>
    <Library>SpawnPointRuntimeCommon_OSI</Library>
    <Type>Runtime</Type>
    <Priority>0</Priority>
    <Profile>DefaultRuntimeCommon</Profile>
  </Spawner>
</Spawners>
```

ProfilesCatalog.xml:

```
<ProfileGroup Type="Spawner">
  <Profile Name="DefaultPreRunCommon">
    <List Name="SpawnPoints">
      <ListItem>
        <String Key="Road" Value="1"/>
        <IntVector Key="Lanes" Value="-1,-2,-3,-4,-5"/>
        <Double Key="SStart" Value="0.0"/>
        <Double Key="SEnd" Value="1000.0"/>
      </ListItem>
    </List>
    <List Name="TrafficGroups">
      <ListItem>
        <Double Key="Weight" Value="1"/>
        <Reference Type="TrafficGroup" Name="DefaultPreRunCommon_Cars"/>
      </ListItem>
      <ListItem>
        <Double Key="Weight" Value="1"/>
        <Reference Type="TrafficGroup" Name="DefaultPreRunCommon_Trucks"/>
      </ListItem>
    </List>
  </Profile>
</ProfileGroup>
```

Other profiles are already partially adapted to the new structure („ProfileGroup“).

Weight instead of probability.

Next slide

NEW SPAWNER IMPLEMENTATION (III).

ProfilesCatalog.xml:

Previous slide

```
<ProfileGroup Type="TrafficGroup">
 <Profile Name="DefaultPreRunCommon_Cars">
   <List Name="AgentProfiles">
     <ListItem>
       <String Key="Name" Value="LuxuryClassCarAgent"/>
       <Double Key="Weight" Value="0.4"/>
     </ListItem>
     <ListItem>
       <String Key="Name" Value="MiddleClassCarAgent"/>
       <Double Key="Weight" Value="0.6"/>
     </ListItem>
   </List>
   <NormalDistribution Key="Velocity" Max="43.210" Mean="30.828" Min="18.446" SD="6.191"/>
   <LogNormalDistribution Key="TGap" Max="45.060" Min="1.174" Mu="1.984" Sigma="0.912"/>
   <DoubleVector Key="Homogeneity" Value="0.795, 1.0"/>
 </Profile>
 <Profile Name="DefaultPreRunCommon_Trucks">
   <List Name="AgentProfiles">
     <ListItem>
       <String Key="Name" Value="TruckAgent"/>
       <Double Key="Weight" Value="1.0"/>
     </ListItem>
   </List>
   <NormalDistribution Key="Velocity" Max="30.200" Mean="27.320" Min="24.440" SD="1.440"/>
   <LogNormalDistribution Key="TGap" Max="75.944" Min="0.346" Mu="1.634" Sigma="1.348"/>
   <Bool Key="RightLaneOnly" Value="true"/>
 </Profile>
</ProfileGroup>
```

Homogeneity describes the difference in velocity across neighbouring lanes.

If RightLaneOnly is true, agents only spawned on the rightmost lane.

NEW OBSERVER IMPLEMENTATION (I).

slaveConfig.xml:

```
<Observations>
  <Observation>
    <Library>Observation_Log</Library>
    <Parameters>
      <String Key="OutputFilename" Value="simulationOutput.xml"/>
      <Bool Key="LoggingCyclicsToCsv" Value="false"/>
      <StringVector Key="LoggingGroups" Value="Trace,Visualization,RoadPosition,Sensor,Driver"/>
    </Parameters>
  </Observation>
</Observations>
```

NEW OBSERVER IMPLEMENTATION (II).

Publish information to the DataStore:

```
extern "C" DYNAMICS_REGULAR_DRIVING_SHARED_EXPORT ModelInterface *OpenPASS_CreateInstance(
        std::string componentName,
        bool isInit,
        int priority,
        int offsetTime,
        int responseTime,
        int cycleTime,
        StochasticsInterface *stochastics,
        WorldInterface *world,
        const ParameterInterface *parameters,
        PublisherInterface * const publisher,
        AgentInterface *agent,
        const CallbackInterface *callbacks)
{
        PublisherInterface * const publisher,
        AgentInterface *agent,
        const CallbackInterface *callbacks)
{


 double engineMoment = GetEngineMoment(accPedalPos, gear);
 GetPublisher()->Publish("EngineMoment", engineMoment);
```

```
//! Interface which has to be provided by observation modules
class PublisherInterface
{
public:
    PublisherInterface(DataStoreWriteInterface *const dataStore) :
        dataStore(dataStore)
    {
    }

    PublisherInterface(const PublisherInterface &) = delete;
    PublisherInterface &operator=(const PublisherInterface &) = delete;

