Simulated Railroad Framework, http://simulrr.sourceforge.net
Synopsis: 000_Synopsis

This file valid for step 0033.10
Issue Date: 2017-03-17

The SRR Controller (Train Manager) (*not finished* - might change completely)

1 Synopsis

The SRR Controller (Train Manager) is a part of the SRR Framework that is intended to be used in the frame of an SrrTrains layout.

The SRR Controller (Train Manager) is an extension of the Simple Scene Controller (121_SimpleSceneController), which is a part of the SMUOS Framework.

Where the SMUOS Framework is open to use cases designated by the term "Simple Multiuser Scenes", on the other hand the SRR Controller (Train Manager) is specialized to the application for railroad simulation.

The SRR Controller (Train Manager) consists of two X3D prototypes that are contained in the files SrrControlTm.x3d (client and server) and SrrControlTmNs.x3d (network sensor).

Additionally, the SRR Controller uses the "core functions" from the SscCore prototype, which is contained in the file SscCore.x3d.

2 The Purpose of the SRR Controller (Train Manager)

The train manager extension enables the SRR/SMUOS Framework to support bound models of tracks and turnouts and to support unbound models of rail vehicles. Rail vehicles can be combined to trains (*rail vehicles and trains not yet implemented*).

The example SRR Objects that accompany the train manager extension, provide the support of the bound models for tracks and turnouts and the support of the models of rail vehicles (*rail vehicles and trains not yet implemented*).

The support for the bound models of tracks and turnouts is mainly located in the module coordinator (see 221_ModuleCoordinatorTm) and in the associated SRR Objects.

Please find some general explanations about unbound models at 052_UnboundModels.

Note: currently the support for unbound models is not implemented, only the bound models of tracks and turnouts are supported, rail vehicles do not exist yet.

3 External View

The SRR Controller (Train Manager) is an SSC Extension, hence it is implemented according to the extensibility concepts of SMUOS (see 051_Extensibility).

3.1 MOO Diagram (MOOD)

SSC Extensions are initialized together with the SSC Base.
Two phases of startup are relevant, the "initialization" and the "start activity" phases.

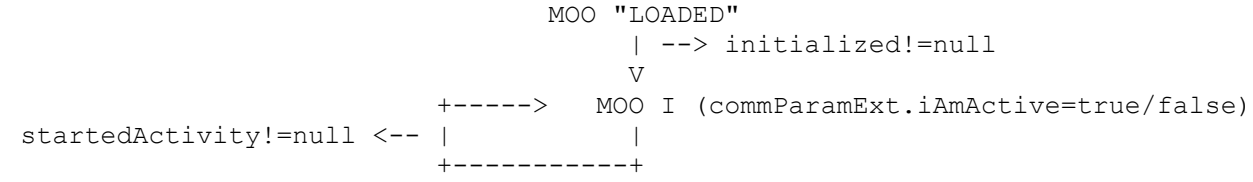

Figure 1: State Diagram of the SRR Controller (Train Manager)

3.2 State Event Matrix

Following events in following states lead to execution of following use cases:

Event \ State	MOO "LOADED"	MOO I (inactive)	MOO I (activated)
commParam	Initialization	-	-
initializeStateAnd StartActivity	-	StartActivity	-
startActivity	-	StartActivity	-
traceLevel*Request	SetTraceLevels	SetTraceLevels	SetTraceLevels
registerVehicleTypes	-	-	VehicleRegistration
announceModCoord	-	ModCoorAnnouncement	ModCoorAnnouncement
deannounceModCoord	-	ModCoorAnnouncement	ModCoorAnnouncement
announceReplicator	Replicators	Replicators	Replicators
deannounceReplicator	Replicators	Replicators	Replicators

Table 1: State Event Matrix of the SRR Controller (Train Manager)

3.3 Use Cases

The State Event Matrix displays, which use case is available via which field of the SrrControlTm prototype in which mode of operation (MOO), either MOO "LOADED" or MOO I (initialized). The field commParamExt.iAmActive is taken as an additional differentiation.

The following sections explain each use case in detail

3.3.1 Initialization

When the user (frame author) has added the SRR Controller (Train Manager) to the "sscBaseExtensions" (MFNode) field of the SSC Base, then the SSC Base will try to initialize the train manager by sending the commParam at some time during the overall initialization of the SSC.

