Hibernation of Advanced Railroad Trains (ArrT)

Step Three – ..... and Communicate them (aCt)

SRR Objects

This hibernation report tries to provide an overview of all SRR objects.

1 Introduction

SRR objects are nothing more than MIDAS objects.

However, they have the special feature that they need the Train Manager Extension of the SMUOS Framework, so they are closely linked to the SRR Framework.

- SRR Framework = SMUOS Framework + Train Manager Extension

SRR objects help

- To model tracks and switches
- To model rail vehicles (not yet implemented),
- and to link all of them together functionally (railway kinetics / kinematics).

SRR objects already existed at the first LAN party in March 2010 and with the re-design that has been going on since then, there are no big changes to them.

It will not be possible to couple vehicles at the end of Step 0033, bumpers and intersections will not exist yet and collisions will not be considered.

Nevertheless, there will be some small additions to the LAN Party # 3, which were not available in March 2010.

1. The master avatar container will be abandoned (moved to Simple Scene Controller)
2. Dynamic Modules will be supported
3. "Roles" will be abandoned and completely replaced by "Keys"
4. A MIDAS Base (MIB) will exist to ease implementation of MIDAS Objects
5. A "Beam to meeting place" function will exist
6. The SRR Framework will output status messages occasionally
7. A function will exist to reset all keys
8. There will be changes in the documentation and in the blogs
9. ~~??? It will be possible to use the SRR Framework from web spaces (monolithic layouts)~~
10. The base module of the SRR Framework will be replaced by the SMUOS Framework

2 Content

Inhaltsverzeichnis

3 Literature

[1] Dissertation; Hürlimann, D.; 2002;
Objektorientierte Modellierung von Infrastrukturelementen und Betriebsvorgängen im Eisenbahnwesen;
Institut für Verkehrsplanung und Transportsysteme, ETH Zürich, Zürich.

4 What is a colon graph?

If you want to simulate operations in the railway industry, then the dissertation of Dr. techn. D. Hürlimann at the ETH Zurich, [1], always a good clue.

Following these approaches, the SRR Framework models the track topology using a so-called colon graph.

- Each track section is represented by an edge and each edge has exactly one node at each of its two ends.
- Each node has exactly one adjacent node and each node terminates 0 or more edges.

So you can represent the most important track types.

The following track types will be available in Step 0033:

- Track section,
- Two-channel diverter.

The following track types can also be displayed with this approach, but will only be realized in a later step:

- Three-channel diverter,
- Crossing,
- Crossing diverter,
- Bumper

Other types of track, for which further concepts are necessary, include for example:

- Turntable, transfer table,
- Rail tracks on rail vehicles (for example, to transport another rail vehicle).

Figure 1 shows an example of a double-point graph for two track sections, a turnout and a buffer.

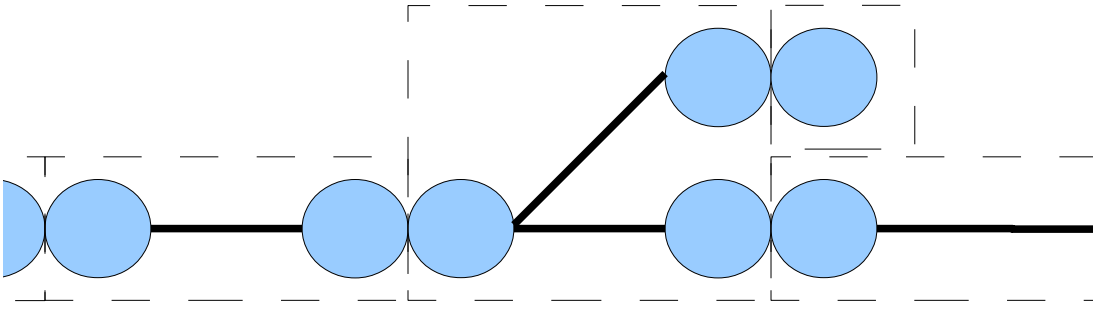

Figure 1: Track Elements in a Colon GRaph

The node representing the buffer does not terminate a single edge, the end nodes of a normal track segment each terminate one edge, and the node modeling a vertex terminates two edges.

In order to model such colon graphs, the SRR Framework provides the SRR objects for track and turnouts.

These are:

- SrrTrackEdge a track edge
- SrrTrackNode a track node
- MoosSwitchB a point machine
- SrrBasicTrackSection orchestrates two track nodes and one track edge
- SrrBasicTurnout2Way orchestrates three track nodes, two track edges and a turnout drive

You can learn more about them in chapter 6 ..

5 Topology versus Geometry

As you can easily see, the colon graph is just the logical topology of a track layout, but not geometric or even geographic features are shown, even if only the geometric length of an edge in meters.

For this purpose, the SRR Framework defines an interface with which you can assign geometrical properties to each edge (the SRR object SrrTrackEdge), the so-called "track geometry".

The SRR Framework comes with an "exemplary track geometry" that defines a "track geometry node" (TGN) and some finished track and turnout models that build on this "track geometry".

