BCP

Best Current Practices

ALP over RTP

Application Layer Protocol over Real Time Protocol

! ! ! ! ! D R A F T ! ! ! ! !

! ! ! ! ! N O T Y E T I M P L E M E N T E D ! ! ! ! !

1 Introduction

This paper uses experiences I made during the conduction of an X3D hobby project between the years 2008 and 2019.

That project had its concerns about the usage of the Network Sensor Node (NSN) within Simple Multiuser Sessions for simple games (SMS) and use cases generally designed for collaborating (e.g. 3D chat, virtual model railroad, and so on).

I created experimentally some X3D prototypes that could be used as wrappers around the NSN, in order to ease the application of the NSN, hence specializing the general NSN for more concrete use cases.

What I did NOT do, was to define the communication protocol, which the NSNs of the different vendors should use.

Knowing that would be the real challenge the world is waiting for, and having some time at holiday's season 2020, I am taking a few hours to write it down in a systematic way, here in this paper.

1.1 Open Issues

Analyse in detail: RFC 3550 – RTP / RFC 4960 – SCTP / SCTP over UDP

More sophisticated methods of interpolation during replay are FFS

Some kind of micro-authorization and accounting, to be allowed to use this or that model within this or that scene, is FFS

The mechanisms of how to distribute the URLs for loading of dynamic models at runtime, are FFS

If a dynamic model is added to the scene, then new SCTP streams and SSRC IDs have to be allocated for the model. These mechanisms are FFS

SCTP may be transported over UDP, in order to support NAT, ICE and so on. This is FFS.

The transport of real-time media over IP multicast among the scene instances may need something like an "overall multicast SCTP association", too, this is currently FFS

This chapter holds a structural description of the PDUs of the ALP, with special reference to the transport of ALP over RTP and SCTP. A general description independent of the transport protocols is FFS.

Write an I-D: "Collaborative 3D Profile (C3P)" – Shared State / ALP over RTP – RTP over SCTP – SCTP over Multicast

Table of Content

2 Overview

2.1 Network Sensor Node (NSN) – Examples from BS Contact

SrrTrains v0.01 used the Event Stream Sensor of the BS Contact™ Web3D Browser, hence we're starting with their description (following link):
https://www.bitmanagement.de/download/BS_Collaborate/BS_Collaborate_documentation.pdf

The NSN and the related nodes provide following fields (at least):

NetConnection {

field	SFBool	enabled	TRUE	
eventOut	SFBool	isActive		
field	MFString	address	"localhost"	
field	SFInt32	port	0	
field	SFInt32	protocol	0	
field	SFTime	timeOut	0	
field	SFBool	secure	TRUE	}

BSCollaborate {

field	SFNode	connection		
eventIn	MFString	tryLogin		
eventOut	SFBool	loginResult		
eventIn	SFTime	logOut		
eventIn	SFVec3f	userPos		
eventIn	SFRotation	userOri		
field	MFNode	users	[]	
eventOut	SFNode	hasJoined		
eventOut	SFNode	hasLeft		
eventOut	SFNode	hasMoved		
eventIn	MFString	meSay		
eventOut	SFNode	hasSaid		}

UserData {

field	SFInt32	idx	-1	
field	SFString	nickname	""	
field	SFString	avatarString	""	
eventOut	SFString	loginState		
eventOut	SFVec3f	position		
eventOut	SFRotation	orientation		
eventOut	SFBool	isMoving		
eventOut	MFString	chat		
field	SFNode	userData	NULL	}

EventStreamSensor {

field	SFNode	connection	
eventOut	SFBool	initialized	
fields	...	***arbitrarily defined fields similar to a Script node***	}

2.2 General Operational Paradigm of the Network Sensor (Examplified with BS Contact)

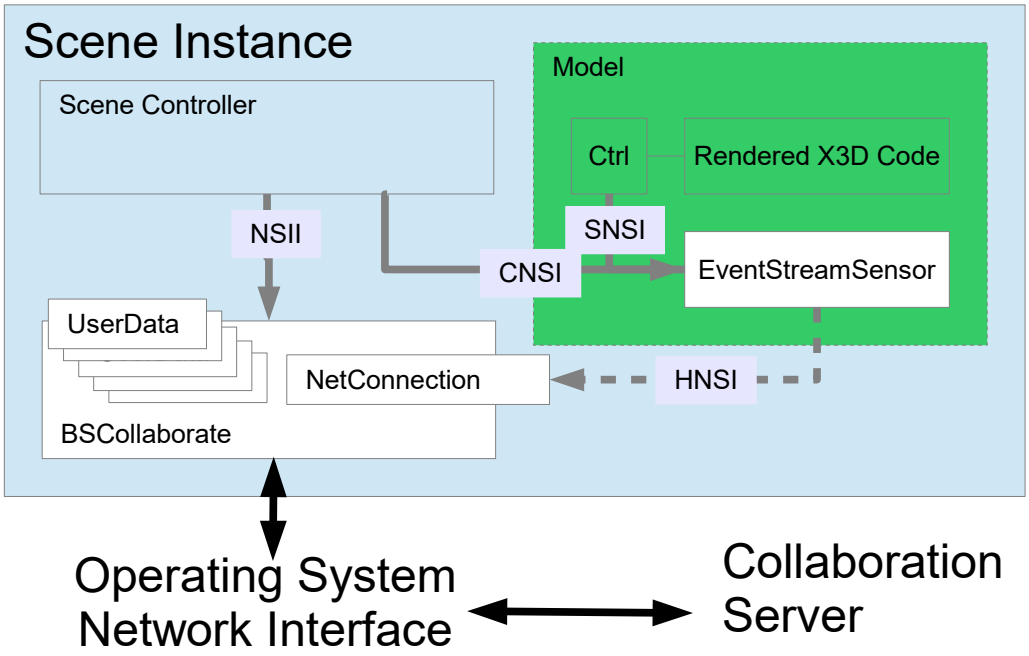

Figure 1: Operational paradigm of the Network Sensor

The **Scene Controller** (a conceptual part of software within the scene, which controls the scene as a whole) uses the **Network Sensor Infrastructure Interface (NSII)**, which is provided by the nodes

- BSCollaborate (all fields),
- NetConnection (all fields) and
- UserData (all fields),

as well as the **Common Network Sensor Interface (CNSI)**, which is provided by the node

- EventStreamSensor (fields "connection" and "initialized"),

in order to realize the **Maintenance Use Cases** and the **Common Use Cases** (see next chapter).

The **Ctrl part of the model** (a conceptual part of software within the model, which controls the model as a whole) uses the **Specific Network Sensor Interface (SNSI)**, which can be arbitrarily defined by some fields of the EventStreamSensor node,

in order to realize the **Model Specific Use Cases** (see next chapter).

