SrrTrains v0.01

Notes about the BIMPF (Browser Independent Multi Player Framework) – III

Christoph Valentin, March 2011

1 Table of Contents, Drawings, Tables

Table of Contents

1 Table of Contents, Drawings, Tables....1
2 References....2
3 Summary....2
3.1 Models....4
3.2 Modules....4
3.3 Frame....5
3.4 Pros and Cons of Assigning Dedicated Controller Roles....5
4 The Central Controller Role....5
5 The MOC Role (Module Controller)....5
6 The OBCO Role (Object Controller)....6
7 Target Configurations....7
8 Kinds of SRR Objects....8
8.1 "No State" SRR Objects (SrrObjectBaseNoState)....8
8.1.1 SrrBeamer and SrrBeamerDestination....8
8.1.2 SrrLockA (Carried Keys Lock)....9
8.2 "Standard" SRR Objects (SrrObjectBase)....9
8.2.1 SrrSwitchA (Binary Switch)....9
8.2.2 SrrKeyContainer (Key Container)....9
8.3 "Animated" SRR Objects (SrrObjectBaseAnim)....9
8.3.1 SrrDriveA (Carousel Drive)....9

Illustration Index

Drawing 1: Example: controller role for SRR Controller ("central controller")....3
Drawing 2: Example: a snapshot of a module activity matrix....4
Drawing 3: Example: Moving a MOC role from one scene instance to another....6
Drawing 4: Example for centralised and decentralised controller roles....8

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] Notes about the BIMPF – I – The Central Controller Role
http://simulrr.files.wordpress.com/2010/09/notesaboutbimpf-i.pdf
available at http://simulrr.wordpress.com/berichte/ (page is in German language)

3 Summary

SrrTrains defines some kind of organization for the parts of the scene graph of an SrrTrains layout (modules, models)[1]:

- A ***scene instance*** contains one and only one instance of the ***frame*** (the frame is "the surroundings of the scene", i.e. avatar handling, chat, the GUI for the users etc.)
- Each instance of the frame contains one instance of each ***loaded module*** (a module is a part of the landscape and establishes a local coordinate system)
- Each loaded module instance contains instances of ***models*** (models can be incorporated into a module – i.e. intrinisc model –, they can be referenced by the module statically – e.g. in a <ProtoInstance> or <Inline> node) or they can be loaded dynamically at runtime)

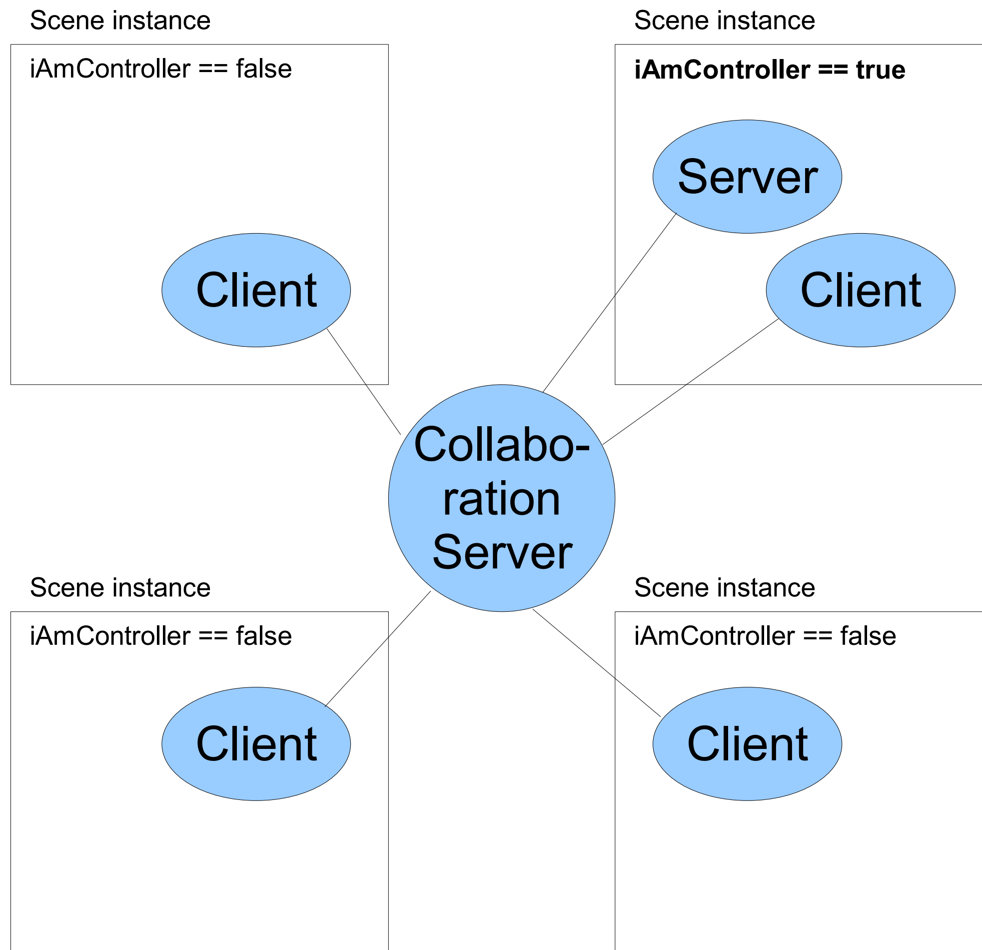
Parts of the scene can have their local state. Now, it's necessary to synchronize the local states of the scene parts among all scene instances. SrrTrains uses the services of ***network sensors*** for this purpose.