More about this "exemplary track geometry" can be found in chapter 7.

6 SRR Objects for Tracks and Turnouts (directory tmm/)

6.1 Overview

Figure 2 shows an overview of the relationships between the various SRR objects for track and points

- A "Basic Track Section" orchestrates exactly one "Track Edge" and two "Track Nodes"
- A "Basic Turnout 2-Way" orchestrates exactly two "Track Edges", three "Track Nodes" and one "Turnout Drive"
- Each track edge contains exactly one TGN

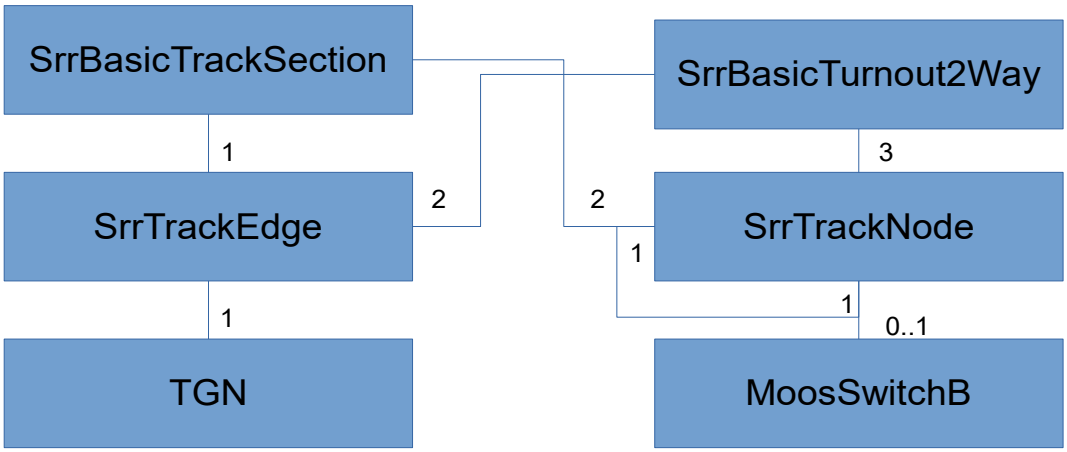

Figure 2: Overview about SRR Objects for Tracks and Turnouts

The track and turnout SRR objects only support the MOOs "LOADED", II and V, which means they can only be used as bound objects.

The fields of the uiObj interface do not offer the following fields:

- "universalObjectClass", "commParam" and "initialized",

while the following fields are actually offered:

- objId, modParam, disable, parentObj, attached, enabledOut, and dependentMobs.

6.2 SrrTrackNode

An "SrrTrackNode" SRR object models a track node in the colon graph.

It always has one of the two objects SrrBasicTrackSection or SrrBasicTurnout2Way as parent object and offers the following parameters:

Let N be the number of track edges terminated by this node.

The field name = 'neighborName' type = 'SFString' value = ''

contains the "objId" of the desired neighbor node with which the node is to be linked when "linking the tracks". This field is from accessType = "initializeOnly" and thus can only be set once.

The array <field name = 'trackEdges' type = 'MFNode' value = '' />

contains pointers to all N edges that are terminated by this node. This field is set to the correct values during the (re-) attachment of the parent object and can already be used when "linking the tracks".

The array name = 'isNodeA' type = 'MFBool' value = ''

For all N edges terminated by this node, it indicates whether it is the A-side or the B-side of the edge. This field is set to the correct values during the (re-) attachment of the parent object and can already be used when "linking the tracks".

The field name = 'neighbor' type = 'SFNode' value = 'NULL'

points as a pointer to the neighboring node of this node. This field is NULL if and only if the node is not linked to any neighbors. This field is set by "linking the tracks" and reset to "null" when the node is "leased". This also applies if the neighboring node is left behind by this node

The field name = 'turnoutSwitch' type = 'SFNode' value = 'NULL'

contains a pointer to the "turnout drive" when N is greater than 1. If N is less than or equal to 1, this pointer contains the value NULL.

The field name = 'gauge' type = 'SFFloat' value = '1.435'

contains the gauge of the track that is valid at this node. Neighboring nodes should always have the same gauge, but along one edge, a running gauge can be modeled by setting different gauges at the A-node and the B-node of the edge.

6.3 SrrTrackEdge

A "SrrTrackEdge" SRR object models a track edge in the colon graph.

It always has one of the two objects SrrBasicTrackSection or SrrBasicTurnout2Way as parent object and offers the following parameters:

The field name = 'trackNodeA' type = 'SFNode' value = 'NULL'

contains a pointer to the track node on the A side of the track edge. This field is set at the "Track linking" when the node is linked to its neighbor node. When the node is "unlinked", this field is reset to the value NULL. This also applies if the neighboring node is left behind by this node.

The field name = 'trackNodeB' type = 'SFNode' value = 'NULL'

contains a pointer to the track node on the B side of the track edge. This field is set at the "Track linking" when the node is linked to its neighbor node. When the node is "unlinked", this field is reset to the value NULL. This also applies if the neighboring node is left behind by this node.