2.3 Overview about the Use Cases of the Network Sensor

The Scene Controller and the Ctrl part of the Model shall use the NSII, the CNSI and the SNSI, in order to realize following use cases:

1. The Maintenance Use Cases
   (a) Initialize and authorize the NetConnection, keep it alive
   (b) Load, initialize, purge and unload avatars
   (c) Initialize static NSNs
   (d) Load, initialize, purge and unload dynamic models with embedded NSNs
2. The Common Use Cases
   (a) Movement of initialized avatars
   (b) Gestures of avatars (w. or w./o. body tracking)
   (c) Chat, Voice Chat (both could be realized outside of the X3D scene or inside)
3. The Model Specific Use Cases
   (a) Event Distribution
   (b) Persistent Storage and Distribution of State
   (c) Simple Server Side Calculations
   (d) Customized Client Side Calculations

2.3.1 The Operational Use Cases of the Network Sensor

Introduction

The operational use cases are mostly defined by the "arbitrarily defined fields similar to a Script node" of the EventStreamSensor node.

There's always a pair of fields for each functionality of the NSN, i.e. one field for network output (eventIn) and one field for network input (eventOut).

Assume a scene, where some animation is pre-defined by a Time Sensor and an Interpolator and can be started every now and then by touching some geometry.

Such pre-defined animation can be easily made multiuser capable by distributing a simple SFBool event over the network.

```
EventStreamSensor {
        field       SFNode      connection IS NetConn
        eventOut    SFBool      initialized
        eventIn     SFBool      evt_startAnimation
        eventOut    SFBool      startAnimation_evt      }
```

Assume three scene instances of the users Alice, Bob and Charlie.

Let's further assume that Bob touches the geometry in his scene instance and now the animation should be started in all three scene instances.

Bob's scene instance will route an SFBool event to the field "evt_startAnimation" of the NSN. The prefix "evt_" means, the NSN should distribute an event to all scene instances without storing anything at the server.

As a result the field "startAnimation_evt" will fire in all three scene instances and the animation(s) will start:

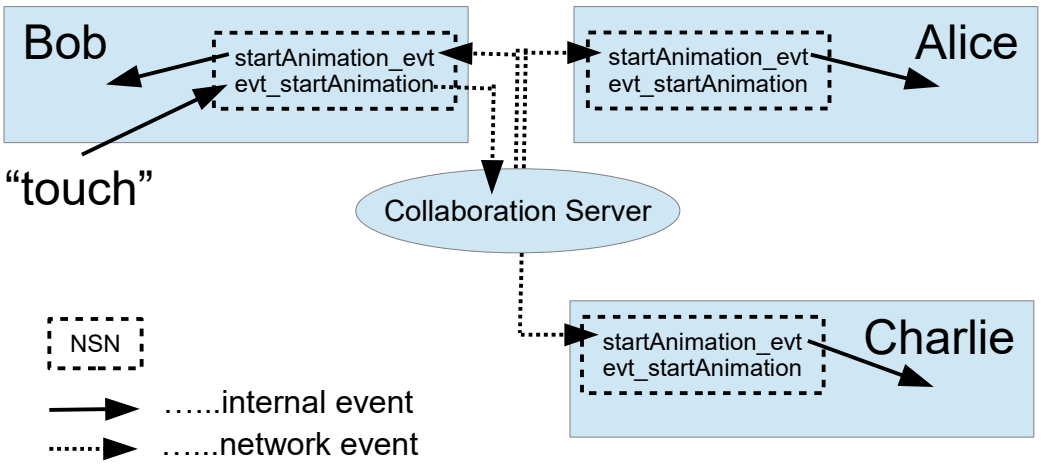

Figure 2: Operational use case "Event Distribution" - three scene instances

Animation and Simulation Paradigm of Single-User X3D

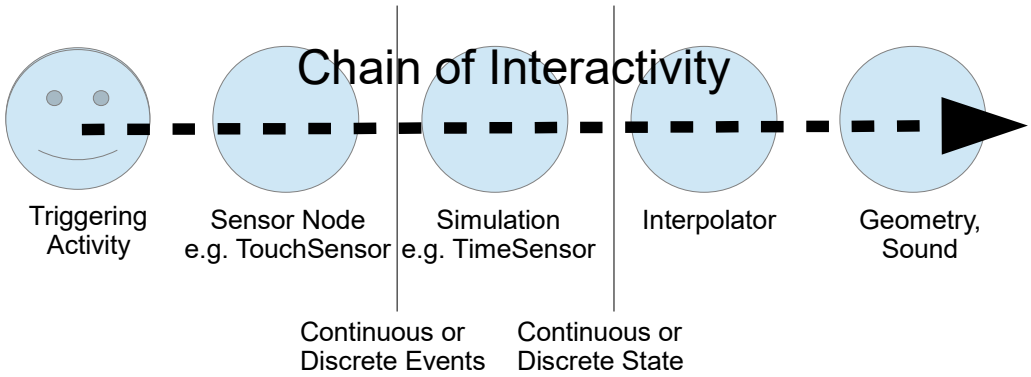

Figure 3: Animation and Simulation Paradigm of Single-User X3D

2.3.1.1 Event Distribution – UC 3.a.

We had a simple example in the introduction already, where a simple event was distributed from the triggering scene instance to all scene instances.

This paradigm can be depicted as follows: First, we can express, where the chain of interactivity is broken to insert the inter-networking with the remote scene instances:

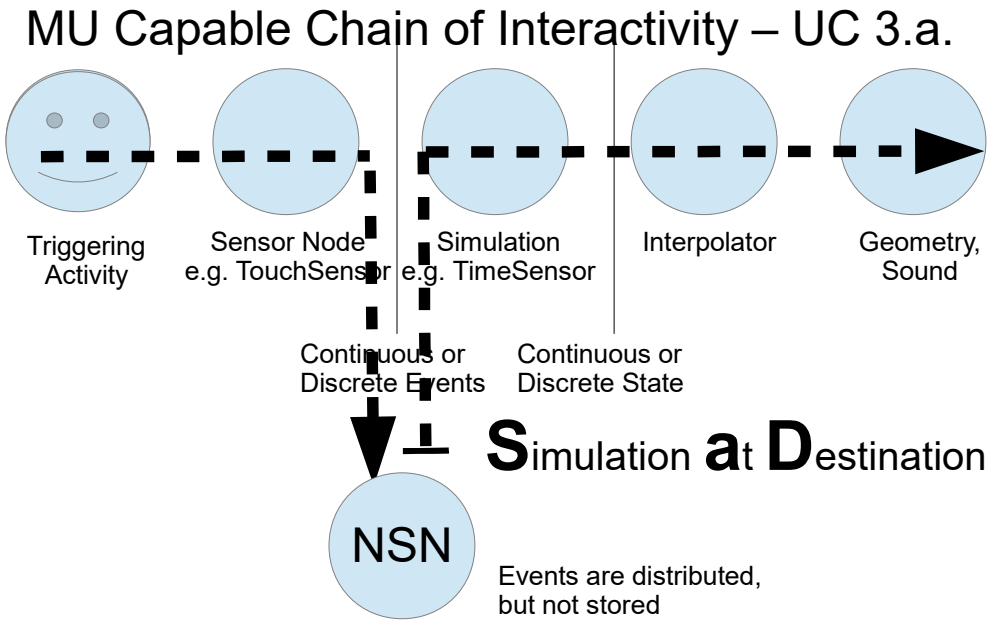

Figure 4: Animation and Simulation paradigm with Event Distribution – UC 3.a.