For each state, one of the scene instances is responsible to keep the state up to date and to distribute the value of this – global – state to all scene instances (keeping the local states up to date). It's said, that the ***server software*** of this part of the scene is active in this scene instance / this scene instance has the ***controller role*** for this part of the scene.[2]

This situation – one scene instance having the controller role – is depicted in Drawing 1 for the example of the SRR Controller (that is a part of the SRR Framework, which is used by the frame).

1 This organization of the scene is due to the fact, that the network sensors are distributed within the scene and each network sensor needs a "streamName" attribute (SFString), which identifies the network traffic of this network sensor. Basically, the SRR Framework defines the "streamName" = **Srr-<moduleName>-<objId>** for each network sensor of the SRR Framework.

2 Strictly spoken, the controller role is only necessary for states, that have to be calculated within the scene (e.g. by a <Script> node) and set via a **set_state** field of the network sensor. States, that can be calculated on the collaboration server (using e.g. **add_**, **sub_** etc. Prefixes), need not define a controller role. The latter kind of states is not subject of the present work.



Drawing 1: Example: controller role for SRR Controller ("central controller")

It's not only the SRR Controller, which keeps a state for a part of the scene, but there exist also other parts of the SRR Framework (one for each part of the scene).

Hence an SrrTrains layout is split into three kinds of software (parts of the scene), and therefore the SRR Framework provides three kinds of X3D prototypes:

Part of the Layout	Part of the SRR Framework
Model	SRR Objects
Module	Module Coordinator
Frame	SRR Controller

This paper describes the relations among the different controller roles of the SRR Framework

Object Controller (OBCO) – controller role for each object, see chapter 6

Module Controller (MOC) – controller role for each module, see chapter 5

Central Controller – controller role for the SRR Controller, 4

Additionally, this paper describes two ***target configurations*** of an SrrTrains multiuser session (see chapter 7) and the currently supported ***kinds of SRR Objects*** (see chapter 8).

3.1 Models

Each model uses SRR Objects to instrument the interactivity and animation of the model.

SRR Objects hold a local state and they contain one or more network sensors to synchronize the local states with the global state among all scene instances.