The field name = 'trackGeometry' type = 'SFNode' value = 'NULL'

contains a pointer to a TGN. Each "normal" track edge must be assigned exactly one TGN, which deals with the geometric and geographic properties of the track edge.

The field name = 'exitViewpoint' type = 'SFNode' value = 'NULL'

contains a pointer to an X3D <Viewpoint> node.

If a user enters a rail vehicle, then one must "bind" a <Viewpoint>, which moves with the rail vehicle, so that the user is "taken" by the vehicle. Now, when the user leaves the vehicle, you have to bind a <Viewpoint> again, which rests opposite the landscape.

This field allows the scene author to associate each track edge with a <Viewpoint>, which is bound by the SRR Framework as soon as the user leaves the vehicle and when the vehicle is just entering this track edge.

6.4 Track Geometry Nodes (TGNs) e.g. SrrTrackSectionA.x3d

Through the TGN, the SRR framework accesses the geometric properties of a track edge.

The TGN can also perform calculations to give the surrounding track model a basis for the graphical representation of the track.

This is best seen by looking at the file tg / SrrTrackSectionA.x3d.

This implements a model of a track section that corresponds to a track edge using the "ABI track geometry". The "ABI track geometry" is defined in the TGN "SrrTrackGeometryABI.x3d".

Here you can see first the SRR objects and then the graphic elements, which then really become visible:

```
<!--  ***** SRR objects to instrument the track section *****  -->
          <ProtoInstance DEF='TrackSection' name='SrrBasicTrackSection'>
            <fieldValue name="dependentMobs">
              <ProtoInstance DEF="TrackNodeA" name="SrrTrackNode">
                <fieldValue name="gauge" value="0.545"/>
                <IS>
                  <connect nodeField="neighbourName" protoField="neighbourA"/>
                </IS>
              </ProtoInstance>
              <ProtoInstance DEF="TrackNodeB" name="SrrTrackNode">
                <fieldValue name="gauge" value="0.545"/>
                <IS>
                  <connect nodeField="neighbourName" protoField="neighbourB"/>
                </IS>
              </ProtoInstance>
              <ProtoInstance DEF="TrackEdge" name="SrrTrackEdge">
                <fieldValue name="trackGeometry">
                  <ProtoInstance DEF='TrackGeometry' name="SrrTrackGeometryABI">
                    <fieldValue name="trackElementLength" value="0.5"/>
                    <IS>
                      <connect nodeField="vectorA" protoField="vectorA"/>
                      <connect nodeField="vectorB" protoField="vectorB"/>
                      <connect nodeField="vectorI" protoField="vectorI"/>
                      <connect nodeField="normalA" protoField="normalA"/>
                      <connect nodeField="normalB" protoField="normalB"/>
                    </IS>
                  </ProtoInstance>
                </fieldValue>
                <IS>
                  <connect nodeField="exitViewpoint" protoField="exitViewpoint"/>
                </IS>
              </ProtoInstance>
            </fieldValue>
          </ProtoInstance>
<!--  ***** graphical representation of the track section *****  -->
          <Switch DEF="OnOffSwitch">
            <Shape>
              <TriangleStripSet DEF='VisualTrackGeometry' stripCount="3">
                <Coordinate DEF='VisualTrackCoordinates' point='0 0 0, 1 0 0, 0 1 0'/>
                <TextureCoordinate DEF='VisualTrackTextureCoordinates' point='0 0, 1 0, 0 1'/>
              </TriangleStripSet>
              <Appearance>
                <Material />
                <ImageTexture url='"../tg/trackElementA.jpg"'/>
              </Appearance>
            </Shape>
          </Switch>
        </Group>
```

The TGN (marked in red) is inserted here as the child of the SrrTrackEdge node and receives the

values "vectorA", "vectorB", "vectorI", "normalA" and "normalB" from the user of the model.

These values are then used to calculate a circle segment that is the base of the track section.