Second, if we assume three scene instances being present (A, B and C), if we assume user A is the one who triggers the change in the simulation and animation, then we can depict a flow as follows:

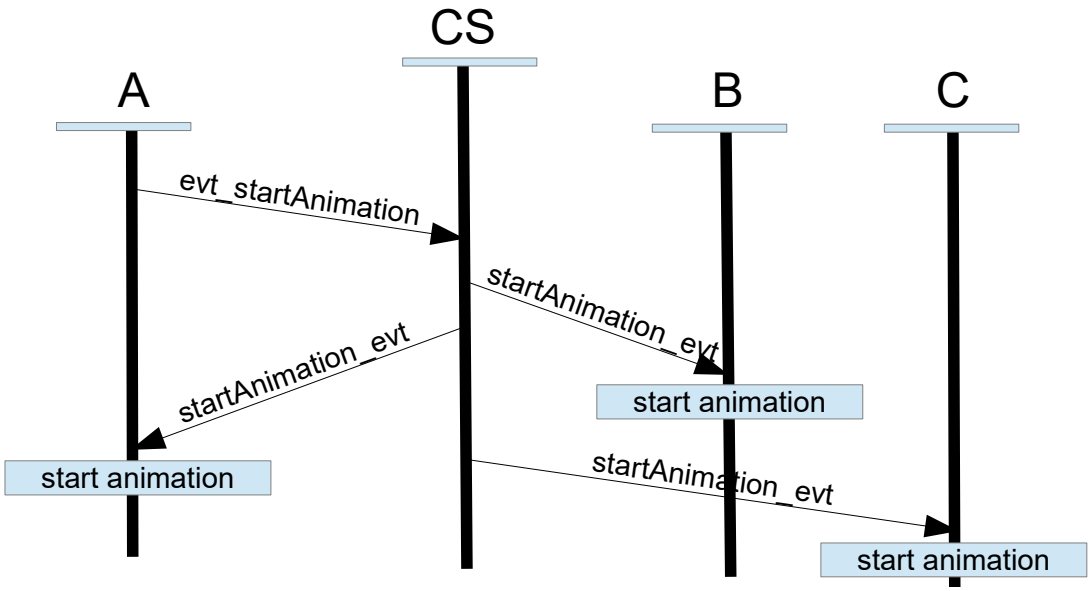

Figure 5: Message flow with event distribution – UC 3.a.

2.3.1.2 Persistent Storage and Distribution of State – UC 3.b.

Use case 3.a. with simple event distribution comes – as we already assumed – very simple.

On the other hand, it comes with the disadvantage of calculating the state from the input events in each and every scene instance again. In case of simple animations and simulations that might be acceptable, but if the animation and simulation gets more sophisticated, then we will fear the state might diverge in different scene instances and we might be afraid of extra calculation burden.

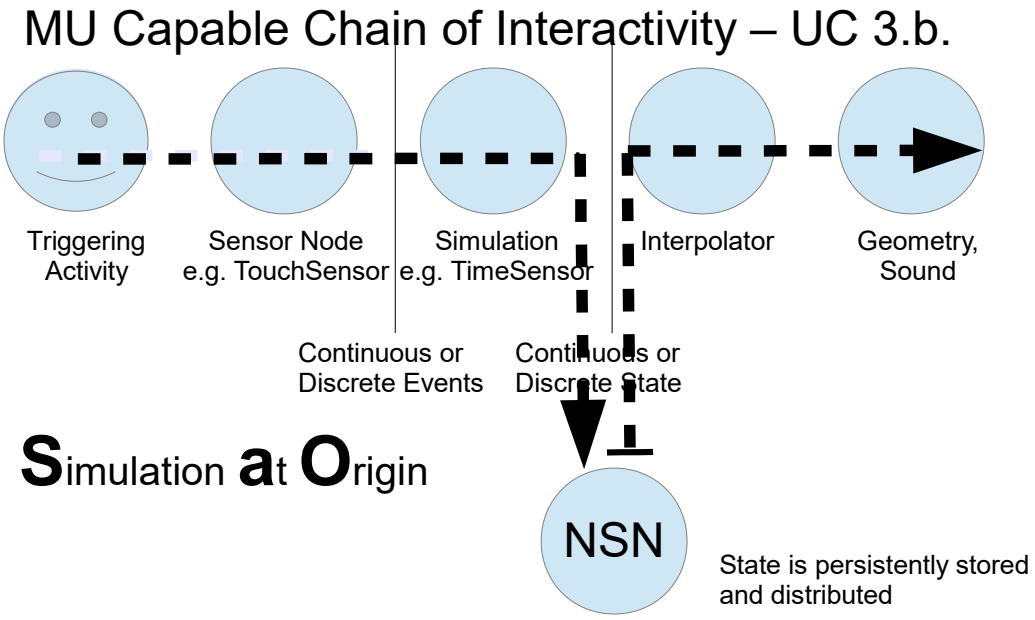

Figure 6: Animation and Simulation paradigm with distribution of state – UC 3.b.

Here we calculate the state in one and only one scene instance, maybe directly from user input, and distribute the state to all scene instances.

In case of initialization of an NSN the current state can be provided by the collaboration server.

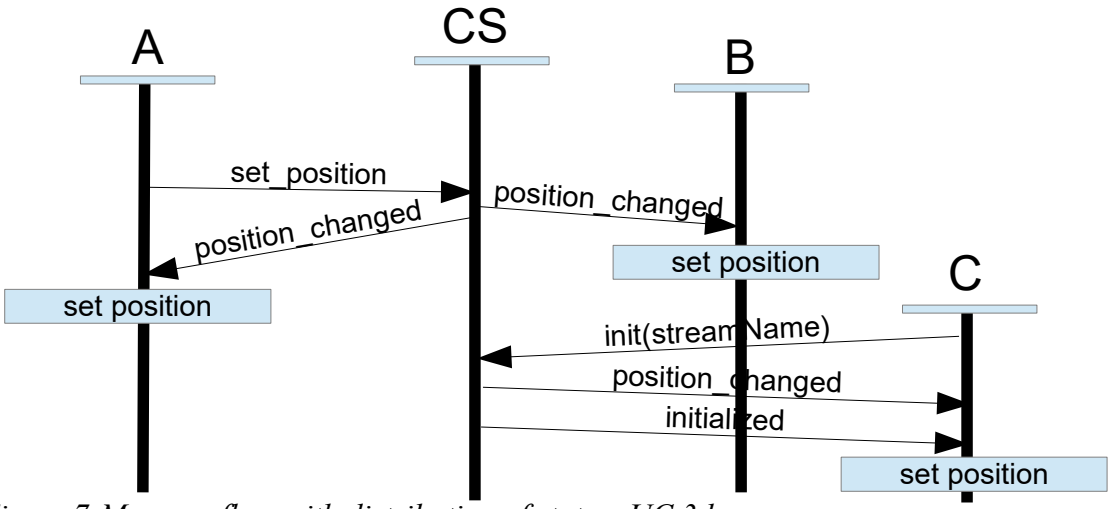

Figure 7:Message flow with distribution of state – UC 3.b.

