SMUOS/C3P

Notes about the **Network Semaphores (Controller Roles)**

Christoph Valentin, August 2013 (updated April 2018)

Table of Contents

Index of Drawings

Table Index

1 References

[1] SMUOS/C3P *: The Idea
http://smuos.wordpress.com/2011/03/01/smuos-and-the-ietf/ (meanwhile deleted)

[2] SMUOS *: The Project
http://smuos.sourceforge.net

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

[4] The SrrTrains Project
http://letztersein.wordpress.com/srrtrains-v0-01

* SMUOS = Simple Multiuser Online Scenes

C3P = Collaborative 3D Profile

2 Summary

This hobby report deals with a possible application of the X3D Network Sensor node within a new X3D component, which is to be defined.

We rely on the experiences with the example implementation BS Collaborate [3].

Those experiences were made during conduction of the SrrTrains project [4].

3 Motivation

The SMUOS Framework [2] uses network sensors for the **following use cases**

(a) in the Simple Scene Controller Base (SSC Base)

(b) in Simple Scene Controller Extensions (SSC Extensions)

(c) in the SSC Dispatchers

(d) in MIDAS Objects

In all cases **we rely on what we call "Client Based Server Software" (CBSS)**.

That means, we do not use prefixes like "add_", "inc_" or "dec_" to perform server side calculations, but we use the prefix "set_" to apply the result of client side calculations to the states, which are stored persistently on the collaboration server (CS) afterwards.

Client side calculations have the **advantage** they can be easily and flexibly programmed, e.g. using client side JavaScript.

On the other hand, those calculations have the **disadvantage** a clear distinction must be made to define, which scene instance is responsible to perform the calculations and to update the states. Otherwise (if two or more scene instances felt "responsible" to update the states at the same time) an unpredictable behavior would be the result.

For this reason, **we defined "controller roles"**.

Each network sensor is "controlled" by one distinct scene instance. This scene instance, and only this scene instance, is allowed to perform client side calculations for this network sensor and to update states using the "set_" prefix.

Using the present mechanisms of the network sensor (events, states, ...) [3], we described the BIMPF approach to achieve our goals in an experimental way. This is examplified in the SrrTrains project [4] and in the SMUOS project [2].

We think it would be of advantage not only for us, but for a broader variety of applications, if the network sensor would support the controller roles in a native way (this is similar to the fact that semaphores should be supported by the operating system).

4 Proposal

Thus we do following proposal:

We do this proposal without having tried it.

Some experiments with real software should be done, before incorporating the proposal to the X3D standard.

1. Enhance the Network Sensor by five additional fields

outputOnly	SFBool	controller	
inputOutput	SFBool	requestController	default: TRUE
outputOnly	SFBool	controllerRequest	
inputOnly	SFBool	controllerGrant	
initializeOnly	SFString	followStreamName	default: ""

2. Define a new prefix/suffix for network sensor events, which provides for routing of events to the scene instance owning the controller role
   "ff_", "_ff"

ad 1.

The field "controller" indicates, whether this scene instance holds the controller role for this network sensor. The network sensor together with the CS guarantee that only one scene instance receives "controller"=true at one time for one network sensor.

Sending an SFBool event to the field "requestController" a scene instance can request the controller role for this network sensor.

The setting of "requestController" indicates during initialization of the network sensor, whether the network sensor shall queue up at the CS for getting the controller role.

The fields "controllerRequest" and "controllerGrant" enable the CBSS to reject or to allow a controller request of another scene instance.

Setting the field "followStreamName" to a value not equal the empty string will cause the controller role of the network sensor to "follow" the controller role of another network sensor, i.e. the sensor that is identified by its "streamName" equal to the "followStreamName".

Hence an input to the "requestController" field will not have any effect, if the "followStreamName" field matches the "streamName" of another existing network sensor.

If not any scene instance feeds an event to the "requestController" field and if the "followStreamName" field is empty and if the "requestController" field is initialized to a value of "TRUE" in all scene instances, then the CS and the network sensor will care automatically that one and only one scene instance owns the controller role for this network sensor at any given time in the lifetime of the multiuser session.

ad 2.

Events at fields with the "ff_" prefix will not be distributed evenly to all scene instances, but they will be routed to the scene instance that owns the controller role for this network sensor.