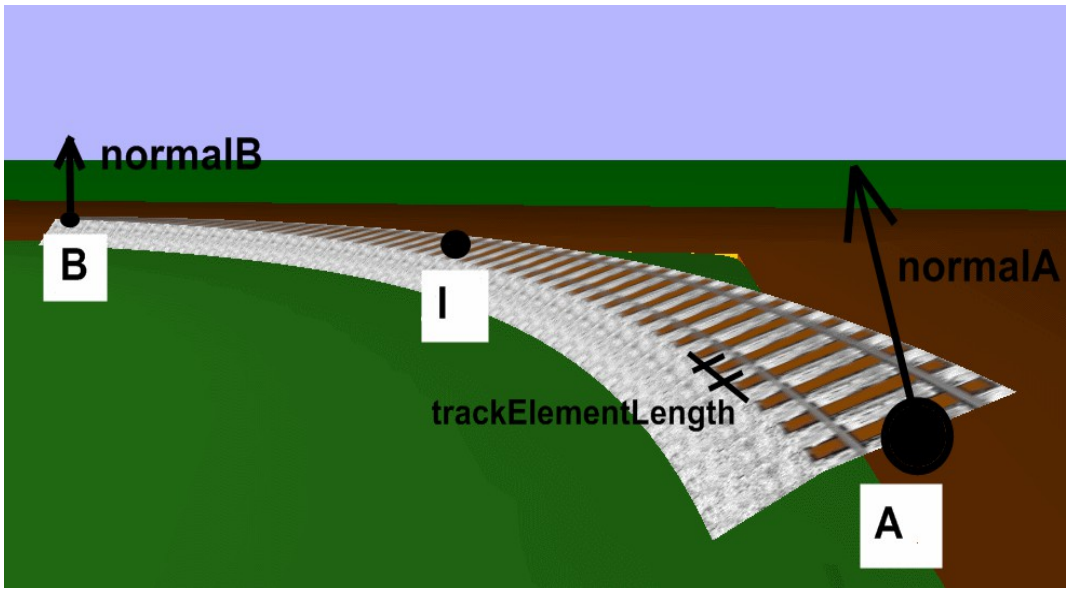

Figure 3: Parameter of the ABI track geometry

If you re-initialize the TGN, it takes these parameters and the parameter "trackElementLength", first calculates the circle section internally and the external parameter "length", divides the length "length" into a number of track elements (here thresholds) and outputs the parameters "trackCoordinates", "alongVectors" and "normalVectors".

These output parameters are arrays of vectors used by a script (not shown here) to set the geometric properties of the <TriangleStripSet /> before it is made visible using the OnOffSwitch switch.

The <ImageTexture /> node is scaled so that the image "tg / trackElementA.jpg" is displayed exactly "length" / "trackElementLength".

Each TGN (not just the TGN of the "ABI track geometry") MUST offer the following parameters:

The field name = 'objId' type = 'SFString' value = ''

serves to assign a <objId> to the TGN.

The field name = 'parentObj' type = 'SFNode' value = 'NULL'

serves to determine the <objId> s of the parent objects. Thus, a compound <objId> can be generated, e.g. for tracer issues.

The field name = 'modParam' type = 'SFNode' value = 'NULL'

is used to (re) initialize the TGN. Here, the term "(re) initialization" is used because a TGN is not really an SRR object.

With the field name = 'attached' type = 'SFNode'

the TGN must tell if the (re-) initialization was successful. The value initialized = NULL means that the (re) initialization has gone wrong.

The field name = 'enabledOut' type = 'SFBool'

must be set to false if the TGN has been disablemented.

The TGN must have the field name = 'length' type = 'SFFloat' value = '0.0'

set to the correct value at the latest during initialization.

With the field name = 'calculateAxleTransformation' type = 'SFNode'

If one instructs a TGN to represent an axis relative to the parent edge of this TGN. If you pass the "Value" pointer to an axis on the TGN, then the TGN must evaluate the properties of the axis and then set the Value.transformation.transformation property of that axis to the correct value.

With the field name = 'disable' type = 'SFTime'

you can disable the TGN.

6.5 SrrBasicTrackSection and SrrBasicTurnout2Way

These SRR objects are the "parent objects" to "orchestrate" all SRR objects of each track type.

SrrBasicTrackSection orchestrates 2 track nodes and one track edge into a general track section. The track edge must contain a TGN.

SrrBasicTurnout2Way orchestrates 3 track nodes and 2 track edges into a 2-way turnout. The track edges must each contain a TGN and the track node at the point point must contain a points drive.

6.6 Orchestration of the tracks and points

The systematics of the MIDAS objects (in the form of the prototype MibCore) ensures that the dependent MOBs are first initialized and attached - ie

- TGN (Track Geometry Node) - one per track edge,
- MoosSwitchB (point machine) - one per switch,
- SrrTrackNode - 2 pieces at a track section and 3 pieces at a switch and
- SrrTrackEdge - one at a track section and two at a switch,

before the "parent objects" are initialized and attached, ie

- SrrBasicTrackSection
- SrrBasicTurnout2Way

Since the module coordinator (exactly the TMM extension of the module coordinator) keeps both a list of all track edges of the module and a list of all module track nodes, these two nodes will announce themselves during their (re-) attachments to the MC and wait for them Success message before the "parent object" can be further initialized and attached.

SrrTrackEdge additionally checks if it actually contains a TGN.

Afterwards - during the attachment of the "parent object" - this sets some pointers in the Track Edge and Track Node objects, so that then the "linking of the tracks" can take place properly.

The tracks are still not linked, a "wheels of the wheels" is not yet possible.

6.6.1 Linking the tracks

The "Track Linking" will be implemented in Step 0033.11 of the SrrTrains v0.01 project.

6.6.2 Unlinking a node

The "de-linking of nodes" and "new linking of tracks" will be implemented in a step after Step 0033.

7 The "Example Track Geometry" (in the directory tg /)

The SRR framework (which builds on the SMUOS framework) has an intentional void.