Disadvantage: One and only one scene instance should be responsible at a time for distributing the state to all scene instances, otherwise the system might cease to be a deterministic system.

2.3.1.3 Simple Server Side Calculations – UC 3.c.

So we have a good solution – UC 3.a. "Event Distribution" – for very simple problems and we have a solution for more sophisticated simulations – UC 3.b. "Persistent Storage and Distribution of State" – which comes with a few disadvantages:

- one scene instance – and only one scene instance – must be responsible for the whole simulation of a model
- handover of a model to another user during the simulation run seems only to be possible by removing the model completely and adding it again for another user

This fits to use cases, where each model is guided by one and only one pilot.

If more than one user contributes input to the simulation of a model, then it seems to be advantageous to do the simulation calculations at the collaboration server:

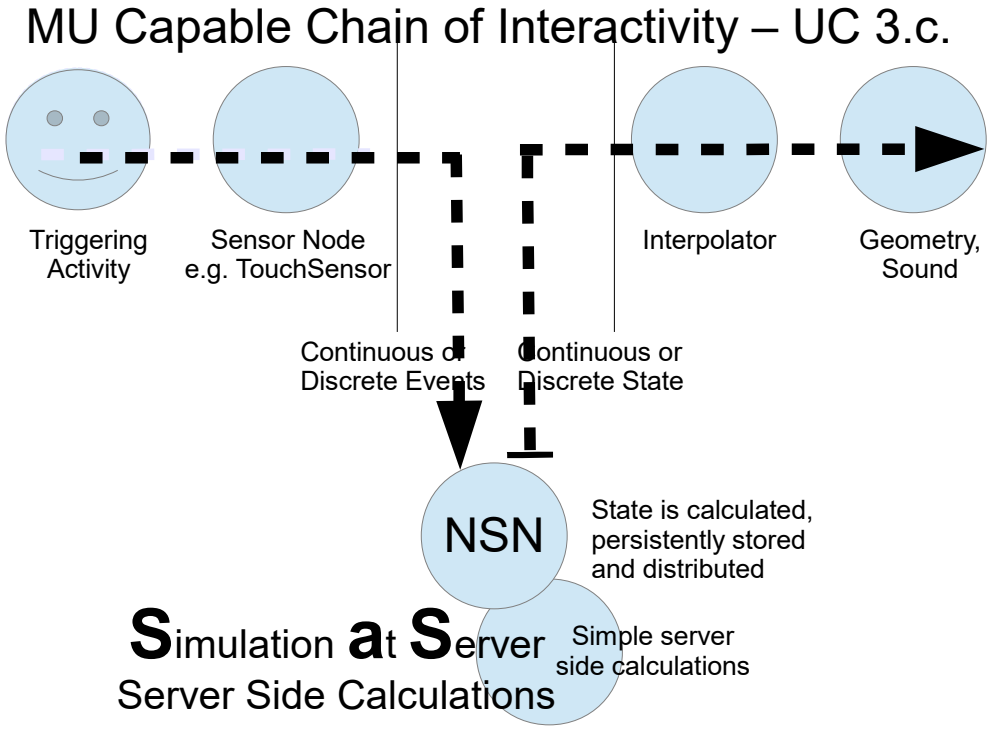

Figure 8: Animation and simulation paradigm with server side calculations – UC 3.c.

Events are e.g. "add_state", "sub_state", if values have to be added to or subtracted from the state or "inc_state", "dec_state", if the state has to be incremented or decremented by one.

Anyway, the server side calculations can only be simple calculations, because they have to be standardized in the X3D standard (otherwise the browser vendor must be cooperating with the server vendor).

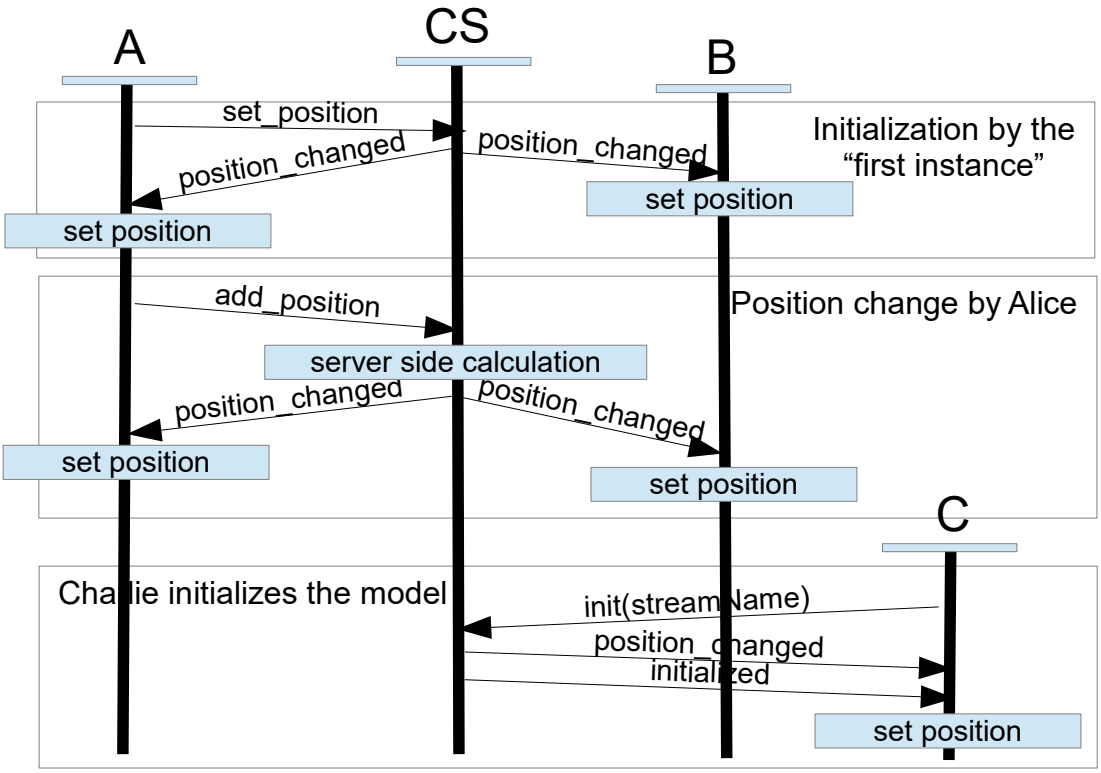

Figure 9: Message flow with server side calculations – UC 3.c.

