

1 Table of Contents

Table of Contents

1 Table of Contents.....	1
2 References.....	1
3 Foreword (History).....	1
4 Summary.....	2
5 The Server Role.....	3
6 The Controller Role.....	4
7 Granting the Controller Role.....	4
8 Taking the Controller Role.....	5
8.1 Initialization of the First Scene Instance.....	6
8.2 Scene Instance with Controller Role Leaves the Game.....	7
9 Conclusions.....	7
10 Outlook.....	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 of a network sensor implementation),
http://bitmanagement.de/download/BS_Collaborate/BS_Collaborate_documentation.pdf

3 Foreword (History)

After some initial trials within the "Rollercoaster" project in 2008, a new hobby project was started in January 2009 - "SrrTrains v0.01". The goal was to implement an X3D based (see [1]) framework for simulated multiplayer railroads.

Test versions of BS Contact and BS Collaborate were available on the net, but after some first trials it was decided to broaden the approach to more browsers, sticking to the fact of X3D being browser independent per se.

First trials with the collaboration server from Octaga were promising, but it happened that newer versions of Octaga collaboration server were no more available on the Internet for free.

Hence the "SRR Framework" (the central component of the SrrTrains concept) was developed for following modes of operation:

- Single Player Mode with BS Contact
- Multi Player Mode with BS Contact and BS Collaborate
- Single Player Mode with Octaga Player

The SRR Framework underwent so-called "steps" of development, i.e "step 0001" thru "step 0032",

afterwards the "First LAN Party" was undertaken (in March 2010).

Following incidents determined the further development of the SRR Framework

- The 1st LAN Party revealed some instabilities of the SRR Framework
- The efforts for "SRR v0.01" were estimated to be more than 2 staff years (approx.)
- DeepMatrixIP was announced and released on the X3D-public mailing list.

Hence the decision was to start "step 0033", which should

- launch an open source project on the sourceforge
- re-design the SRR Framework completely
- support more browsers than step 0032 supported (e.g. plus Instant Player + DeepMatrixIP)

Step 0033 is currently ongoing.

The approach, that was originally used to distinguish "EventStreamSensor" of BS Contact/BS Collaborate and "NetworkSensor" of Octaga Player / Octaga Collaboration server, should now be generalized to be used for Instant Player / DeepMatrixIP additionally.

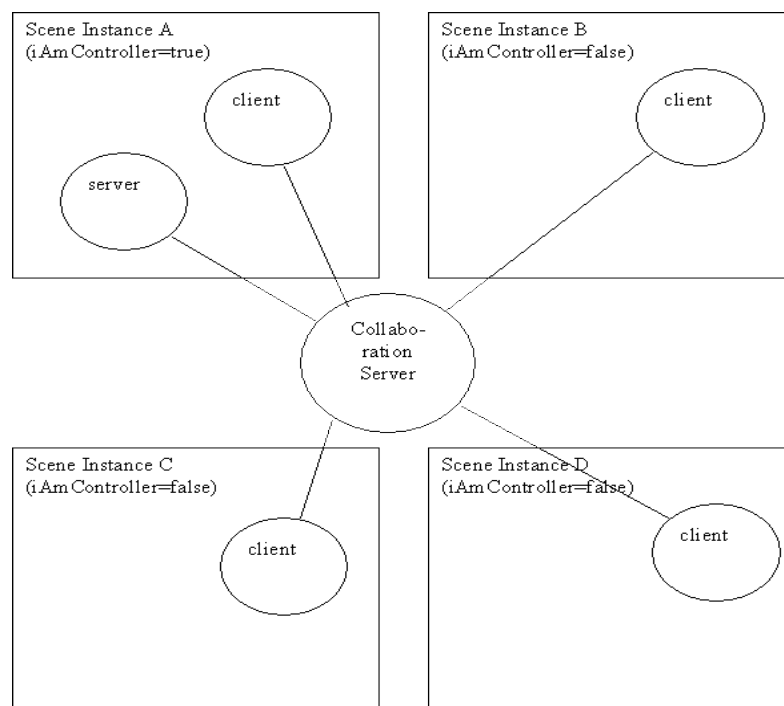
This approach was called "BIMPF"¹ (Browser Independent Multi Player Framework).

The BIMPF approach is described at the project WIKI of the sourceforge project, please refer to <http://sourceforge.net/apps/mediawiki/simulrr> .

