

Simulated Railroad Framework, <http://simulrr.sourceforge.net>
Synopsis: [000_Synopsis](#)

This file valid for step 0033.11
Issue Date: tbd.

The Frame and the Modules

=====

1 Synopsis

Each Simple Multiuser Scene contains exactly one frame. The frame is the set of all common functions that support the modules and the models when providing virtual senses and skills to the user.

The concepts of SMS require that modules and models are built by declarative 3D principles. However this does not hold for the frame. The frame may be any kind of software that integrates the top level modules into a VR/AR platform.

Within the present Concepts' Descriptions there is not a specific paper for the frame, reason is the above mentioned fact that frames may be very different and it's hard to specify any requirements for the frame.

However, one of the responsibilities of the frame is to load, initialize and optionally disable and unload the top level modules.

Simple Multiuser Scenes are built by modules (an SMS comprises a frame and at least one module), each module can be provided by a different author.

Modules are visited or inhabited by models, where the X3D standard and the present concepts guarantee interoperability and re-usability of models, modules and frames.

2 Purpose

Why is an SrrTrains layout / a Simple Multiuser Scene built by modules?
Well, there are several reasons.

- 1) a *real* model railroad can be built by modules, too.
- 2) combining modules into an SMS enables more than one author to contribute to one SMS.
- 3) slicing an SMS into modules can save memory and/or CPU resources,
 - a) because a module can be deactivated (switch off animation/simulation)
 - b) because a module can be unloaded completely (i.e. it disappears)

2.1 Static Modules

If a module is referenced, loaded and initialized immediately after the initialization of the Simple Scene Controller, and if the module is never unloaded at runtime of the scene instance, then we call this module a "static" module.

Static modules can be deactivated or activated at any time. Activation/deactivation is a local category, i.e. it happens only within the scope of one scene instance. A module can be active in one scene instance and inactive in another scene instance.

The SRR/SMUOS Framework uses single character indications for the activity of static modules:

- 'o' the module has not yet been initialized
- '-' the module is inactive (in this scene instance)
- '+' the module is active (in this scene instance)
- '*' the module has got the MOC role (in this scene instance)

Only one instance of the module can have the MOC role, it is active then, too.

2.2 Dynamic Modules

If a module is loaded or unloaded on demand, during the lifetime of the scene instance, then we call this module a "dynamic" module. It is the task of the frame to load/unload dynamic modules and to trigger their initialization upon loading and their disabling before unloading.

Loading/unloading is a local category, i.e. it happens only within the scope of one scene instance. An exception to this rule is the global deregistration of a module, when the Simple Scene Controller triggers the unloading of the module in all scene instances.

Dynamic modules can be deactivated or activated at any time, given they have been loaded and initialized. Activation/deactivation is a local category, i.e. it happens only within the scope of one scene instance. A module can be active in one scene instance and inactive in another scene instance.

The SRR/SMUOS Framework uses single character indications for the activity of dynamic modules:

- 'o' the module has not yet been loaded/initialized, or it has been unloaded
- '-' the module is inactive, but has been loaded (in this scene instance)
- '+' the module is active (in this scene instance)
- '*' the module has got the MOC role (in this scene instance)

Only one instance of the module can have the MOC role, it is active then, too.

2.3 Dependent Modules (aka Moving Modules)

Dependent modules are an additional concept. Please refer to [054_MovingModules](#).

2.4 Registration and Deregistration of Modules

Registration is the process of

- assigning a moduleIx (SFInt32) to a moduleName (SFString)
- reserving space in the "Communication State" (commState) for the data of the module

The moduleIx of a module is a global category. It has the same value for a given module in all scene instances.

Implicit Registration is done by the SRR/SMUOS Framework automatically, during the announcement of the module at the Simple Scene Controller. Registration is only done, if the module has not been registered yet.

Explicit Registration is a feature, that can be used by the frame to learn the moduleIx of a module prior to loading and initializing it.

Explicit registration is useful for dynamic modules, because in many cases the frame will need to store data of the dynamic module in some arrays, before the module is actually loaded and initialized. With this feature, the frame can use the moduleIx as an index into these arrays.

Modules that have been implicitly registered, are automatically deregistered, when the last instance of the module has been unloaded.

Modules that have been explicitly registered, will stay registered, until the frame deregisters them explicitly.

3 External View

3.1 The Fields of uiModule

uiModule is the external interface ("minimum interface") of an SrrTrains/SMS module. The frame (actually the module wrappers) uses modules via the uiModule interface.

3.1.1 commParam

After having loaded the module (and after the SSC has been initialized), the frame (actually the module wrapper) sends a reference to the common parameters to this field and hence triggers the initialization and the attachment of the module.

That means: the module coordinator attaches himself - and the module - to the Simple Scene Controller after all parts of the module coordinator have been initialized.

Hence the module can take access to the MAM, which is maintained by the Simple Scene Controller, and which allows the module to get activated or deactivated and to attain the MOC role.

Hence the MIDAS objects that will be attached to the module, can maintain their activity and their OBCO role.

3.1.2 moduleName

BEFORE triggering the initialization, the frame (actually the module wrapper) sets the name of the module via this field. Please refer to [011_NamingRules](#) for further information.

3.1.3 mwParam

BEFORE triggering the initialization, the frame (actually the module wrapper) adds a reference to the module wrapper parameters. This is necessary, because the module coordinator needs access to the module wrapper parameters.

The module wrapper parameters hold parameters that are published by the frame on a "per module" basis (e.g. gravity).

3.1.4 initialized

The module reports successful (initialized != NULL) or unsuccessful (initialized == NULL) initialization and attachment at this field.

3.1.5 disable

The frame (actually the module wrapper) disables and detaches the module, before he actually unloads the module, by setting the SFTIME field to a value "now".

4 Internal View

Typically a module is implemented in a separate file, which contains the <Scene>

- with a <ProtoDeclare> with the definition of the module
- with a <ProtoInstance> of the module

The <ProtoDeclare> contains the definition of the module, typically comprising

- the module coordinator (a <ProtoInstance>)
- some landscape
- some viewpoints and their avatar container(s)
- references to static models
- intrinsic models (incl. References to MIDAS Objects)
- <ROUTE>s to route the "modParam" event from the module coordinator to the static models and to the MIDAS Objects

Some of the fields of the uiModule interface (see chapter 3) can be provided by directly <connect>ing the fields of the module coordinator with the uiModule interface. Please refer to the description of the module coordinator at [201_ModuleCoordinator](#) and at [221_ModuleCoordinatorTm](#).

4.1 Authoring Support

The modules of the demo layout have been constructed mainly with the help of X3D-Edit, some parts (mainly the landscape) were generated with the help of Blender and Gimp.

It's planned to provide Blender Python Scripts together with the SRR Framework, so that modeling SrrTrains modules should become easier in the future.

5 Additional Info

If a frame wants to instantiate and use a top level module (either statically or dynamically), or if a model wants to instantiate and use a dependent module (always statically), then it need not instantiate and use the modules directly, via the uiModule interface (see chapter 3), but it may use some helpers, which are provided with the SMUOS Framework, i.e.

- Module Wrappers to put a transformation "around" the module and
- The SMS Little Loader to load modules dynamically.

These helpers are not actually a part of the SMS Concept, but it can be assumed that similar frameworks as the SMUOS Framework will provide similar means to instantiate and use SMS Modules.

The Module Wrappers and the SMS Little Loader are actually some basic X3D prototypes of the SMUOS Framework and they are explained in the paper [101_SmsBase](#).