In case of simple simulations – not as simple as with Event Distribution UC 3.a., but still quite simple – this solution is the best one.

2.3.1.4 Customized Client Side Calculations – UC 3.d.

Solution UC 3.c. is already a quite good solution for quite sophisticated simulations, however, if we need even more flexibility and expendability – if we want to implement functions "on top" of X3D, then we need a more sophisticated solution.

This is the implementation of Customized Client Side calculations, which have to be executed on a clearly defined scene instance and then the state has to be distributed as usual (see UC 3.b.).

Routing the events to that clearly defined scene instance – aka "Controller" – must however be supported by the collaboration server, using the prefix "ff_<fieldname>".

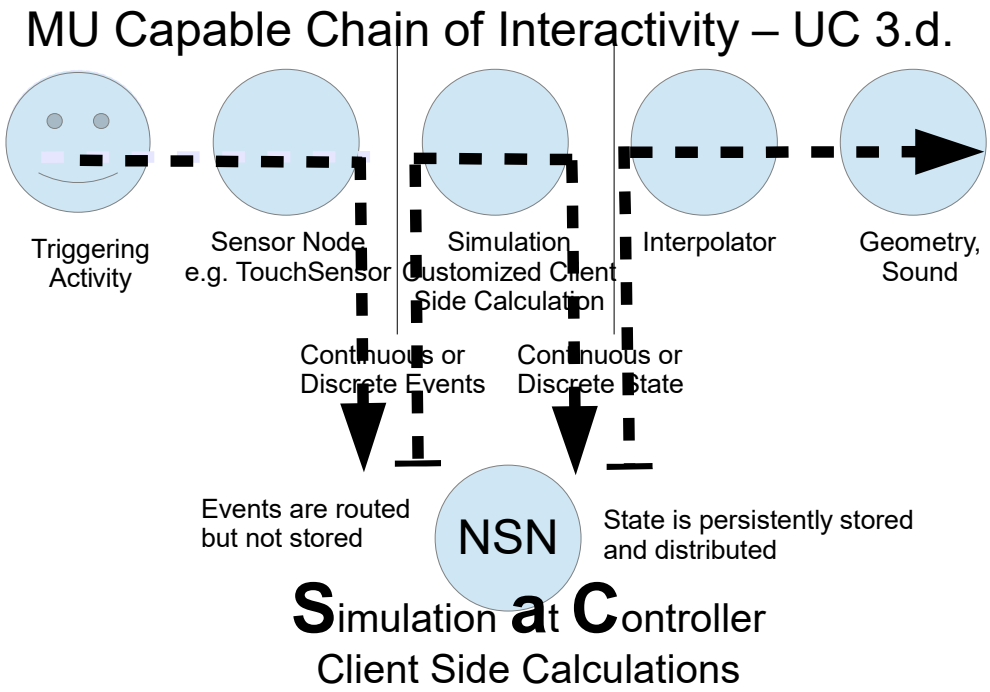

Figure 10: Animation and Simulation Paradigm with client side calculations – UC 3.d.

Hence we can implement quite complex use cases "on top" of X3D, however it comes with a little performance drawback with respect to UC 3.c.:

- The network must be traversed twice – the RTT (round trip time) accounts twice
- The required bandwidth is approx. twice the bandwidth as of UC 3.c.

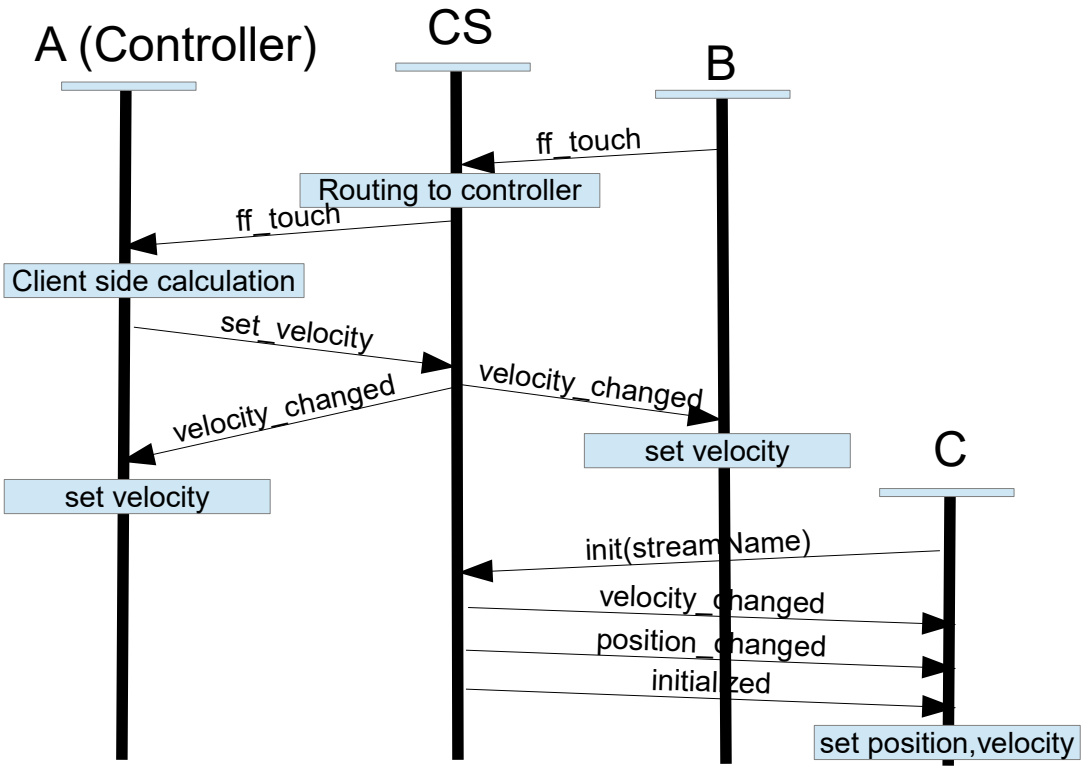

Figure 11: Message flow with client side calculations – UC 3.d.

2.3.2 Real-Time Transport of States and Events

This chapter examplifies a multiuser session with three scene instances A(lice), B(ob) and C(harlie) and a collaboration server CP[1].

The basic idea is to transmit states and events over RTCP APP packets and reliable transport SCTP (ordered service, U bit cleared), if non-realtime behaviour is required, and to transport states over RTP and unreliable transport SCTP (unordered service, U bit set), if realtime behaviour is required.

We assume the usage of SIP for session control, where the ALP RTP session is described by an SDP offer and an SDP answer among the CP and each user.

Additionally, an AVP RTP Session for Voice Chat might be described by the SDP in parallel to the ALP RTP session.

