SrrTrains v0.01

Notes about the **Inofficial "Consummatum Est" Release,**

which is currently available at
https://lc-soc-lc.at/host/DownloadArea.htm

Christoph Valentin, 2015-12-18

1 Tables of Contents

Table of Contents

Index of Drawings

Table Index

2 Introduction

The concept SrrTrains (so-called "DIY-virtual-multiplayer-model-railroad") is accompanied by quite a bunch of bits and bytes, and they are available via several .zip files, where each .zip file contains different parts of the software.

Older releases of SrrTrains software are still available at a folder at mediafire.com (see the links at the https://lc-soc-lc.at/host/DownloadArea.htm to get connected).

The current release is an inofficial release with the nick name "Consummatum est". Currently (as of 2015-12-18) it is available at the official sourceforge pages of the Simulated Railroad Framework project and of the Simple Multiuser Online Scenes project (see the links at the https://lc-soc-lc.at/host/DownloadArea.htm to get connected).

3 Contents of the SrrTrains Releases (w/o SMUOS/DIGITS)

So first we'll try to give an overview, what is contained in which .zip file.

Component	Status	.zip file			License
		SRR	TP	TLS	
SMUOS Framework	v0.01 pre-alpha	☑	☑	☑	LGPL
Example Basic MIDAS Objects	v0.01 pre-alpha	☑	☑	☑	LGPL
Beamer Manager Extension	v0.01 pre-alpha	☑	☑	☑	LGPL
Key Manager Extension	v0.01 pre-alpha	☑	☑	☑	LGPL
Train Manager Extension	v0.01 pre-alpha	☑	☑	☑	LGPL
Example Track Geometry and Models	v0.01 pre-alpha	☑	☑	☑	LGPL
Docu: Concepts' Descriptions	v0.01 pre-alpha	☑	☑	☑	LGPL
Docu: Product Database (SRR)	V0.01 pre-alpha	☑	☑	☑	undef
Example Vehicle Models	not impl. yet		☑	☑	undef
Demo Layout	v0.01 pre-alpha		☑		undef
Docu: Testcase Descriptions	v0.01 pre-alpha		☑		undef
SRR Test Frame	v0.05.1			☑	sharew
Modified Demo Layout	v0.01 pre-alpha			☑	undef
Empty Frame	v0.01 pre-alpha			☑	undef
SrrTrainsExamples	v0.01 pre-alpha			☑	undef
Docu: Product Database (TLS)	v0.01 pre-alpha			☑	undef
Docu: Project WEB http://simulrr.sourceforge.net http://smuos.sourceforge.net	alw. up to date	Online resources			undef
Docu: SrrTrains' Homepage https://lc-soc-lc.at/host	alw. up to date				undef

Table 1: Software Items, that are contained in the three .zip files SRR, TP and TLS

The components in the blue rows are described in product database "SRR", the components in the green rows are described in the product database "TLS".

The components are assembled to following "releases" (.zip files):

SRR...............the SRR/SMUOS Framework.............SRR_0033.09bf4.zip
TP..................the Test Package.................................TestPackage_0033.09bf4.zip
TLS...............the SRR Tools.....................................This inofficial release does not contain TLS

So we see, the "release" **SRR** is the core SRR/SMUOS Framework, that can be used within your own SrrTrains scene. It is developed in open source projects on the sourceforge and it is licensed by an LGPL.

If you like to support us with testing the SRR/SMUOS Framework – or if you like to get a first impression of the functionality – then you can download the "release" **TP** (which contains a demo layout additionally).

The "releases" **SRR** and **TP** support following Web3D Browsers:

- BS Contact in single-player-mode
- BS Contact in multi-player-mode (with BS Collaborate)

Last but not least, if you really want to start to model SrrTrains models, modules and frames, then the use of the **SRR Tools (TLS)** could be helpful.

Currently the SRR Tools consist mainly of the SRR Test Frame, which is a nice little GUI (written in Visual Basic) to test your models and modules in a BS Contact/.Net/Windows environment.

4 Summary

When first dealing with 3D graphics, it was no long way to get in contact with VRML/X3D.

Additionally, I had in mind a nice little use case, which I called the "DIY-Virtual-Multiplayer-Model-Railroad".

Quite a lot of people should be convinced to take advantage of a growing biotope of hobby enthusiasts and of professional/commercial service providers.