From the beginning we have only implemented a very simple, one might say rudimentary, track geometry that reduces each track section to a circular arc.

Although this circular arc is at least spatially designed - so that you can model even hills and valleys in the track - to roller coasters - but the railroad specialist missing the beloved transition bows. These are currently not modelable.

To remedy this, we have "outsourced" the track geometry, so to speak.

This means that we have left a gap in the SRR framework, in which everyone can use his own track geometry, but have added an "example track geometry" to the SRR framework, including some models for track sections and points, namely the "rudimentary track geometry".

This is in the directory tg / and is also licensed with an LGPL.

8 SRR Objects for Rail Vehicles (in directory tmm/)

These will be implemented in step 0033.11 of the SrrTrains v0.01 project, WHICH IS CURRENTLY IN A "CLOSED" STATE. It can be re-opened by interested third parties.

Therefore we did not translate this chapter from German to English. So sorry.

8.1 Grundlegende Gedanken

8.1.1 Der Zustand eines Zuges

Prinzipiell sollen Schienenfahrzeuge und Züge über die Gleise bewegt werden, indem man einzelne Achsen über die Gleise bewegt, nämlich mit Hilfe des SRR Objektes SrrAxle.

Aus den translativen und den rotatorischen Positionen der Achsen sollen dann die Positionen der Drehgestelle und Fahrzeuge hergeleitet werden.

Dabei sollen alle Achsen eines Zuges insoferne gleichgeschaltet sein, als sie sich relativ zum Gleis mit derselben Geschwindigkeit bewegen. In Längsrichtung sei der Zug also ein ***starrer Stab***.

Diese Geschwindigkeit – die Geschwindigkeit des Zuges – und die translative Position der Achsen relativ zum Gleis soll von einem SRR Objekt berechnet werden, das für jeden Zug genau einmal existiert. Es ist das das SRR Objekt ***SrrTrainState***.

SrrTrainState entspricht einer "unsichtbaren Referenzachse", die den Bezugspunkt des Zuges definiert. Sobald und solange der SrrTrainState aktiv ist, liefert er gültige Werte für

- Modul,
- Track Edge,
- isAtoB und
- ess

der unsichtbaren Referenzachse. Damit ist die absolute Position der Referenzachse, also ***absPosTrain*** definiert.

SrrTrainState liefert darüber hinaus Werte für die Geschwindigkeit und die Beschleunigung des Zuges:

- ***velocityTrain***
- ***accelerationTrain***

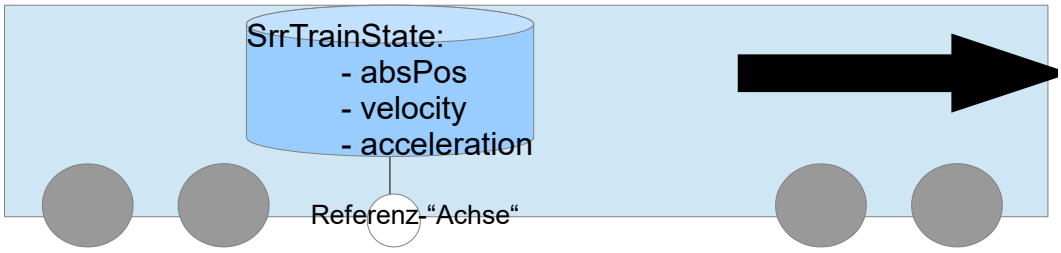

Abbildung 4: Zustand eines Zuges, bezogen auf eine "unsichtbare Referenzachse"

8.1.2 Zugriff auf alle Teile des Zuges

Jeder Zug hat ein astrales SRR Objekt ***SrrTrainBus***, das es allen Teilen des Zuges ermöglicht, innerhalb einer Szeneninstanz direkt miteinander in Kontakt zu treten.

Der SrrTrainBus speichert unter anderem einen MFFloat Wert, nämlich die "axleOffsets", sodass sich die absolute Position einer Achse [i] jederzeit zu

- absPosAxles[i] = absPosTrain + axleOffsets[i]

ergibt.

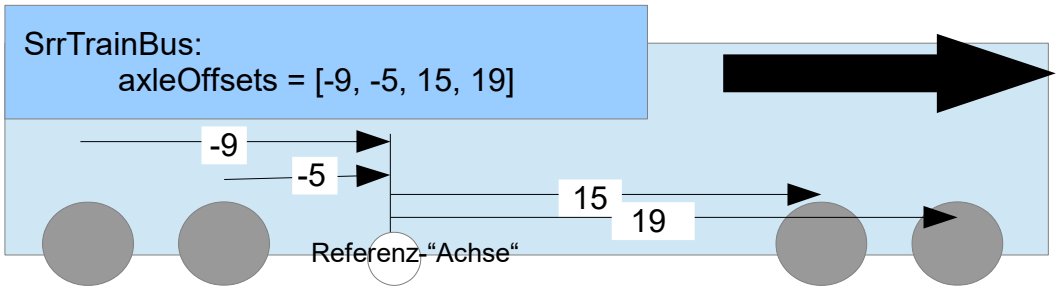

Abbildung 5: Beispiel für axleOffsets[]

Dieser MFFloat Wert "axleOffsets" und andere Parameterwerte, die der SrrTrainBus speichert, können sich immer dann ändern, wenn

- ein Zug "gegründet" wird, wenn
- zwei Züge zu einem Zug "vereinigt" werden, oder wenn
- ein Zug in zwei Züge "aufgetrennt" wird.