The abbreviation "ff" is inspired by theory of linear differential equations, where the "forcing function" takes a similar role as the "ff" prefix/suffix does for the proposed network sensor enhancement.

5 Examples

Please find following figures just for demonstration of the concepts of the proposal.

The names of the messages between network sensors and collaboration server (CS) are exemplary and need not be identical to the actually used message identifiers.

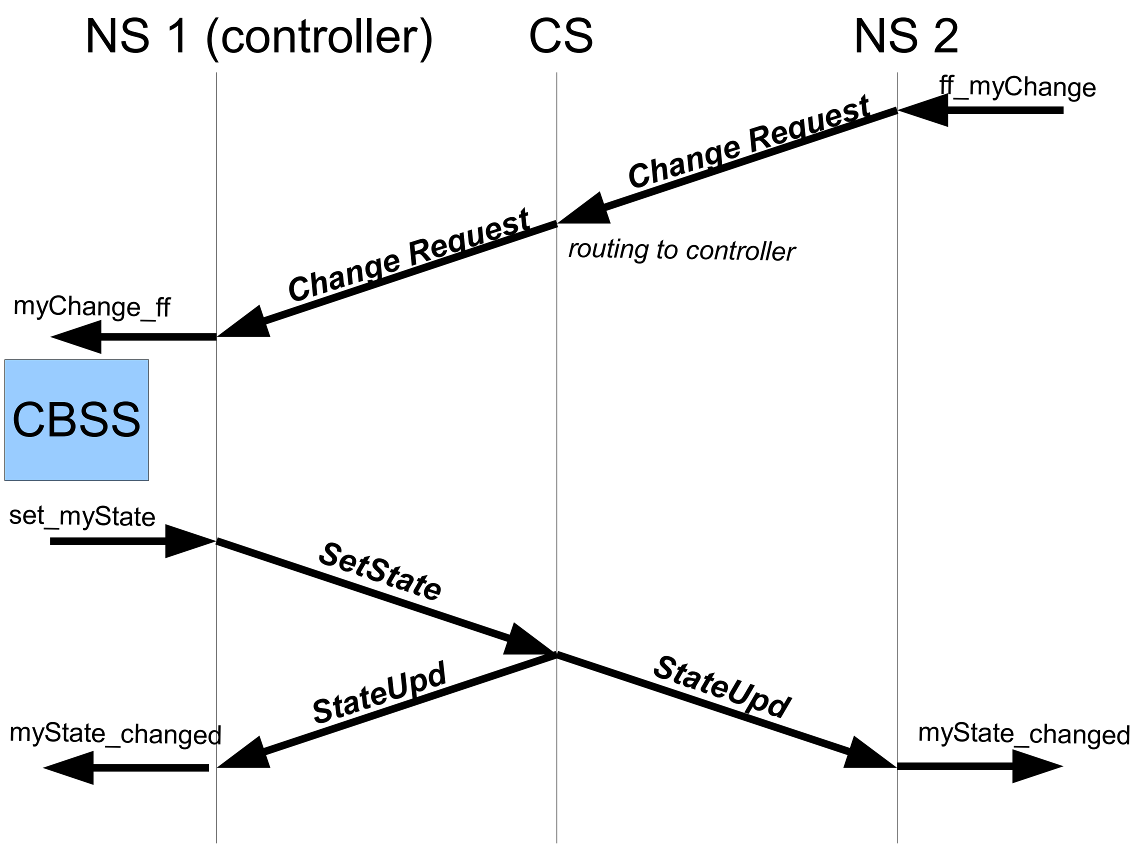

Figure 1: Principle of CBSS with enhanced network sensor

Figure 1 demostrates the basic idea of Client Based Server Software (CBSS). If we talk about one network sensor within the scene, then we have N instances of this network sensor, where N is the number of users logged in, i.e. the number of scene instances.

The "Change Request" (let's call it so) is similar to any other network sensor event in that it is not stored at the CS. On the other hand, it is not distributed to all scene instances, but it is routed to only one scene instance. This is the scene instance that owns the controller role for this network sensor.

Since this routing is done by the CS, we see the CS must keep track of all controller roles.

The CBSS is active in one and only once scene instance (but it is present as e.g. JavaScript in all scene instances).

The CBSS calculates the new value of the state and sends the new value to the CS, who stores it persistently and distributes it to all scene instances, as usual.