The train manager will use the services of "SSC Core" to initialize its network sensor and the contained SSC dispatchers. However, as long as the SSC Base does not request "start activity", the train manager will store received values of the global states "Vehicle Type List" (VTL) and "Train/Vehicle State" (TV State) without processing them.

When all mandatory SSC extensions are successfully initialized and all optional SSC extensions are successfully or unsuccessfully initialized, then the SSC Base will continue its own initialization. This will eventually lead to "start activity".

3.3.2 Start Activity

Quite at the end of its own initialization, the SSC Base will trigger "start activity" of all successfully initialized SSC extensions.

Here two options exist.

Either the scene instance is the first scene instance and all extensions must initialize their global states – in this case the SSC Base will send "initializeStateAndStartActivity" – or the global state is already valid and the extensions just need to initially process the global state and start their activity – in this case the SSC Base will send "startActivity".

The fields "initializeStateAndStartActivity" and "startActivity" are actually handled by the contained "SSC Core" prototype, however the specific actions are implemented by the SRR Controller (Train Manager):

At the first option, the train manager will send an initial value of the global states. After reception the SSC Core prototype will request the processing of the states. The train manager will initially process the global states and SSC Core will report "startedActivity" to the SSC Base.

At the second option, the train manager will immediately process the stored values of the received global states and SSC Core will report "startedActivity" to the SSC Base.

3.3.3 SetTraceLevels

The frame can set the trace levels of the train manager (SrrControl, TrainControl and Modules). The semantics of the SrrControl and TrainControl trace levels are very similar to those of the trace levels SscBase and CommControl of the SSC Base, as the semantics of the Modules trace levels.

Please refer to 015_Tracer for a more detailed description of the trace levels of the SSC Base.

3.3.4 ModCoorAnnouncement

When a module is initialized, and when the module has got a train manager extension of the module coordinator, then the module coordinator extension announces itself at the SRR Controller (Train Manager).

When a module is being disabled, and when the module has got a train manager extension of the module coordinator, then the module coordinator extension deannounces itself at the SRR Controller (Train Manager).

???Hence the SRR Controller (Train Manager) is always informed about the being ???loaded/unloaded of all relevant modules and can decide whether to load/unload ???unbound models, which are assigned to this or that module.

3.3.5 VehicleRegistration

Before an unbound model (vehicle model) can be loaded – loading is done locally in each scene instance -, the URL of the model needs to be distributed to all scene instances and a "Vehicle Type Id" needs to be assigned to the "Vehicle Type".

Additionally, the vehicle is assigned a list of "Vehicle Type Categories" that are used to display only subsets of vehicle Ids at the replicators.

The user (frame author) sends an MFString value to the uiControl field "registerVehicleTypes", where each element of the list is an SFString value of the format

```
'<vehicleTypeId>;<VehicleTypeCategories>;<vehicleTypeUrls>'
```

The vehicle Type Id needs to be a unique string identifying the vehicle type.

The vehicle type categories are a comma-separated list of strings that describe the categories of the vehicle.

The vehicle type URLs are a comma-separated list of strings that define the set of URLs, which will be searched when loading the model.

Neither commas (',') nor semicolons (';') are allowed in any of these fields, but the separating commas and semicolons.

3.3.6 Replicators

When an unbound model (vehicle model) is being created from its vehicle type (* not yet implemented *), then it needs to be assigned an initial state (position and velocity).

In the case of vehicle models, the initial state is derived from a so-called "Replicator".

That means: a given position in the track layout is referenced by an extended object ID (i.e. the extObjId of the replicator). The replicator itself describes the position within the track layout (name of the parent edge, position relative to the parent edge, direction) and the initial velocity.

Furthermore the replicator provides a graphical interface to the user, where he can select one from the (filtered) list of registered vehicle type ids, which shall be created at this replicator.

Everytime, when a replicator announces/deannounces itself at the SRR Controller (Train Manager), then SRR Controller (Train Manager) updates the list of "registeredReplicatorIds".

Everytime, a new vehicle is registered, the SRR Controller (Train Manager) updates the list of "registeredVehicleIds" in all replicators (filtering by "categories").

4 Internal View

tbd

5 Additional Info

tbd