Though I am still convinced VRML/X3D being the right choice for such a project, the initial ignition of the biotope has not happened yet (or it has happened somewhere else or under different circumstances than I am aiming at).

So I am drumming again and telling everybody what a nice hobby this is, hoping for more feed back and support.

4.1 The Idea of the SRR/SMUOS Framework

When doing first trials with the network sensor, and when having a look to its interface, then it is understandable that this very general interface cannot be used directly by modellers (as long as they are not programmers additionally).

Something in between modeller and network sensor must exist that specializes the services of the network sensor to some easy-to-use interface.

A few additional prototype nodes should be defined to provide something that is as easy to use as the DIS component, but more general (imho, the DIS component is too specific, on the other hand).

So, when a railway enthusiast/modeller will build his model railroad by using VRML/X3D + SRR/SMUOS Framework, then this fact alone will guarantee that models, modules and SrrTrains layouts will be compatible to each other.

An SrrTrains rail vehicle would be usable on each and every SrrTrains layout, as long as the layout used modules, which used the SRR/SMUOS Framework in turn.

In another kind of interpretation, this would mean: the SRR/SMUOS Framework should close gaps, which cannot be closed (currently) by the VRML/X3D standards and hence it can be interpreted as an attempt to support the evolution of VRML/X3D.

4.2 How Much of the SRR/SMUOS Framework is Available Currently?

Well, the idea of the SRR/SMUOS Framework is to build model railroads.

However, currently, it is rather a rudimentary framework for multiuser chat worlds with shared events/shared states.

Following features are **supported since step 0033.04 or earlier (4),**
supported since step 0033.06 (6), supported since step 0033.07 (7), supported since step 0033.08 (8) or envisioned for some future (env)

Features written in **bold** letters are envisioned for the next step (aka "LAN Party #3").

- Architectural Features
    - Basic Initialization (7) – necessary for the use on Web Spaces
    - Tracer (4)
    - MMF Architecture[1] (4)
    - Static Modules[2] (4)
    - Dynamic Modules[3] (4)
    - SRR/MIDAS Objects (4)
    - Console Interface (4)
    - Static/Intrinsic Models[4] (4)
    - Modularity of the SMUOS Framework (6)
    - **Dynamic Models[5] (env)**
    - Handover (env)
    - Moving Modules (env)
- Convenience Features
    - Web3D Browser Support (see "Current Interworking Matrix" on project WEB)
    - MU System Support (see "Current Interworking Matrix" on project WEB)
    - Authoring Support (env)
    - Docu: Project WEB (4)
    - Docu: Concepts' Descriptions (4)
    - Docu: Testcase Descriptions (4)
    - Docu: Product Database (4)
    - Monolithic Layout (.zip) (4)
    - Monolithic Layout (Webspace) (7)
    - Distributed Layout (env)

1 An SrrTrains layout consists of **m**odels, **m**odules and of a **f**rame (MMF).
2 Static Modules are loaded and initialized at the startup of the scene instance.
3 Dynamic Modules can be loaded and unloaded on demand, during the lifetime of the scene instance.
4 Static/Intrinsic Models are loaded and initialized together with their parent module.
5 Dynamic models can be loaded/unloaded on demand, during the lifetime of the scene instance. They can change their parent module (handover).

- Features of the SMUOS Framework
  - SSC Parameters of the UOC "Base" (8)
  - Enter Moving Model (4)
  - Join Avatar (4)
  - Binary Switch (4)
  - Carousel Drive (4)
  - Trigger (8)
- Beamer Manager Extension[6]
  - Define Meeting Point (4)
  - Beam to Meeting Point (4)
- Key Manager Extension[7]
  - SSC Parameters of the UOC "Keys"(8)
  - Create/Take/Put/Reset Keys (4)
  - Carried Keys Lock (4)
  - Contained Key Lock (8)
- Train Manager Extension
  - Tracks and Turnouts (6)
  - **Rail Vehicles (env)**
  - **Create Vehicle (env)**
  - **Move Train (env)**
  - Switch the Points Manually (6)
  - **Vehicle's Basic User Interface (env)**
  - **Delete Train (env)**
  - **Vehicle's User Interface (env)**
  - Gauge Check (env)
  - Handling of Derailment (env)
  - Train Changes Module (env)
  - Bumpers (env)
  - Coupling and Collisions (env)
  - Decoupling Track (env)
  - Train Movers (env)
  - Bursting Open the Points (env)
  - Derail in Curves (Speeding) (env)
  - Derail on Points (env)
  - Lock the Points (env)
  - Interlocking (1900's) (env)

6 The Beamer Manager extension was separated from the base module in step 0033.05.
7 The Key Manager extension was separated from the base module in step 0033.05.