In SrrTrains gibt es keine Fahrzeuge, die nicht Teil eines Zuges sind.

Es kann aber natürlich Züge geben, die nur aus einem Fahrzeug bestehen, und das auch noch ohne Antrieb (z.B. abgestellte Waggons).

Die oben genannten "Prozeduren", also ***Gründung, Vereinigung und Trennung von Zügen***, werden aus sogenannten "Sub-Prozeduren" zusammengesetzt. Darüber später mehr.

8.1.3 Antriebe und Führerstände

Nun wollen wir aber auch in der Lage sein, ein Schleudern oder ein Blockieren einzelner Antriebe – oder sogar einzelner Achsen – zu modellieren.

Wir werden also die SrrAxle Objekte nicht direkt vom SrrTrainState ansteuern.

Deshalb fassen wir immer eine oder mehrere Achsen mit Hilfe eines Antriebs zusammen.

Das SRR Objekt ***SrrDrive*** enthält also immer ein oder mehrere SRR Objekte ***SrrAxle***.

Jedes SrrAxle Objekt ist in genau einem SrrDrive Objekt enthalten, ausgenommen die Referenzachse, die direkt dem SrrTrainState Objekt zugeordnet ist.

- Der SrrTrainState verteilt die deltaEss Events also an alle SrrDrives, die sie dann an die SrrAxles weiterleiten.
- Die rotatorische Position bekommen die SrrAxle Objekte direkt von den SrrDrive Objekten.

Jetzt bleibt noch die Frage, wie die Beschleunigung des Zuges zustande kommt.

- Dazu liefert jedes SrrDrive Objekt eine Kraft eff an den SrrTrainState

Jeder Antrieb wird von höchstens einem Führerstand gesteuert. Jeder Führerstand kann beliebig viele Antriebe steuern.

Wir haben also folgenden Ablauf der Eisenbahnkinetik:

A) Jedes SrrDrive Objekt wird von höchstens einem Führerstand gesteuert und berechnet kontinuierlich den Kraftbeitrag zur Beschleunigung des Zuges.
Dabei wird jeder Achse ein bestimmter Anteil der Gesamtmasse zugeordnet, damit die Wirkung der Schwerkraft gleich subtrahiert werden kann

B) Das SrrTrainState Objekt berechnet aus den Kraftbeiträgen und der Gesamtmasse des Zuges eine Beschleunigung und eine Geschwindigkeit, daraus ein deltaEss Event

C) Das deltaEss Event wird vom SrrTrainState an alle SrrDrives und damit an alle SrrAxles verteilt. Die Referenzachse wird von SrrTrainState extra berücksichtigt

D) Aus den deltaEss Events ergeben sich die translatorischen und rotatorischen Positionen der Achsen

E) Die translatorischen und rotatorischen Positionen der Drehgestelle, Fahrzeuge und sonstigen Teile des Zuges werden aus den translatorischen und rotatorischen Positionen der Achsen berechnet

F) Um die rotatorische Position der Achsen kümmern sich die SrrDrive Objekte

8.1.4 SrrVehicleTrains versus SrrTrains

Da wir in Step 0033.11 das Kuppeln und Entkuppeln von Fahrzeugen noch gar nicht implementieren, können wir uns vorerst auf "one-vehicle-trains" zurückziehen.

Anstatt also einen Zug aus mehreren UBOs zusammenzusetzen (aus einem "Zug", aus mehreren "Fahrzeugen" und aus "Fahrzeug-zu-Zug-Beziehungen"), werden wir immer nur einzelne UBOs erzeugen, löschen, laden oder entladen.

Trotzdem werden wir in diesem Paper einige Gedanken zu zusammengesetzten Zügen wälzen.

8.2 One-Vehicle-Trains

Ein "one-vehicle-train" ist ein UBO der UOC "SrrVehicleTrains", enthält aber auch schon SRR Objekte, die später für zusammengesetzte Züge Verwendung finden sollen.