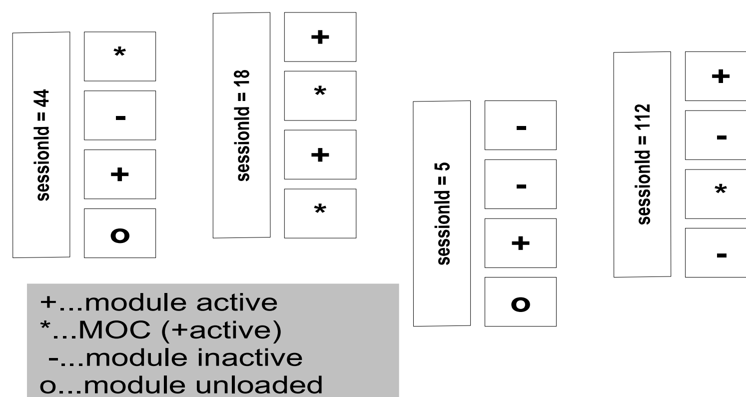
One scene instance is responsible to set the – global – state of the SRR Object and to distribute the state to all scene instances (it is said, this scene instance has the ***OBCO role*** for this object).

How is the OBCO role for an object appointed? Well, this is described in the next chapter.

3.2 Modules

The layout is split into parts of the landscape, where each part (i.e. each module) can be loaded/unloaded independently from the other parts and it establishes an own local coordinate system.

Modules can be loaded/unloaded and they can be activated/deactivated. An example of a snapshot of the ***module activity matrix*** is shown in the next figure.



Drawing 2: Example: a snapshot of a module activity matrix

Each SRR Object is attached to one module and if a module is inactive, then the SRR Objects of this module avoid calculation and output of the local state, thus saving processor power.

The module activity is a matrix (modules can be activated/deactivated in each scene instance independently), and if at least one instance of the module is active, then one of the active instances will be assigned the ***MOC role***.

The SRR Objects of a module will have their OBCO roles in that scene instance, which has the MOC role for this module.

3.3 Frame

The SRR Controller contains server software, too. The scene instance, which has the controller role for the SRR Controller, is said to be the central controller (see the paper "Notes about the BIMPF - I" [4] for more details about the central controller role).

3.4 Pros and Cons of Assigning Dedicated Controller Roles

Tbd.

4 The Central Controller Role

The SRR Controller needs a dedicated scene instance, which is responsible to calculate and distribute the "Communication State" (commState).

This dedicated scene instance is said to be the central controller, or just "controller".

Assigning the central controller role is not easy, from a distributed system perspective, if it is not impossible at all, due to the lack of network semaphores in the network sensor API.

It must be guaranteed, that never ever two or more scene instances take the central controller role concurrently, and on the other hand, the time gap Δt, where none of the scene instances has got the controller role, should be short enough to ensure service quality by retransmission of requests.

However, the current experimental implementation of the SRR Framework contains a selection mechanism, that worked sufficiently on two LAN parties (approx. 5 testers were present at each of the LAN parties).

In case of an MMORPG environment, it can be predicted, that this mechanism will fail frequently.

Please refer to [4] for more details about the central controller role.

5 The MOC Role (Module Controller)

The assignment of MOC roles is part of the module activity matrix. The module activity matrix is set centrally by the server software of the SRR Controller (i.e. by the "central controller").

The module activity matrix contains information about

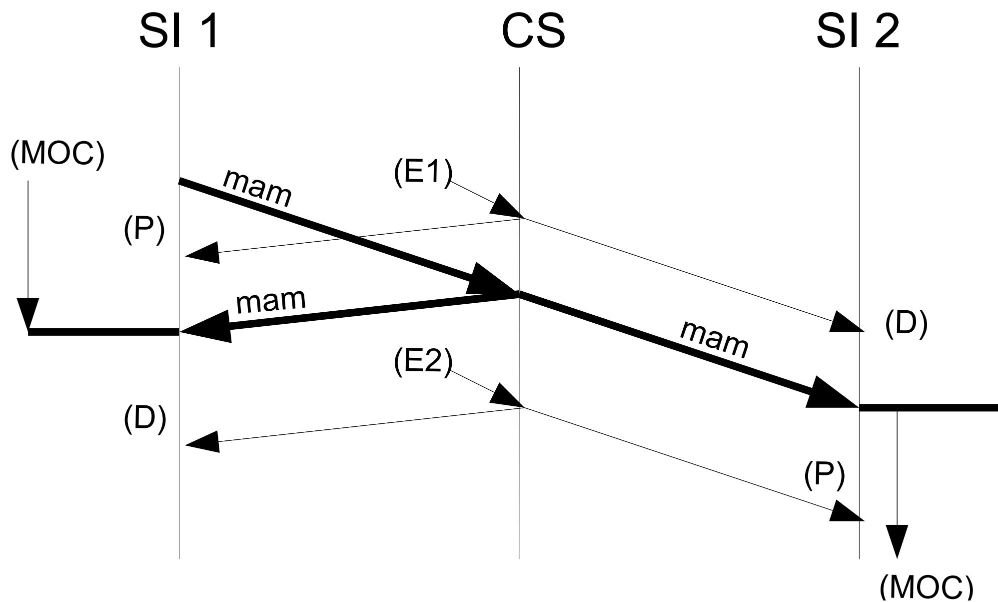
- module attachment state
- module activity state
- MOC roles