4.3 How Much of the SRR Tools is Available Currently?

The features of the SRR Tools are not documented on the Web, currently.

5 The Architecture of an SrrTrains Layout

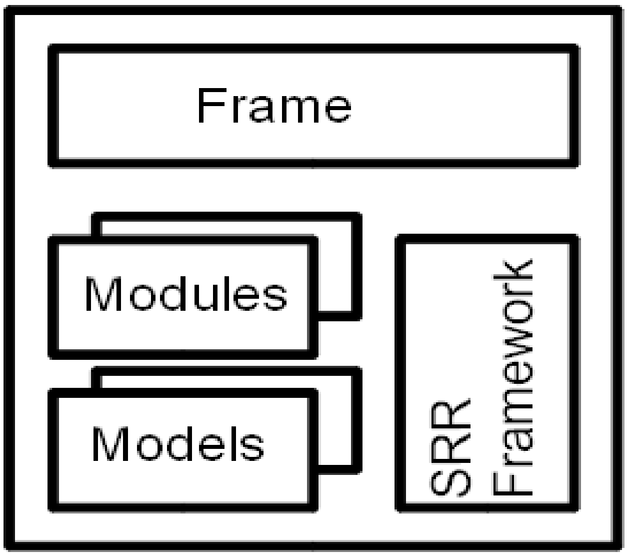
When defining the architecture of an SrrTrains layout, we just follow the architecture of a "real" model railroad.

That means:

The layout ist built by **modules** that are plugged together.

Models are used on the modules to simulate railway operation.

Some general means must exist to support the operation of the layout (in a real model railroad this is e.g. the power supply, in SrrTrains we call it just "**frame**" to indicate it is something "surrounding" the models and modules).



Drawing 1: Architecture of an SrrTrains layout

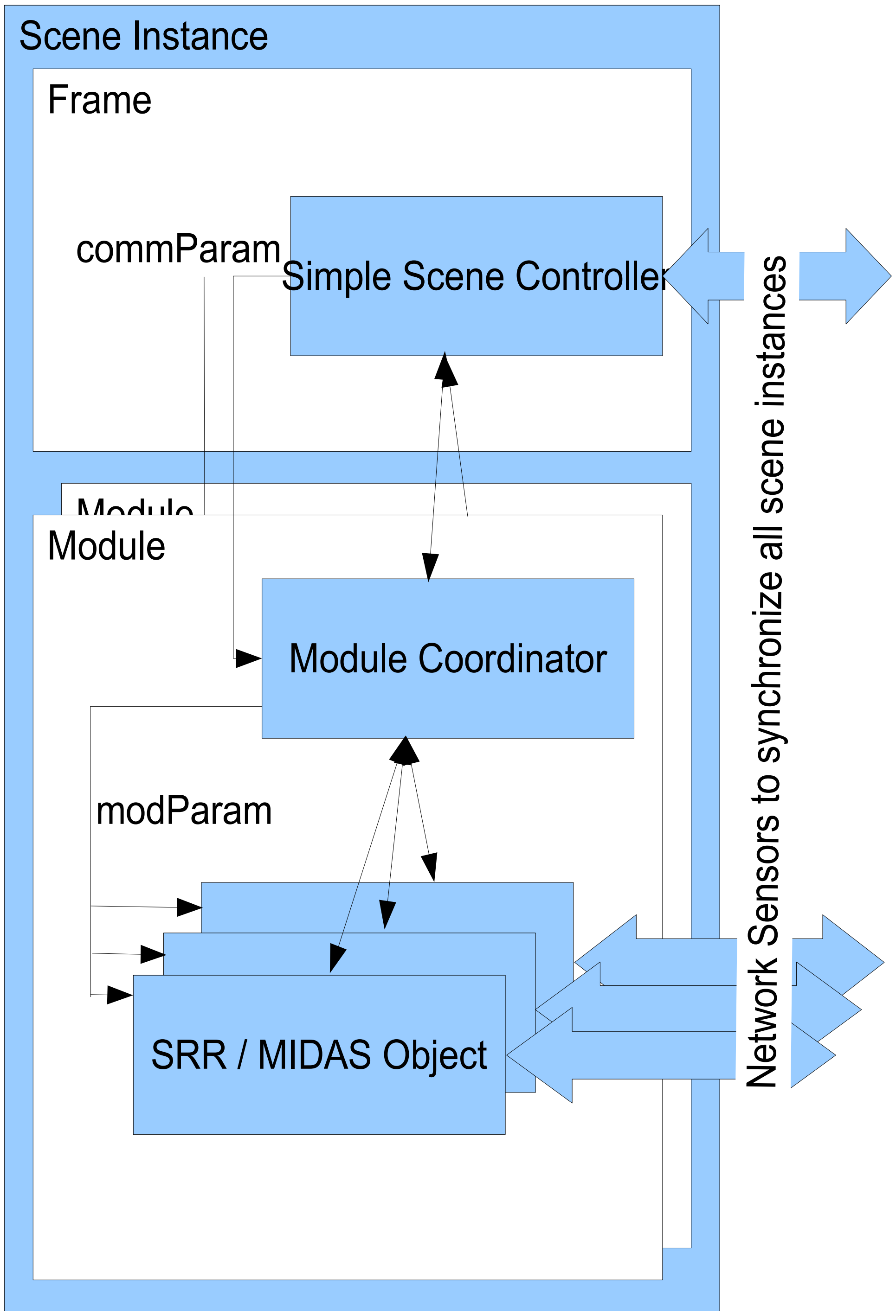
Now the SRR/SMUOS Framework can be seen as "glue" between models, modules and frames, thus it is clear that at least one X3D prototype must be provided for each of those parts of the scene.