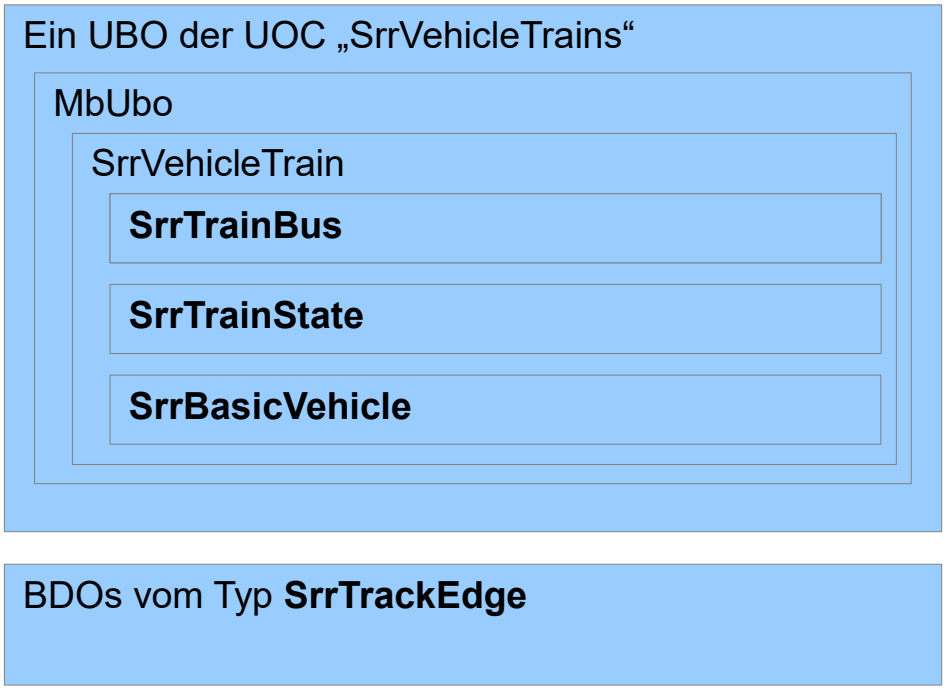

Figure 1: Prinzipieller Aufbau eines "SrrVehicleTrains" UBO

MbUbo ist die Model Base für UBOs.

SrrVehicleTrain ist das "Master" SRR Objekt für "one-vehicle-trains" und es orchestriert

- ein MIDAS Objekt ***SrrTrainBus***,
- ein MIDAS Objekt ***SrrTrainState*** und
- ein MIDAS Objekt ***SrrBasicVehicle*** mit seinen "üblichen" dependent objects
  - ***SrrCabs*** + ***SrrCab (N mal)***
  - ***SrrDrives*** + ***SrrDrive (N mal)***
  - ***SrrAxle (N mal)***

SrrTrainBus kann auf alle Elemente des Zuges innerhalb einer Szeneninstanz zugreifen. Alle Elemente des Zuges innerhalb einer Szeneninstanz können auf den SrrTrainBus zugreifen.

SrrCabs enthält N Cabs eines Fahrzeuges.

SrrDrives enthält N Drives eines Fahrzeuges.

Jedes SrrDrive enthält N SrrAxles.

8.3 Zusammengesetzte Züge (not yet implemented)

Dieses Kapitel hält erste Ideen für zusammengesetzte Züge, die auf folgende SRR Objekte aufbauen. Diese SRR Objekte sollen sowohl für "one-vehicle-trains" als auch für zusammengesetzte Züge verwendet werden:

- MbUbo.....generelle Basis für UBOs
- SrrTrainBus.....SRR Objekt genau ein mal für jeden Zug (MOO III)
- SrrTrainState.....SRR Objekt genau ein mal für jeden Zug (MOO IV)
- SrrBasicVehicle.....SRR Objekt genau einmal für jedes Schienenfahrzeug
- SrrDrives + SrrDrive + SrrAxle.....SRR Objekte für jedes Schienenfahrzeug
- SrrCabs + SrrCab.....SRR Objekte für jedes Schienenfahrzeug

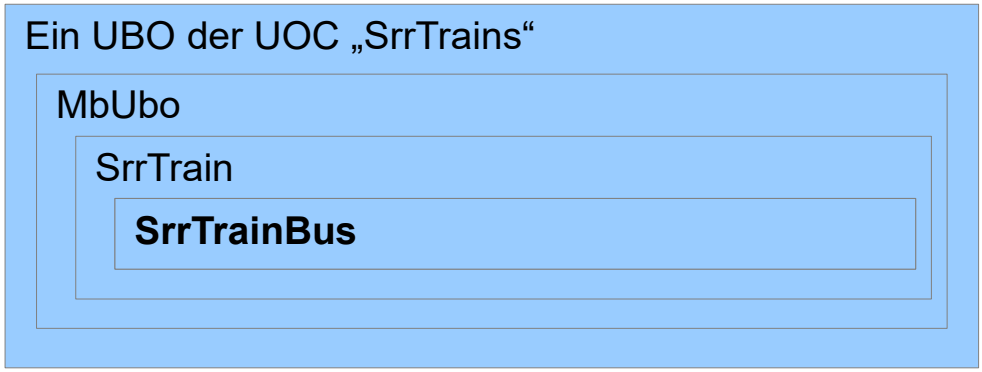

Abbildung 6: Prinzipieller Aufbau eines "SrrTrains" UBO

SrrTrain ist das "Master" SRR Objekt für zusammengesetzte Züge, es orchestriert

- ein SRR Objekt ***SrrTrainBus***,
- stellt die Verbindung zum SrrTrainState her und
- stellt die Verbindung zu allen SrrBasicVehicles her

Der Train State wird in einem externen UBO, dem sogenannten "***Abstract State Holder (ASH)***" gehalten (siehe unten). SrrTrain muss diesen externen SrrTrainState mit dem SrrTrainBus verknüpfen.