SIP might be used to convey the URLs of the avatars from the CP to the users A, B and C.

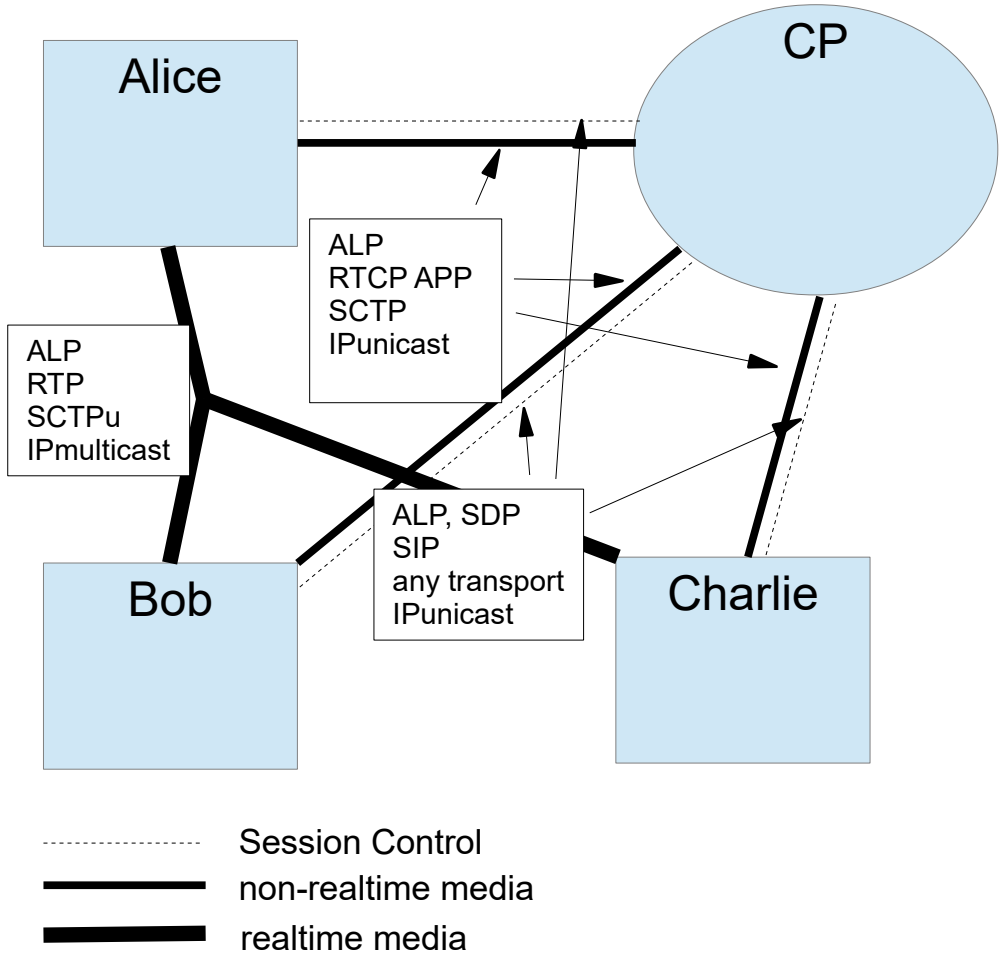

Figure 12: Example – three scene instances A, B and C and a server CP

1 Connectivity Platform (CP) is the term that is used in the Concepts' Descriptions of the SrrTrains v0.01 project, in order to denote some "evolved collaboration server CS"

2.3.2.1 Synchronization Source SSRC

RTP defines the term of a "Synchronization Source (SSRC)", where each SSRC is the source of one stream of real-time media.

The synchronization source is accompanied by a linear clock, which is taken to generate a timestamp in each RTP packet that has its origin at this SSRC.

The stream represents a (quasi-)continuous function state(t), which shall be played back at the destination without changes in the shape, but with a noticeable delay ΔT, if necessary.

state(t) ---> state(t - ΔT)

RTP was designed for functions of shapes with constant sampling rate, e.g. the famous 8000 Bytes/sec of telephony systems, however it can be adapted for 3D Multiuser Sessions with variable frequencies of – let's say – 10 to 100 FPS, as we try to explain in the present chapter 2.3.2 .

For the case of 3D multiuser worlds we would like to define following relations:

1 SSRC <---> 1 Model <---> 1 Stream

This means that we assume that a model always has a small spatial extent, so that a single proper time can be used in the timestamp of the RTP packets for the entire model.

The following figure displays a situation where 3 models are simulated in Alice's scene instance (according to use case UC 3.b.) and the states of the models are then distributed in three real-time streams through the ALP RTP session over the IP multicast address and port number.

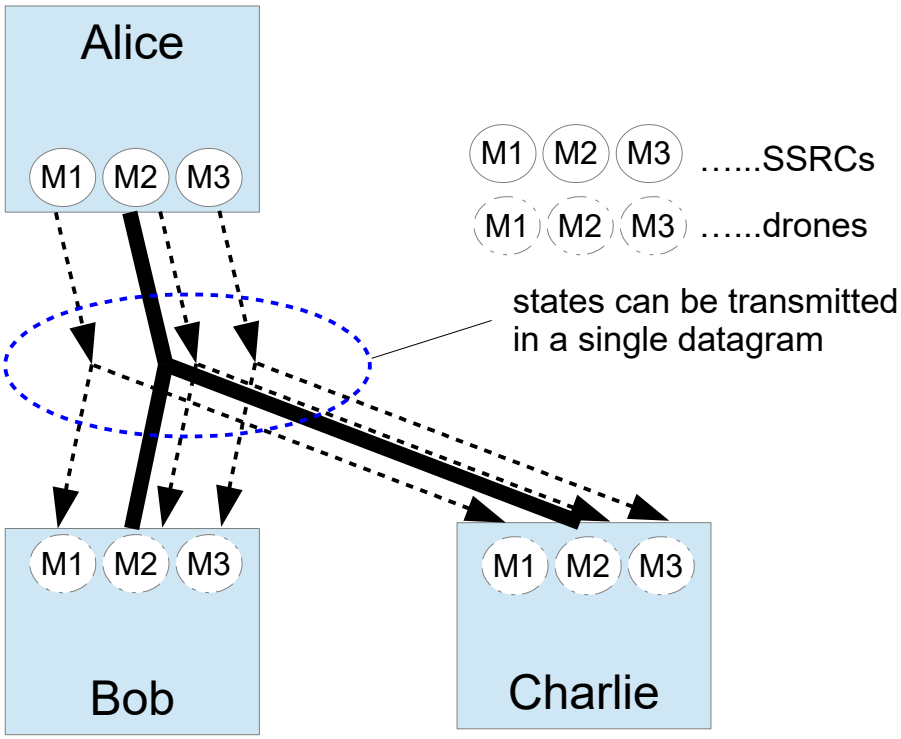

Figure 13: Real-time transmission of continuous state(t), several SSRCs

2.3.2.2 Streaming of Avatar State

2.3.2.2.1 Protocol Overhead

If we are streaming the state of modern H-Anim avatars, then – due to the high number of degrees of freedom – one sample of the state comprises hundreds or even thousands of bytes of data (this is applicable even more, if we decide for a representation in a high level syntax like XML, JSON, YAML or similar).