The arrangement of MOC roles within the module activity matrix can change, when

- a module is activated in a scene instance
- a module is deactivated in a scene instance
- a scene instance explicitely requests a MOC role
- a module is detached in a scene instance
- a scene instance is deleted

The module activity matrix is distributed as a part of the state of the SRR Controller, thus it is

guaranteed that never ever two scene instances have got a MOC role at the same time.



Drawing 3: Example: Moving a MOC role from one scene instance to another

CS......................collaboration server

SI 1/SI 2.............scene instance 1 / scene instance 2

SI 1....................here the central controller

mam...................module activity matrix

E1.......................event 1 (processed (P) by SI 1, discarded (D) by SI 2)

E2.......................event 2 (processed (P) by SI 2, discarded (D) by SI 1)

6 The OBCO Role (Object Controller)

Basically, the controller roles of the SRR Objects will follow the MOC roles of their parent modules.

However, to be flexible for the implementation of handover (where SRR Objects will change their being attached to one module with the being attached to another module), it was decided to define a separate controller role for SRR Objects – the OBCO role.

This allows for short time spans, where some instances of the SRR Object are still attached to the old module and other instances are already attached to the new module. In such time spans it would be indeterminate relying on the MOC roles of the parent modules.

For more information about SRR Objects, please refer to chapter 8 .

7 Target Configurations

The SrrTrains concept deals with ***small user groups***.

It is assumed, that more than one user group can be served by one collaboration server. The software of one user group need not be aware of the other user groups. This could e.g. be achieved by collaboration servers supporting ***chat rooms*** (a network sensor could never cross the borders of a chat room, even, if he uses the same streamName as a network sensor of the other chat room).

The SRR Framework solves the problem of assigning a scene instance to be responsible for the state of the scene: Each module is assigned a scene instance that has the ***MOC role*** and the SRR Controller is assigned a scene instance that has the ***central controller role***. Additionally, the ***OBCO roles*** of the SRR Objects follow the MOC roles of their parent modules.

All these controller roles can be assigned in one of two ways:

- decentralised controller roles
- centralised controller roles

Decentralised controller roles:

As long, as the user does not request anything special, the SRR Framework will autonomously assign controller roles, taking this burden from the author of the scene. In most cases, this will mean, that the first instance, which gets active, will be assigned the controller role. A subsequent change of the controller role will only be done, if there is a reason to do so (e.g.: if the MOC of a module gets inactive, another active instance of the module will be assigned the MOC role).

Centralised controller roles:

If the user wants to control the controller roles, he can implement such function in the frame.

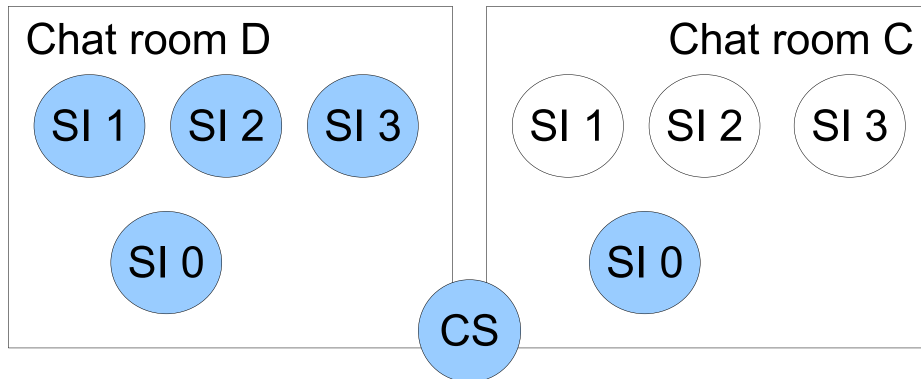
The user interface of the SRR Controller (which is used by the frame) provides fields, that can be used to