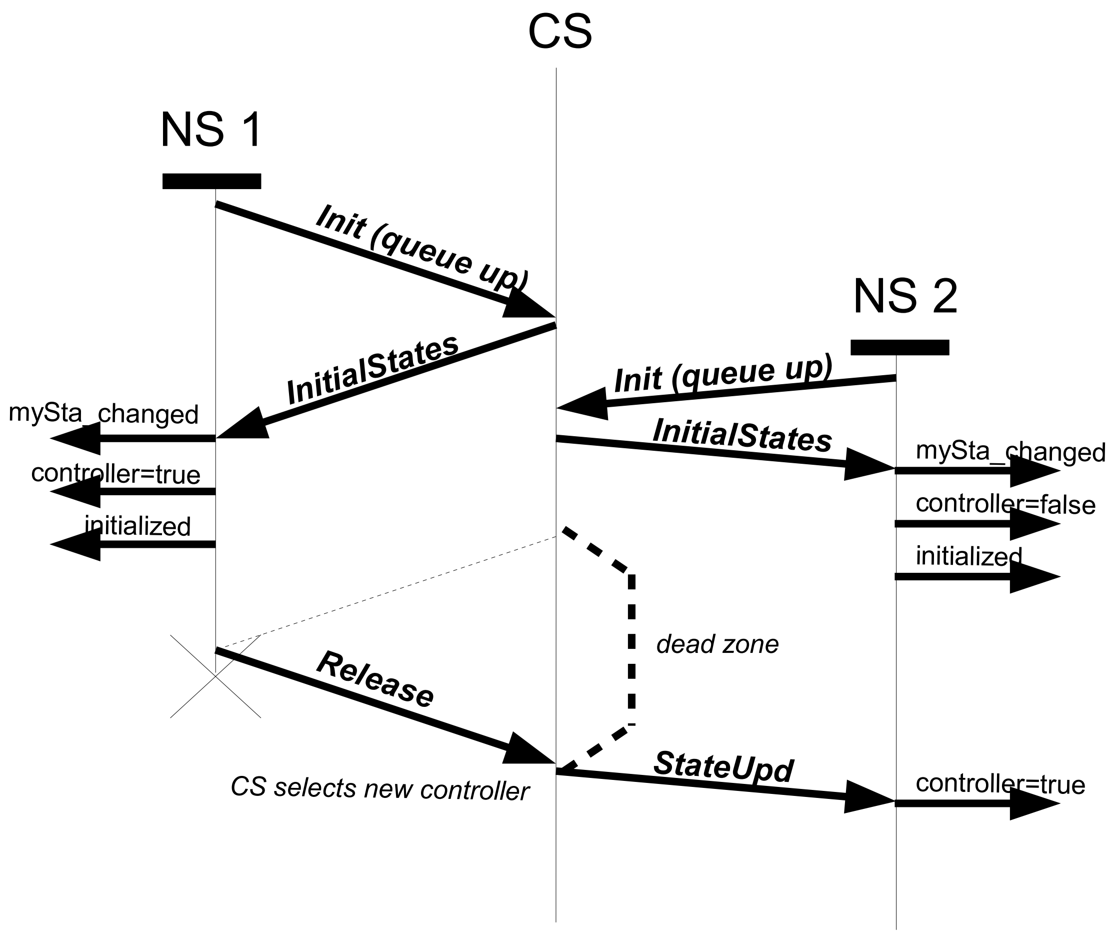

Figure 2: Change of Controller Role without event at "requestController"

Figure 2 Demonstrates, how the CS defines the controller role automatically, if not any scene instance sends the "requestController" event.

- The first scene instance that initializes the network sensor, gets the controller role (if all scene instances initialize "requestController" to "TRUE", otherwise none will get the controller role)
- If a scene instance having the controller role does leave the multiuser session, then the CS defines a new scene instance having the controller role (all scene instances queued up for the controller role, because they have initialized "requestController" to "TRUE")

We see a "dead zone". If a Change Request ("ff_") is received at the CS during this time interval, then the Change Request will not be processed by a controller. The Change Request will get lost.