So we need not worry about the amount of bytes for the RTP Header (12 bytes), the SCTP Common Header (12 bytes) and the IP header (20+ bytes).

If we transmit the state of classical VRML avatars, on the other hand, then the state consists of

- field SFVec3f position 3 x 4 bytes and
- field SFRotation orientation 4 x 4 bytes,

i.e. just 28 bytes in binary encoding.

Hence the transmission of just one sample of avatar state per IP datagram would be a waste of network resources, and we must look for some improvements, e.g. the packing of several samples into one single datagram.

2.3.2.2.2 Packing Several Samples of State into One Datagram

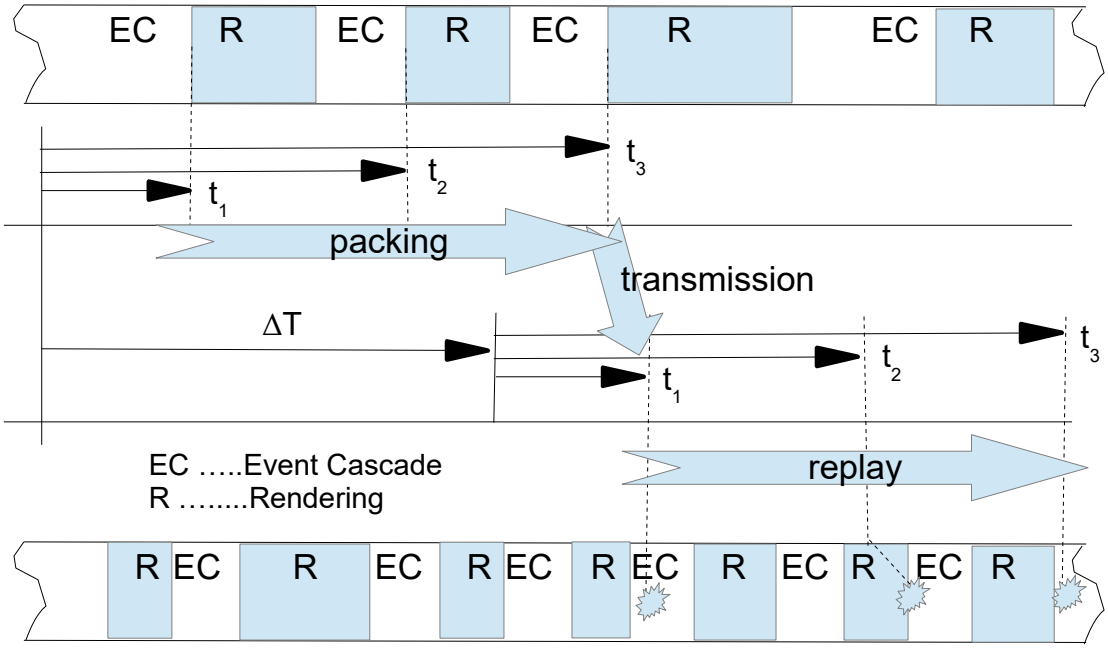

Figure 14: Frames in two scenes, packing of three frames (R) into one datagram

Here the output events of the NSN are fired during the first applicable event cascade (after the timestamp from origin). More sophisticated methods of interpolation during replay are FFS.

2.4 MIDAS Objects

The SrrTrains v0.01 project examplified an approach, where the whole "Chain of Interactivity" regarding a specific model was outsourced to a specialized X3D prototype, the so-called **MIDAS Object**.

Hence an additional layer would be introduced between X3D Player and X3D Scene, the so-called SMUOS Framework.

This is just mentioned as an "FYI".

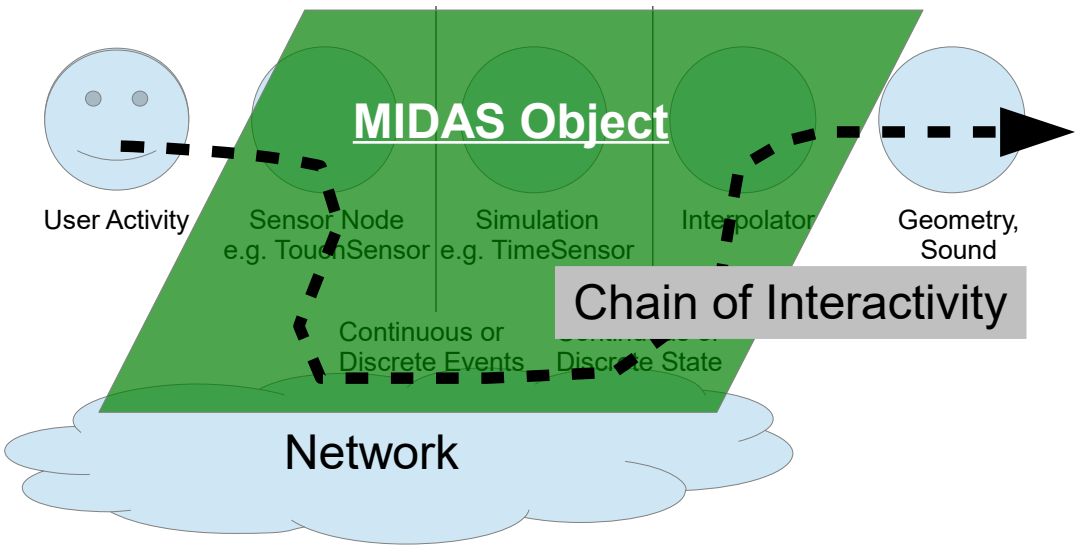

Figure 15: Outsourcing the Chain of Interactivity

The NSNs, generally the whole MU capability would be hidden within the MIDAS Objects, hence the scene author wouldn't see much difference between single-user scenes and multi-user scenes.

3 Application Layer Protocol (ALP) – Transmitted over RTP

The Protocol Data Units (PDUs) of the ALP have been defined during a brainstorming in spring 2019, keeping in mind the Maintenance Use Cases and the Operational Use Cases of the ALP:

Use Cases:
https://github.com/christoph-v/spark/wiki/Operational-Paradigm

Overview about the ALP brainstorming:
https://github.com/christoph-v/spark/blob/master/3rd-party-input/esdp-bs/esdp-bs-GPS.txt

Details of the PDUs:
https://github.com/christoph-v/spark/blob/master/3rd-party-input/esdp-bs/esdp-bs-PDUs.txt

The transport of the **ALP over the RTP** and the **required dynamic behaviour** have been defined in December 2020, please find the according WIKI pages here:
https://github.com/christoph-v/spark/wiki/Transport-RTP-et-al
https://github.com/christoph-v/spark/wiki/Dynamic-Behaviour-of-the-SMS

3.1 ALP – Operational Principles in Case of ALP over RTP

3.1.1 AAA

The user is authenticated by an external session control protocol (e.g SIP or XMPP, ...).

