This text is a service of http://simulrr.wordpress.com/berichte

SrrTrains v0.01

Notes about the **Pre-Alpha Release,**

which is currently available at
http://simulrr.wordpress.com/download-area#release

Christoph Valentin, July 2011

1 Tables of Contents

Table of Contents

Index of Drawings

Table Index

2 References

[1] X3D Specifications
http://web3d.org/x3d/specifications/

[2] The Network Sensor Proposal
http://web3d.org/x3d/workgroups/x3d-networking/

[3] Description of BS Collaborate (an example network sensor implementation)
http://bitmanagement.de/download/BS_Collaborate/BS_Collaborate_documentation.pdf

[4] The Simulated Railroad Framework Project
http://sourceforge.net/projects/simulrr

[5] SrrTrains's Blog
http://simulrr.wordpress.com/simul-rr

3 Contents of the Releases

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

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

Item	Status	Release			License
		SRR	TP	TLS	
SRR Framework	v0.01 pre-alpha	√	√	√	LGPL
Example Basic SRR Objects	v0.01 pre-alpha	√	√	√	LGPL
Example Extension Module + SRR Objects	not impl. yet	-	-	-	LGPL
Documentation – Concepts' Descriptions	v0.01 pre-alpha	√	√	√	LGPL
Documentation – Project Data Base (SRR)	v0.01 pre-alpha	√	√	√	undef
Official Demo Layout	v0.01 pre-alpha		√		undef
Documentation – Testcase Descriptions	v0.01 pre-alpha		√		undef
SRR Test Frame	v0.01 pre-alpha			√	sharew
Modified Demo Layout	v0.01 pre-alpha			√	undef
Empty Frame	v0.01 pre-alpha			√	undef
SrrTrainsExamples (just taken from Demo Layout)	v0.01 pre-alpha			√	undef
Documentation – Project Data Base (TLS)	v0.01 pre-alpha			√	undef
Documentation – Concepts' Descriptions (online) http://members.chello.at/christoph.valentin/concepts/000_Synopsis.pdf	alw. up to date				LGPL
Documentation – Project WIKI http://sourceforge.net/apps/mediawiki/simulrr/	alw. up to date				??
SrrTrains' Blog Pages http://simulrr.wordpress.com/simul-rr	alw. up to date				??

Table 1: Software Items, which are contained in the three releases SRR, TP and TLS

SRR.....the SRR Framework.....Archive: SRR_0033.04bf13.zip
TP.....the Test Package.....Archive: TestPackage_0033.04bf13.zip
TLS.....the SRR Tools.....Archive: SrrTools_0.03.8(0033.04bf13).zip

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

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

The SRR and the TP releases support following X3D players:

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

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)** will 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 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 again drumming and telling everybody what a nice hobby this is, hoping for more feed back and support.

4.1 The Idea of the SRR Framework

When doing first trials with the network sensor (please refer to [2] and [3] for more information about 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 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 Framework in turn.

In another kind of interpretation, this would mean: the SRR 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 Framework is Available Currently?

Well, the idea of the SRR 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 currently**

- X3D Player Support:
  - BS Contact Single-Player-Mode
  - BS Contact Multi-Player-Mode (BS Collaborate)
  - Instant Player Single-Player-Mode
- SRR: Support of "Model/Module/Frame Architecture"
- SRR: Support of static[1] modules
- SRR: Support of dynamic[2] modules
- SRR: Support of SRR objects for models
- SRR: Support of static/intrinsic models[3]
- SRR: Tracer
- SRR: Support of console interface
- SRR: join avatar/beam to remote user
- SRR Objects: Binary switching
- SRR Objects: Storing keys in a key container
- SRR Objects: Taking keys
- SRR Objects: Putting keys
- SRR Objects: Locking switches
- SRR Objects: Carousel drive
- SRR Objects: Support entering moving models[4]
- SRR Objects: Defining meeting places
- SRR Objects: Beam to meeting place
- Demo Layout: A frame with main files for all supported browsers
- Demo Layout: Static modules "Hill", "City" and "Mountains"
- Demo Layout: Dynamic module "Dunes"
- Demo Layout: Intrinsic/static models: "Fun Fair" and "Beach"
- Demo Layout: Chat
- Demo Layout: Testing all features of SRR Framework and Example SRR Objects
- Documentation: Project WIKI (description, interworking matrix, project follow up)
- Documentation: Feature Tracker: is used to organize the work, no docu
- Documentation: Concepts' Descriptions: .txt files in the release
- Documentation: Testcase Descriptions: .txt files in the release
- Deployment: Monolithic layout - .zip file

1 A static module is a module, which is loaded and initialized immediately after the scene has been loaded
2 A dynamic module is a module, which can be loaded/initialized and unloaded on demand during the runtime
3 Static models and intrinsic models are loaded and initialized immediately after the parent module has been loaded
4 When an avatar has entered a moving model, the position/orientation should be transmitted in relative coordinates to avoid a "bouncing avatars" effect. This is achieved by the use of "avatar containers"