4 Summary

It's one of the disadvantages of the BIMPF approach, that some mechanism must be implemented, that assigns a server role to a dedicated scene instance.

This paper exemplifies, how a "controller" role can be established to define one dedicated scene instance, that will be the "central controller" for the whole simulation. See following illustration of the situation:



The conclusion will be, that the services of the "central controller" (server software in the scene

¹ Bimpf is an Austrian dialect word and means something like "little child", "baby" (but not in a polite way)

instance with `iAmController = true`) should be migrated to the MU Server / Collaboration Server as soon as all supported MU systems support those services natively or as soon as all supported MU systems provide a satisfying interface to deploy self-programmed software to the MU server / Collaboration Server.

5 The Server Role

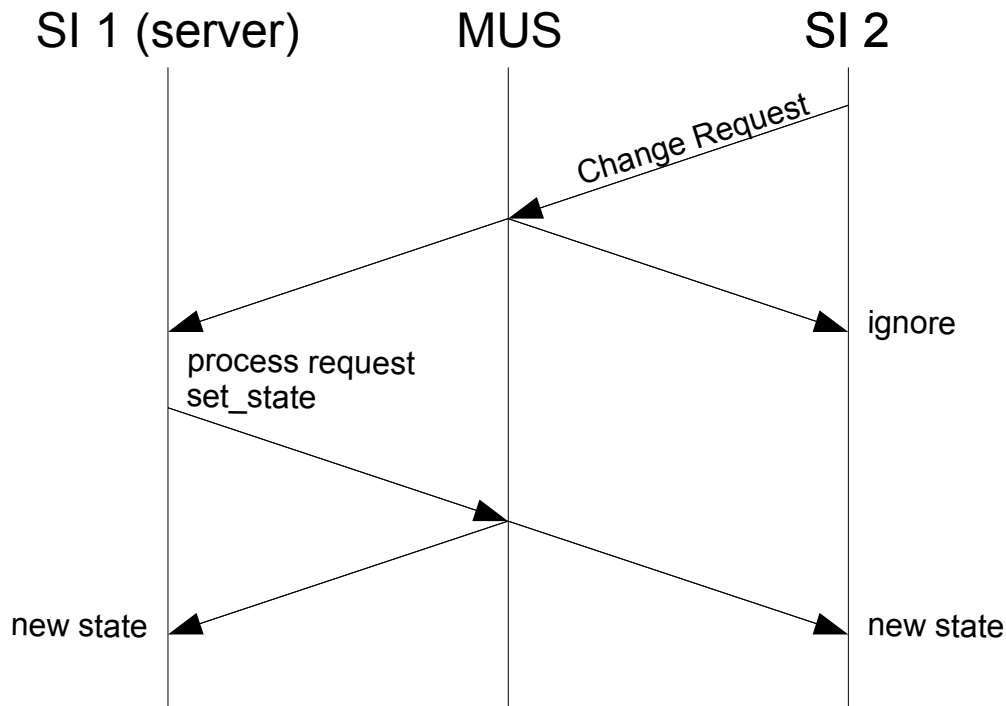
Network Sensors (see [2], [3]) distinguish "states" and "events", where events are just evenly distributed to all scene instances, but states are stored persistently on the MU Server / Collaboration Server and can be modified (`set_....`, `add_....`, etc.) by any scene instance.

As long as only simple changes are requested (i.e. changes, which can be calculated natively by the MU system), there is no need to assign a server role to a scene instance.

As soon, as complex changes of the state have to be calculated by self-programmed software, it can be of advantage to include the serving ("calculating") software into a `<Script>` node in the scene and to assign a "server role" to one of the scene instances.

The serving software in the scene instance with the "server role" will be responsible to receive change requests and to process them.

As a result, the state will be modified and distributed, see following figure:



SI 1.....Scene Instance 1

SI 2.....Scene Instance 2

MUS.....MU Server / Collaboration Server

It's clear from the above figure, that never ever two or more scene instances shall have the server role at the same time. This would cause indeterministic behaviour when modifying the state.

Small time gaps Δt without having a server are acceptable, on the other hand.

6 The Controller Role

In the last chapter we spoke about a generic "server role".

However, each and every network sensor could define its own "server role".

The concrete "server role" of the network sensor of the "SRR Controller" (a central part of the SRR Framework) is called "controller role".

We say, one scene instance is the "central controller" of the railway simulation.

The central controller is responsible for the "Communication State" (commState), a rather complex MFString value, that holds information about the communication state of the scene instances.