So we have following rough relation between parts of the scene and parts of the SRR/SMUOS Framework.

Part of the Scene	Associated Part of the SRR Framework
Models	SRR Objects / MIDAS Objects
Module	Module Coordinator
Frame	Simple Scene Controller

Table 2: Relationship between parts of the scene and parts of the SRR/SMUOS Framework

During initialization, the Simple Scene Controller sends an SFNode reference (the "common parameters" commParam) to all module coordinators, who in turn send their "module parameters" modParam (SFNode references) to the SRR Objects / MIDAS Objects. Hence each SRR Object / MIDAS Object is attached to a module and can initialize (a) network sensor(s) with the stream name(s) "Sms-<moduleName>-<objId>[-<suffix>]", thus guaranteeing uniqueness of all stream names.



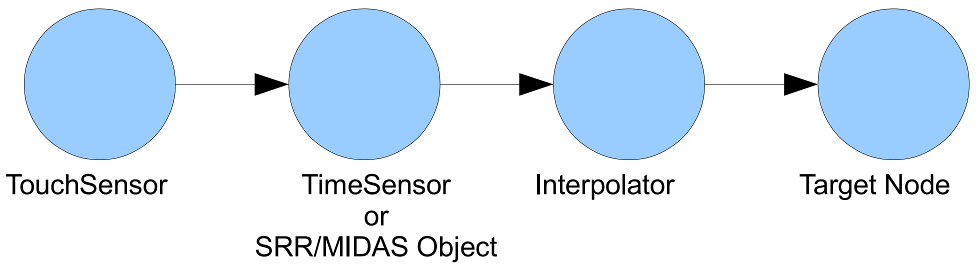
Drawing 2: Initialization of the SRR/SMUOS Framework

After initialization, each SRR/MIDAS Object can communicate with its module coordinator, and each module coordinator can communicate with the Simple Scene Controller.

5.1 What is an SRR/MIDAS Object?

When implementing animation and interactivity with VRML/X3D, we use sensor nodes. The most famous example is probably the time sensor, which can be used for virtually any animation task.

A typical configuration of the event routes for an animated part of a scene looks like:



Drawing 3: VRML/X3D Animation and Interactivity Paradigm

Now imagine the time sensor would contain (a) network sensor(s) for synchronizing several scene instances and imagine additionally that we would have an extendable set of "time sensors" that would not be as general as the original time sensor, but more specialized to concrete purposes.