Die Fahrzeuge werden in externen UBOs der UOC "SrrRailVehicles" gehalten (siehe unten). Sie enthalten auch je ein SRR Objekt vom Typ SrrBasicVehicle, wobei SrrTrain den Kontakt zwischen den SrrBasicVehicles und dem SrrTrainBus herstellen muss.

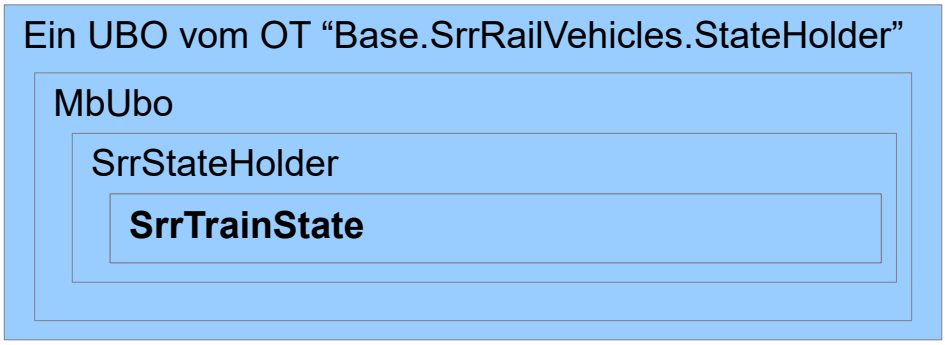

Abbildung 7: Prinzipieller Aufbau des "Abstract State Holder (ASH)"

SrrStateHolder ist das "Master" SRR Objekt für den "Abstract State Holder (ASH)".

Die Object ID des ASH wird von der Object ID des SrrTrain UBO hergeleitet.

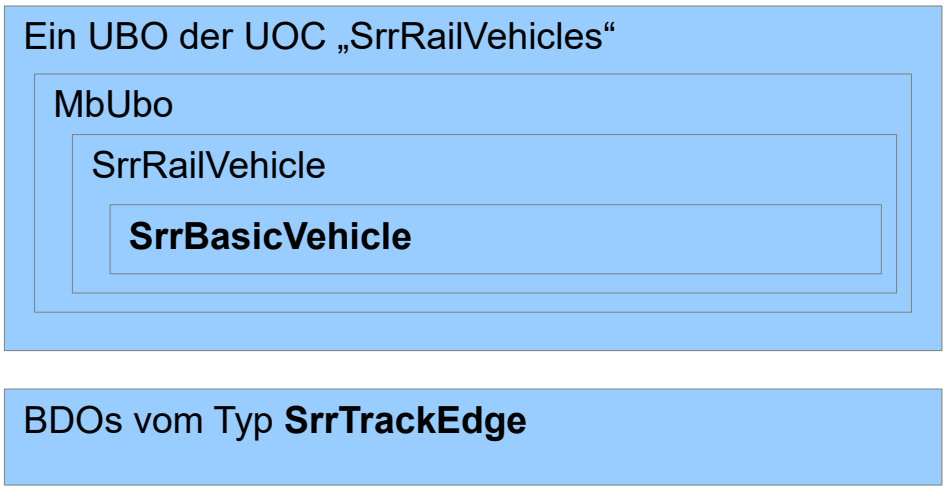

Abbildung 8: Prinzipieller Aufbau eines "SrrRailVehicles" UBO

SrrRailVehicle ist das "Master" SRR Objekt für Fahrzeuge in zusammengesetzten Zügen.

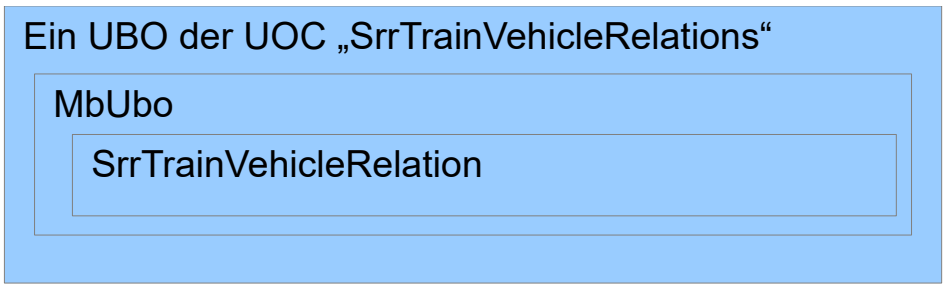

Abbildung 9: Prinzipieller Aufbau eines "SrrTrainVehicleRelations" UBO

SrrTrainVehicleRelation fügt Fahrzeuge (SrrRailVehicle) in Züge (SrrTrain) ein.