The commState contains an MFInt32 "sessionIds", that holds the list of all scene instances, and an SFInt32 "controllerIx", that holds the index of the scene instance with the controller role.

That means, sessionIds[controllerIx] is the sessionId of the central controller.

In short words: it's described in the state itself, which scene instance is responsible for the state.

That means, usually the controller role can be granted only by the controller itself to another scene instance. This is described in the next chapter:

- **Granting the Controller Role**

Under some circumstances, a scene instance has to take the controller role pro-actively, because no scene instance has got the controller role. This must be done carefully to avoid two or more scene instances having the controller role at the same time. These situations are described in the chapter:

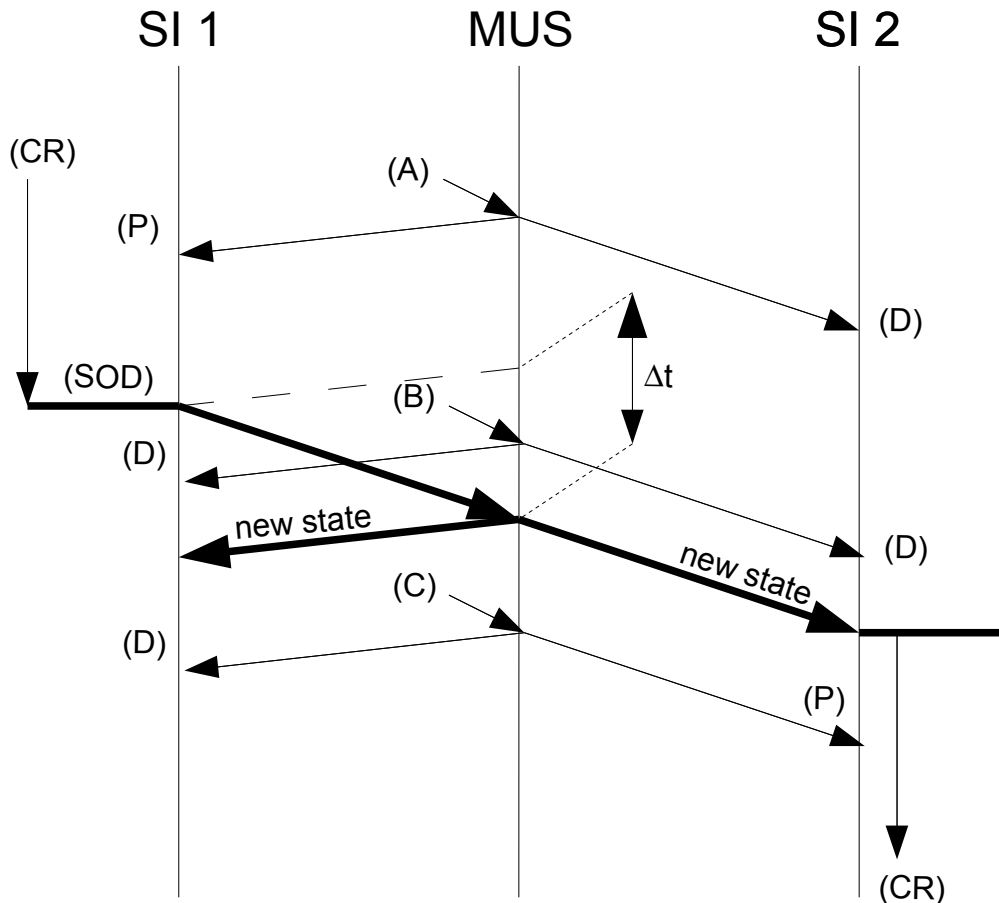
- **Taking the Controller Role**

7 Granting the Controller Role

Usually exactly one scene instance has got the controller role and each other scene instance can request to be granted the controller role with a "change request" event.

Before sending out the new commState (that identifies the new controller), the old controller will pro-actively reset the *iAmController* flag to avoid glare cases.