Then you get to the concept of SRR/MIDAS Objects. Just replace the time sensor by an SRR/MIDAS Object in the above drawing and that's it.

Currently the SRR/SMUOS Framework is accompanied by a set of example SRR/MIDAS Objects:

- Binary Switch (e.g. to open/close doors, to switch lights on/off, and so on)
- Key Container (anything that contains keys, e.g. a key hanger)
- Carried Keys Lock (an object that can be contained in a binary switch to lock/unlock the switch, depending on keys that are carried by avatars)
- Carousel Drive (anything that can be modelled by a periodic motion that can be switched on and off)
- Beamer Destination/Beamer (two MIDAS Objects that allow to bind viewpoints of other modules, selected by a list of descriptions)
- Track
- Turnout
- and some others

Some additional SRR Objects would be necessary to support rail vehicles, e.g.:

- Axle
- RailVehicle
- Cab
- and so on

5.2 Why Do We Need a Module Coordinator?

Well, we saw that each module needs a module name, which has to be used in the stream names of the network sensors to keep them unique.

So we need an entity that maintains this module name.

Furthermore all SRR/MIDAS Objects of a module need some means to communicate with the Simple Scene Controller.

Last but not least: imagine a big model railroad layout with many modules. It could easily happen that some of the modules must be unloaded or deactivated temporarily to save CPU and/or memory resources.

All these are tasks of the module coordinator.

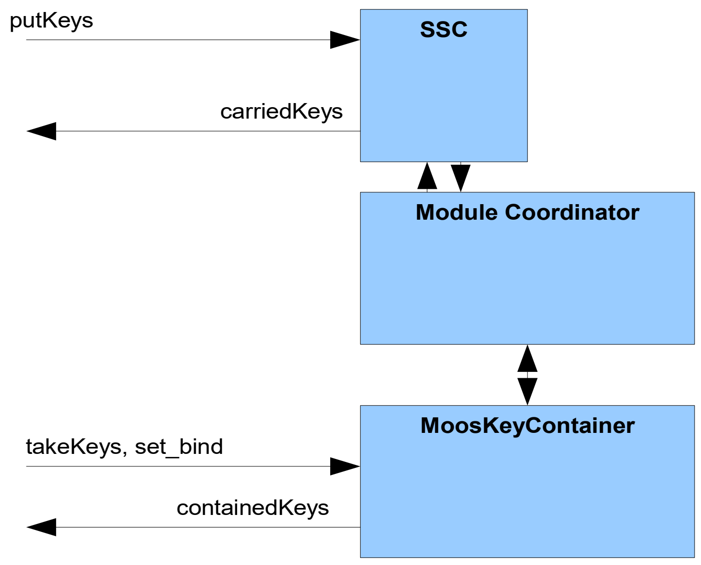
5.3 What is the Purpose of the Simple Scene Controller

The Simple Scene Controller is the **central access point of the SRR/SMUOS Framework**, where the frame can input data and listen to output.

An example should make this transparent:

Imagine a scene, where keys are located in different key containers (e.g. key hangers) in different modules.

This sounds simple, but how to "take" a key by an avatar or how to "put a key back" to a key container? The following drawing should explain the principle:



Drawing 4: Taking and Putting Keys with Key Containers

Initially, some keys are contained in the key container. This is indicated by the MIDAS Object "MoosKeyContainer" at the field "containedKeys". Probably the model author (of the "key hanger" model) has routed this MFString output to some text elements in the graphical representation of the key hanger.

Now, the model of the key hanger contains touch sensors that route MFString events to the "takeKeys" field of the MIDAS Object.

The MIDAS Object ensures that only one of the avatars can take the key and in the scene instance of the avatar, who took the key, it tells the Simple Scene Controller to add the taken key to the "carried keys".

It updates the "containedKeys" field (in all scene instances).

The frame receives the new list of "carried keys" and updates the presentation of the list of carried keys for its user.

Now the user clicks – in the frame – on one of the keys and wants to put it to a key container.

Now it get's a little bit tricky. To which of the many key containers of the scene should the key be put?

Answer: only one of the key containers can be the "bound" key container. The model of the key hanger has decided to "bind" its key container (e.g. by a proxi sensor, when the avatar approached the key hanger, or by an additional touch sensor), so the Simple Scene Controller "knows", to which key container the key has to be put.