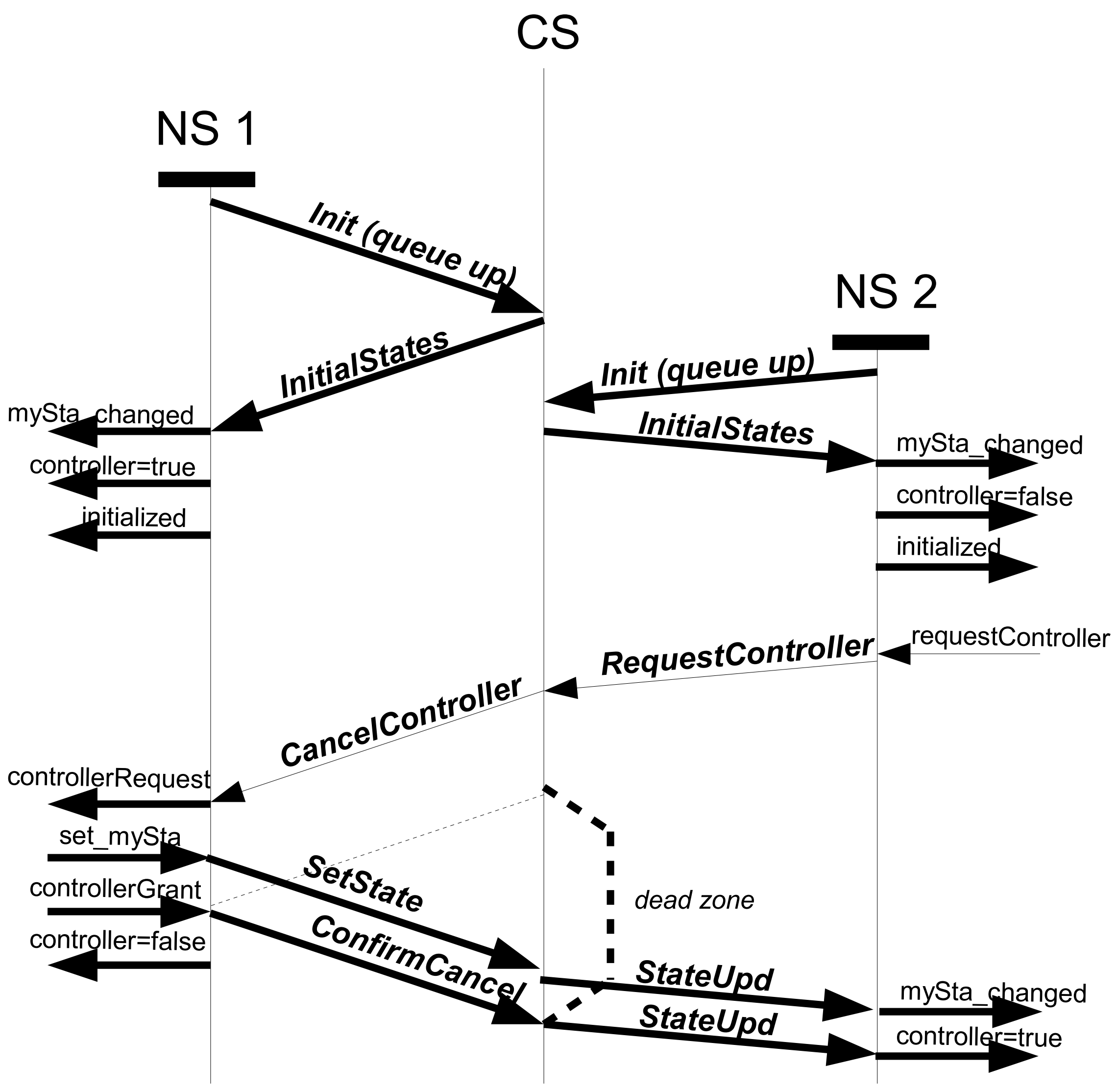

Figure 3: Change of Controller Role with "requestController"

Figure 3 examplifies, how a scene instance (here NS2) can request the controller role and the CBSS (here NS1) must confirm the request. The CBSS **may**, but **needs not** send a last state update, before it confirms the controller request.

A similar function was introduced to the SRR Framework and to the SMUOS Framework, to be able to run the tracer on a selected scene instance and to be sure the CBSS will be traced by the tracer.

However, in conjunction with the SMUOS/C3P idea [1] this might be useful in another sense, too.

We see a "dead zone" again. If a Change Request ("ff_") is received at the CS during this time interval, then the Change Request will not be processed by a controller. The Change Request will get lost.