Following features are **intended for some future**

- Authoring Support: Python Scripts for Blender
- SRR: Modularity of the SRR Framework
- SRR: Support of dynamic models
- SRR Objects: Triggering
- SRR Objects: Locking points and cabs
- SRR Objects: Support of track and turnout models
- SRR Objects: Support of rail vehicle models
- SRR Objects: Tracks and turnouts: switching the points manually
- SRR Objects: Tracks and turnouts: moving a train
- SRR Objects: Tracks and turnouts: creating a vehicle
- SRR Objects: Tracks and turnouts: Basic Interlocking (1900's)
- SRR Objects: Rail vehicles: basic user interface (doze, vConst)
- SRR Objects: Rail vehicles: user interface (cabs)
- SRR Objects: Rail vehicles: deleting a train
- SRR Objects: Rail vehicles: derailment = explosion
- SRR Objects: Rail vehicles: gauge check --> derailment
- SRR Objects: Rail vehicles: speeding in curves --> derailment
- SRR Objects: Rail vehicles: derailment on points
- SRR Objects: Rail vehicles: bursting open the points
- SRR Objects: Rail vehicles: coupling and collisions
- SRR Objects: Rail vehicles: decoupling track
- SRR Objects: Rail vehicles: handover
- Demo Layout: Dynamic modules "PlainsA", "PlainsB" and "PlainsZ"
- Demo Layout: Intrinsic/static models: "Houses" and "Tracks and Turnouts"
- Demo Layout: Dynamic models: "The Rocket", "Rail Car", "Wagon"and "Coach"
- Deployment: Monolithic layout - webspace
- Deployment: Distributed Layout

4.3 How Much of the SRR Tools is Available Currently?

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

5 The Architecture of an SrrTrains Layout

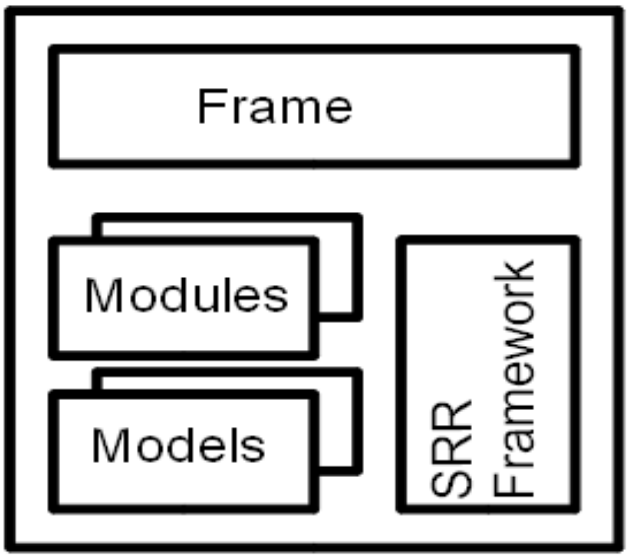
When defining the architecture of an SrrTrains layout, we just follow the architecture of a "real" model railroad.

That means:

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

Models are used on the modules to simulate railway operations.

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 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.

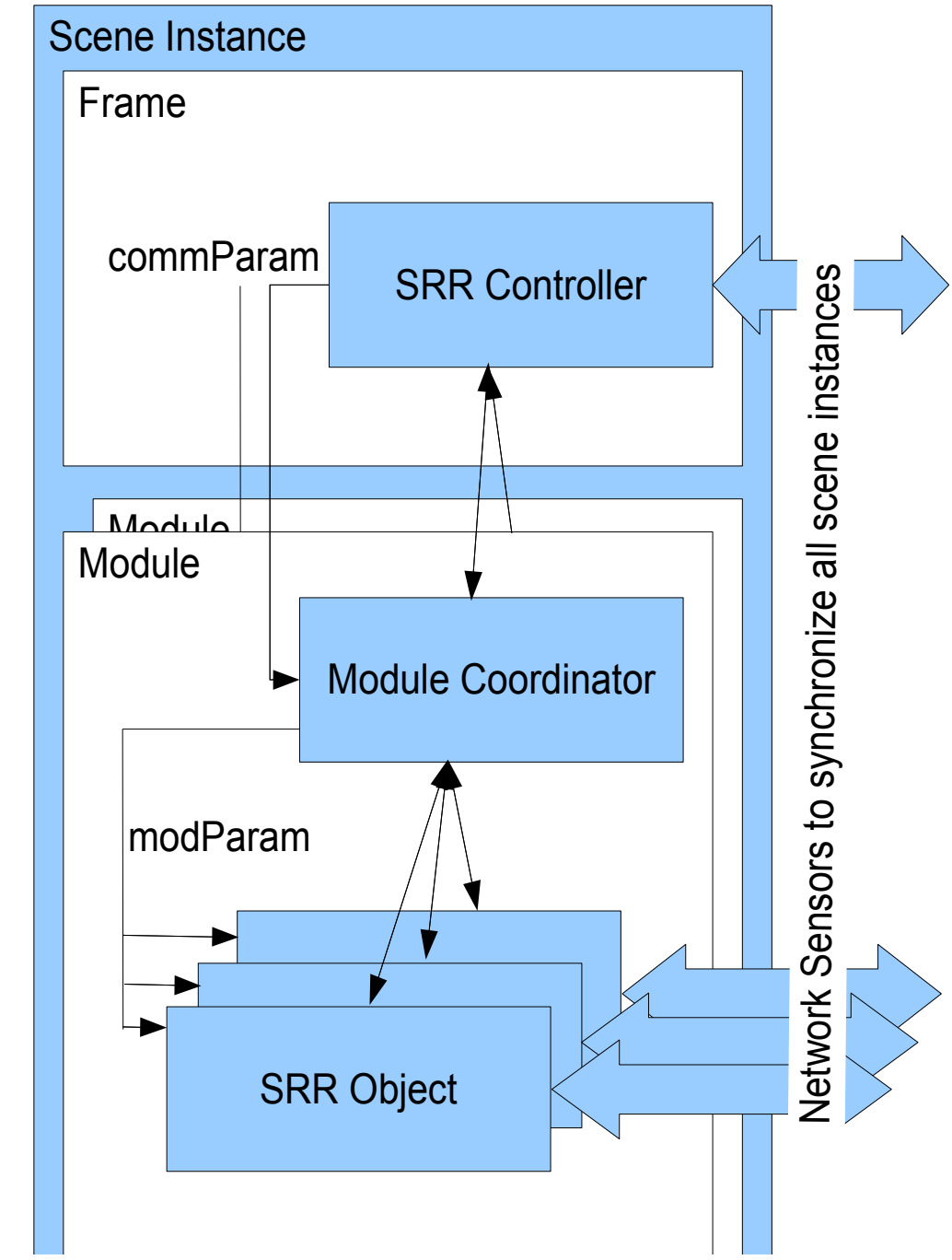
So we have following rough relationship between parts of the scene and parts of the SRR Framework.