    PublisherInterface(PublisherInterface &&) = default;
    PublisherInterface &operator=(PublisherInterface &&) = default;

    virtual ~PublisherInterface() = default;

    /*!
     * \brief Writes information into a data store backend
     *
     * \param key[in]     Unique topic identification
     * \param value[in]   Value to be written
     */
    virtual void Publish(const openpass::datastore::Key &key, const openpass::datastore::Value &value)
    {
    }

    /*!
     * \brief Writes acyclic information into a data store backend
     *
     * \param value[in]   The acyclic event
     */
    virtual void Publish(const openpass::narrator::EventBase &event)
    {
    }

    virtual void Publish(const openpass::datastore::Key &key, const openpass::datastore::ComponentEvent &event)
    {
    }
```

NEW OBSERVER IMPLEMENTATION (III).

DataStoreReadInterface:

```
/*!
 * \brief Retrieves stored cyclic values from the data store
 *
 * \param[in]   time       Timestamp of interest
 * \param[in]   entityId   Entity's id
 * \param[in]   key        Unique topic identification
 */
virtual std::unique_ptr<CyclicResultInterface> GetCyclic(const std::optional<openpass::type::Timestamp> time, const std::optional<openpass::type::Enti

/*!
 * \brief Retrieves stored acyclic values from the data store
 *
 * \param[in]   time       Timestamp of interest
 * \param[in]   entityId   Entity's id
 * \param[in]   key        Unique topic identification
 *
 * \note Current implementation ignores time and entityId
 */
virtual std::unique_ptr<AcyclicResultInterface> GetAcyclic(const std::optional<openpass::type::Timestamp> time, const std::optional<openpass::type::En

/*!
 * \brief Retrieves stored static values from the data store
 *
 * \param[in]   key        Unique topic identification
 */
virtual Values GetStatic(const Key &key) const = 0;

/*!
 * \brief Retrieves keys at a specific node in the DataStore hierarchy.
 *
 * The key parameter has to be prefixed with "Cyclics/", "Acyclics/" or "Statics/" to
 * get access to the different types of stored elements.
 *
 * The following example will retrieve the list of agent ids participating in the current simulation run:
 * \code{.cpp}
 * const auto agentIds = dataStore.GetKeys("Statics/Agents");
 * \endcode
 *
 * The following example will retrieve a list of instantiated sensors for agent 3:
 * \code{.cpp}
 * const std::string keyPrefix = "Agents/3/Vehicle/Sensors";
 * \endcode
 *
 * \param[in]   key  Unique topic identification, including prefix
 */
virtual Keys GetKeys(const Key &key) const = 0;
```

NEW OBSERVER IMPLEMENTATION (IV).

Example for reading information from the DataStore:

```
void ObservationFileHandler::AddVehicleAttributes(const std::string& agentId)
{
    const std::string keyPrefix = "Agents/" + agentId + "/Vehicle/";

    xmlFileStream->writeStartElement(outputTags.VEHICLEATTRIBUTES);

    xmlFileStream->writeAttribute(outputAttributes.WIDTH, QString::number(std::get<double>(dataStore.GetStatic(keyPrefix + "Width").at(0))));
    xmlFileStream->writeAttribute(outputAttributes.LENGTH, QString::number(std::get<double>(dataStore.GetStatic(keyPrefix + "Length").at(0))));
    xmlFileStream->writeAttribute(outputAttributes.HEIGHT, QString::number(std::get<double>(dataStore.GetStatic(keyPrefix + "Height").at(0))));
    xmlFileStream->writeAttribute(outputAttributes.LONGITUDINALPIVOTOFFSET, QString::number(std::get<double>(dataStore.GetStatic(keyPrefix + "Longitudi

    xmlFileStream->writeEndElement();
}
```

```
void ObservationFileHandler::AddEvents()
{
    xmlFileStream->writeStartElement(outputTags.EVENTS);

    const auto events = dataStore.GetAcyclic(std::nullopt, std::nullopt, "*");

    for (const AcyclicRow& event : *events)
    {
        xmlFileStream->writeStartElement(outputTags.EVENT);
        xmlFileStream->writeAttribute(outputAttributes.TIME, QString::number(event.timestamp));
        xmlFileStream->writeAttribute(outputAttributes.SOURCE, QString::fromStdString(event.key));
        xmlFileStream->writeAttribute(outputAttributes.NAME, QString::fromStdString(event.data.name));

        WriteEntities(outputTags.TRIGGERINGENTITIES, event.data.triggeringEntities.entities, true);
        WriteEntities(outputTags.AFFECTEDENTITIES, event.data.affectedEntities.entities, true);
        WriteParameter(event.data.parameter, true);

        xmlFileStream->writeEndElement(); // event
    }

    xmlFileStream->writeEndElement(); // events
}
```