All scene instances will update their *iAmController* flag on reception of the commState. This standard procedure of granting the controller role is shown in following figure:



- (CR).....Controller Role
- (SOD).....Switch over decision
- ΔtTime gap without controller
- (A), (B), (C).....Three requests from any scene instance, (A) is processed by SI 1,
..... (B) is lost, (C) is processed by SI 2
- (P).....Controller processes a request
- (D).....non-controller discards a request

One sees from the figure that

- it is guaranteed, that at maximum one scene instance (SI) has the controller role at the same time
- the time gap Δt without controller derives from the round trip time between the SI and the MUS
- all this is guaranteed, if the MU system keeps the order of states and events

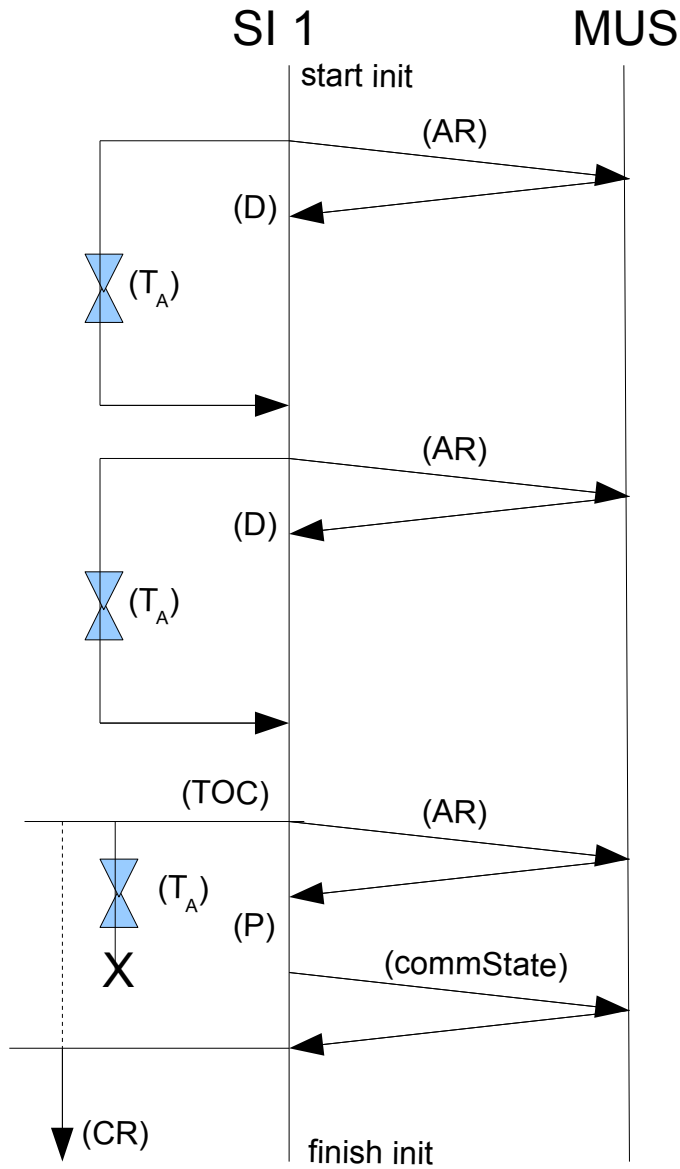
8 Taking the Controller Role

It may happen that no scene instance has the controller role at all, but one scene instance must take the controller role pro-actively. Namely, the situations are:

- the first scene instance is initialized and initializes the commState
- the scene instance with the controller role has been terminated (without being able to update the commState)

These situations can be handled by certain algorithms, using supervision timers. The time gap Δt can be confined by accurately setting the duration of the supervision timers. See following figures for more information:

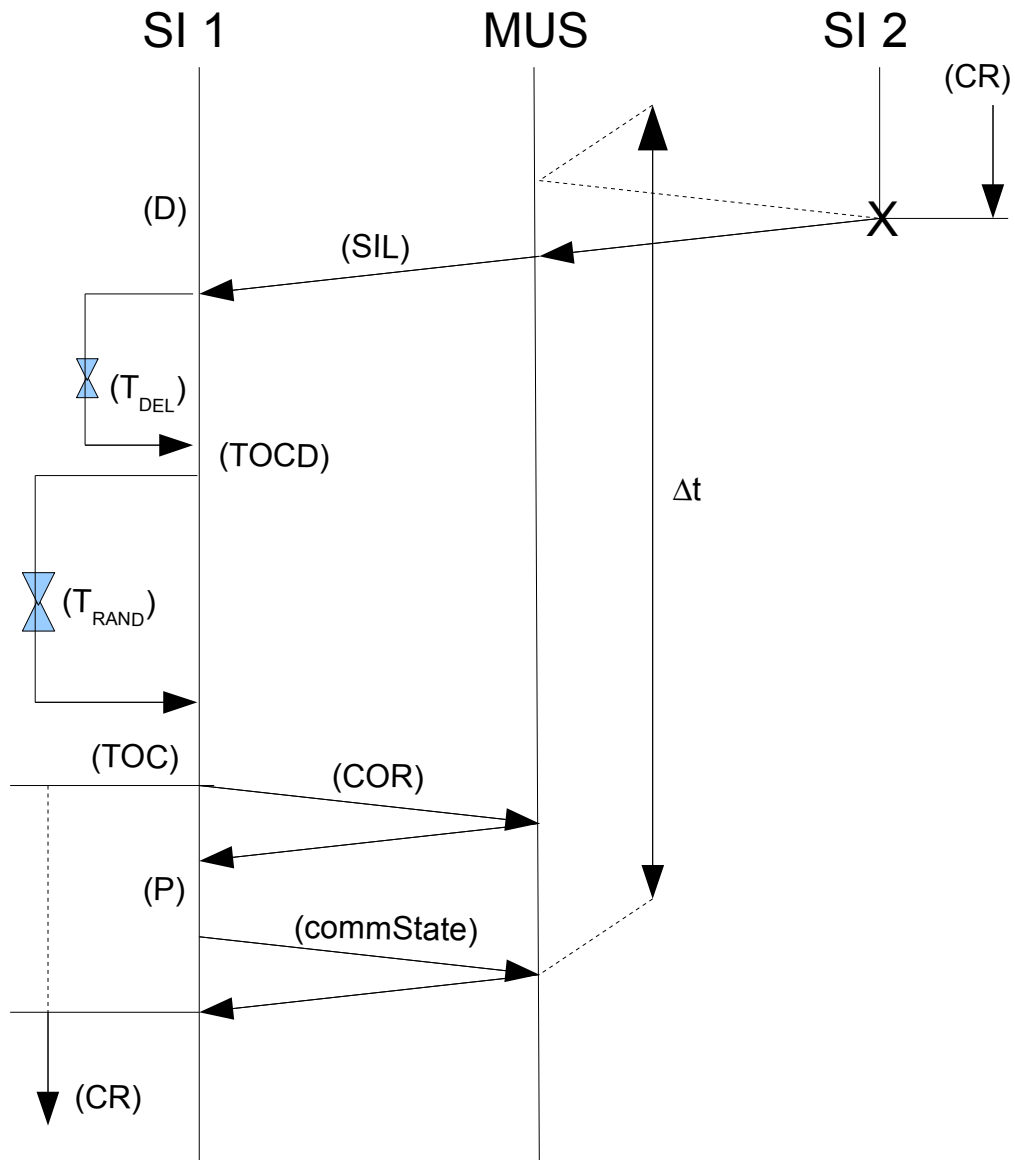
8.1 Initialization of the First Scene Instance



- (AR).....Access Request
- T_AAccess Timer
- (TOC).....Taking over Controller Role
- (commState).....Distribution of first Communication State "commState"

Δt is zero, because SI 1 starts sending change requests earliest in "finish init" phase.

8.2 Scene Instance with Controller Role Leaves the Game



T_{DEL}.....Deletion timer
T_{RAND}.....Random Timer
(TOCD).....first takeover decision
(SIL)....."Session ID Left" Message
(COR).....Controller Request

9 Conclusions

The BIMPF approach seems to be a temporary solution, that enables a scene author to implement "central services" in a <Script> node, which are not yet natively available in the supported MU systems.

It is possible to define a "server" role of one scene instance, where it is guaranteed that never ever two scene instances have got the server role simultaneously. The time gap Δt without having a

server can be identified and confined.

The most complicated case is when a scene instance having the server role leaves the session.

Probably the BIMPF approach could be optimized, when it was supported by the standard (Δt could be minimized):

- The Browser could select the scene instance which has the controller role
- Network Sensors could guarantee the possibility to send a "last message" during shutdown() of the scene. This would enable the controller to grant the controller role to a randomly selected scene instance as a "last action" when being terminated.

10 Outlook

Currently only the "Simulated Railroad Framework" (<http://sourceforge.net/projects/simulrr>) tries to use the BIMPF approach (we are still looking for volunteers to implement the multiplayer mode with Instant Player / DeepMatrixIP and the multiplayer mode with Octaga Player).

The approach could be used as temporary solution for other frameworks, too.

Final solution is envisioned to be some X3D standardized mechanism based on the network sensor, but reaching beyond the functionality of the network sensor.