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

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

During initialization, the SRR 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. Hence each SRR Object is attached to a module and can initialize (a) network sensor(s) with the stream name(s) "Srr-<moduleName>-<objId>[-<suffix>]", thus guaranteeing uniqueness of all stream names.



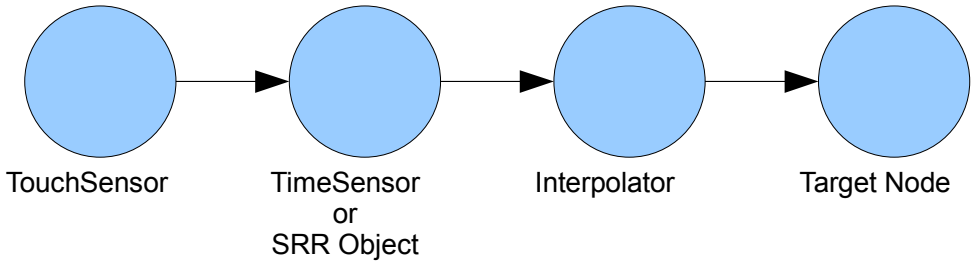
Drawing 2: Initialization of the SRR Framework

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

5.1 What is an SRR 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 Objects. Just replace the time sensor by an SRR Object in the above drawing and that's it.

Currently the SRR Framework is accompanied by a set of example basic SRR 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, which are carried by avatars)
- Carousel Drive (anything that can be modelled by a periodic motion, which can be switched on and off)
- Beamer Destination/Beamer (two SRR Objects that allow to bind viewpoints of other modules, selected by a list of descriptions)

In a few months (when the "Train Manager Extension Module" will have been implemented), some additional SRR Objects will be available, e.g.

- Track
- Turnout
- 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 Objects of a module need some means to communicate with the SRR 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.

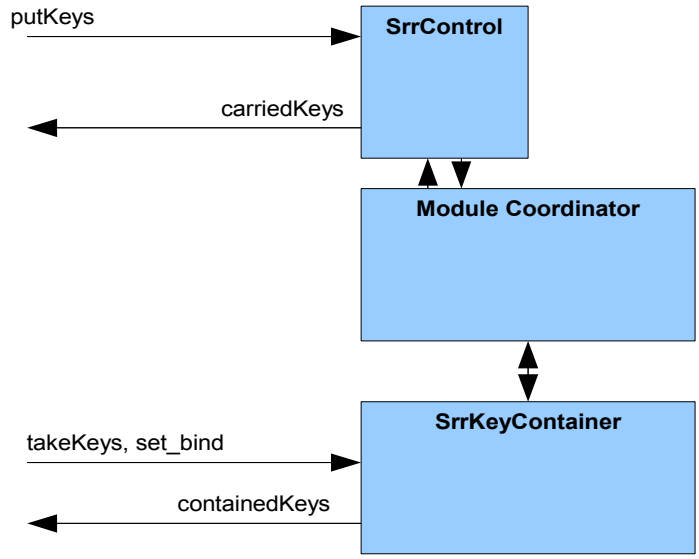
5.3 What is the Purpose of the SRR Controller

The SRR Controller is the **central access point of the SRR 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 SRR Object "SrrKeyContainer" 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 SRR Object.

The SRR 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 SRR 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 SRR Controller "knows", to which key container the key has to be put.