- request the central controller role for this scene instance
- request MOC roles for specified modules, in this scene instance

So, if a scene instance wants to keep all controller roles, it just has to

- request the central controller role at/after startup
- take care, that all modules are loaded and active all the time
- request the MOC role for each module, after it has been loaded and activated

Drawing 4 gives two examples, for decentralised and for centralised controller roles.



Drawing 4: Example for centralised and decentralised controller roles

8 Kinds of SRR Objects

The current version of the SRR Framework supports 3 kinds of SRR Objects:

- "no state" SRR Objects – do not maintain a global state
- "standard" SRR Objects – maintain a discrete global state, changing slowly
- "animated" SRR Objects – continuous global state, changing as a continuous function of time state(t)

The "base class" X3D prototypes, that can be used within SRR Objects, are hence called

- SrrObjectBaseNoState
- SrrObjectBase
- SrrObjectBaseAnim

8.1 "No State" SRR Objects (SrrObjectBaseNoState)

Instead of describing it theoretically, the currently available SRR Objects will be described shortly.

8.1.1 SrrBeamer and SrrBeamerDestination

Since an SrrTrains layout is split into modules and since the authors of different modules may be different, there is no way to define a "gate" interface, where the user can click on some geometry in one module to be moved (bound) to a viewpoint in another module (only the browser immanent interface may be used for this purpose).

SrrBeamerDestination takes a reference to a viewpoint node and an optional description to make this information public via the SRR Controller.

SrrBeamer outputs the collected descriptions of all beamer destinations (of one scene instance) in an MFString field and takes an SFString field "bindBeamerDestination" as input to bind the associated

viewpoint.

8.1.2 SrrLockA (Carried Keys Lock)

A Carried Keys Lock is locked by default, but it can be unlocked by the "carried keys" of the local scene instance (the SRR Controller keeps track of the carried keys of an avatar/user).

The model author has to set the field "fittingKeys" (MFString) and the state of the lock will be reported in the field "locked" (SFBool).

8.2 "Standard" SRR Objects (SrrObjectBase)

Instead of describing it theoretically, the currently available SRR Objects will be described shortly.

8.2.1 SrrSwitchA (Binary Switch)

This SRR Object keeps track of the global state "state" (SFBool). The local (client) part of the SRR Object will provide a smooth, but local, animation, based on the global change of the state.

Parameter: "duration" (of the animation, SFFloat)

Input: "toggle" (SFBool)

Output: "actualState" (SFBool), "softState" (SFFloat)

8.2.2 SrrKeyContainer (Key Container)

This SRR Object keeps track of the global state "containedKeys" (MFString).

It can be used

- to create and store keys
- to take keys (input "takeKeys" (MFString) will lead to update of "carried keys" in SRR Controller and to update of "containedKeys")
- to act as the destination for "putKeys" actions (the "putKeys" field is provided by the SRR Controller to the frame, but a key container has to be "bound" – with the "set_bind" field –, before the SRR Controller can put keys into the container)

8.3 "Animated" SRR Objects (SrrObjectBaseAnim)

Instead of describing it theoretically, the currently available SRR Objects will be described shortly.

8.3.1 SrrDriveA (Carousel Drive)

This SRR Object contains an SrrSwitchA object (to switch the carousel on/off) and maintains the global (animated) state angle(t) (value between 0.0 and 6.28).

The global state angle(t) will be calculated centrally (at the OBCO), by solving the equations:

angle" = (M(angle',p) - friction * angle') / mass

angle' = INTEG(angle")

angle = INTEG(angle')

where M(x',p) is dependent on the parameter p ("actualState" of the contained SrrSwitchA).

The OBCO will distribute targets every now and then, and the client software will use a time sensor and an interpolator to follow the targets in a piece-wise linear function angle(t).