During the registration with that protocol, the scene instance receives a username and an authorization token for the ALP.

Additionally, it might receive authorization tokens or URLs for one or more avatars.

Accounting by session duration is done by the external session control protocol.

Some kind of micro-authorization and accounting, to be allowed to use this or that model within this or that scene, is FFS (besides the authorization for this or that avatar, which might be necessary for the recognition of the virtual identity).

3.1.1.1 Login to the CP via ALP

Registration and Session Setup with the external protocol may happen, before any X3D content has been loaded, thus enabling an "early start" of the voice chat, chat, etc.

After the client has received the ALP authorization token from the external session control, then it must load the X3D scene – at least the central parts of the scene must be loaded that are necessary for performing the login with the CP.

The URL of the scene might have been received via the external protcol, too, or it was already locally installed together with other local resources or it may have been received via any other channel.

Hence the SCTP association between scene instance and CP is now authorized for use and must be maintained periodically.

The CP enforces the operator's policy, whether used avatars have to be authorized or not. Authorization tokens for avatars may be managed by the CP or they may be managed by the external session control (e.g. at the HSS).

3.1.2 Loading of Dynamic Models

The mechanisms of how to distribute the URLs for loading of dynamic models at runtime, are FFS.

If a dynamic model is added to the scene, then new SCTP streams and SSRC IDs have to be allocated for the model. These mechanisms are FFS.

Furthermore the model and all its network sensors have to be initialized.

If a dynamic model is removed from the multiuser session, then the state of the model must be purged from the CP.

3.1.3 SCTP Associations

The network connection between scene instance and collaboration server is realized as SCTP association, where the scene instance takes the role of the SCTP client and the collaboration server takes the role of SCTP server.

Multihomeing may be used, in particular at the server side.

SCTP may be transported over UDP, in order to support NAT, ICE and so on. This is FFS.

The transport of real-time media over IP multicast among the scene instances may need something like an "overall multicast SCTP association", too, this is currently FFS.

3.1.4 SCTP Streams

Each Network Sensor streamName, i.e. each model and each "internal entity" (e.g. UserData), will need:

- an SCTP stream at each SCTP association between scene instance and collaboration server
- in case of real-time transport: an SCTP stream at the "Overall SCTP Association for Multicast Transport"

Stream IDs are defined by RFC 4960 as being 16 bit numbers. Hence the overall number of Models and "internal entities" in a multiuser session is restricted to 64k.

3.1.5 SSRC IDs

SSRC IDs are defined by RFC 3550 as being 32 bit numbers. Hence the 64k models can be multiplied by a sufficient number of scene instances / users in one and the same multiuser session, before reaching the limits.

3.2 ALP – Protocol Data Units – for any Transport Protocol

3.2.1 Login Request/Grant (LI-R/LI-G), Login Challenge (LI-CH)

Login Request (LI-R)

A PDU, which is sent from one scene instance to the login server that is responsible for the user, who wants to login at this scene instance

Login Challenge (LI-CH)

A Login Request may invoke a Login Challenge, which is sent back to the scene instance in order to proceed with the authentication and authorization of this user.

When the scene instance receives the Login Challenge, it must send another Login Request with the final credentials

Login Grant (LI-G)

The Login Request with the final credentials invokes a Login Grant which is sent back to the scene instance. Now the Network Connection between scene instance and server is authorized and the scene instance grants access to the user and authorizes actions according to the user profile

3.2.2 State Upd. Request (SURE), State Upd. Notification (SUN)

State Update Request (SURE) – aka "Reset State"

A PDU, which is sent from one scene instance to the server [hierarchy] or to the scene instance that owns the controller role for the stream (Network Sensor) in question, or – in case of direct transport over IP Multicast – from one scene instance to all other scene instances and to the server [hierarchy]

- it contains the set of states with their values and types that have been authoritatively calculated by the simulation or by the initialization software and shall now be updated and optionally persistently stored at the server [hierarchy] or at the controller instance
- it is invoked by set_<state> event(s) and it invokes <state>_changed event(s) at the interface of the Network Sensor or by similar means of similar SW objects
- at the server [hierarchy] or at the controller instance, this PDU invokes a SUN PDU with the new values of all states that have been touched

State Update Notification (SUN) – aka "State Changed"

A PDU, which is sent from the server [hierarchy] or from the controller instance of the stream (Network Sensor) in question to all scene instances that have subscribed to the stream or to all scene instances, according to the state that has been stored persistently

- it contains the new or current value (if the value has not been updated) of some or all states of a stream (Network Sensor)
- it invokes the <state>_changed event(s) at the interface of the network sensor or some similar means of similar SW objects

3.2.3 State Change Request (SCR)

State Change Request (SCR) – aka "Server Side Calculations"

A PDU, which is sent from one scene instance to the server [hierarchy] or to the scene instance that owns the controller role for the stream (Network Sensor) in question

- it indicates for each affected state a required operation and the required parameter value for the operation
- invoked by add_<state>, inc_<state>, ... or similar event(s) at the interface of the Network Sensor or by similar means of similar SW objects
- at the server [hierarchy] or at the controller instance, this PDU invokes a SUN PDU with the new values of all states that have been part of the SCR.

3.2.4 Broadcast Event (BEV), Route Event (REV)

Broadcast Event (BEV) – aka "Event Distribution"

A PDU, which is sent

a) from one scene instance to the server [hierarchy] and then forwarded to all scene instances that have subscribed to the stream (Network Sensor) in question or to all scene instances

b) from one scene instance to all scene instances

- it is invoked by evt_<event> and it invokes <event>_evt at the interface of the Network Sensor or at similar means of similar SW objects

Route Event (REV)

A PDU, which is routed from one scene instance to the scene instance that owns the controller role for the stream (Network Sensor) in question

- it is invoked by ff_<event> and it invokes <event>_ff at the interface of the Network Sensor or at similar means of similar SW objects
- is used as input for the simulation at the controller, hence it might trigger either a SURE to one, some or all scene instances or a SURE to the server [hierarchy]

3.2.5 Subscribe to Stream (STS), State Purge and Query (SPQR)

Subscribe to Stream (STS)

A PDU, which is sent by a Scene Instance that wants to initialize one or more Network Sensors

State Purge and Query Request (SPQR)

A special PDU to annihilate the value of states at the server or to query for the current state

- invokes SUN back to originator (in case of purging and querying)
- can invoke SUN to one, some or all scene instances (in case of purging)

3.2.6 Reserved States and Events

As explained in the present chapter, we need some reserved <fieldnames> for reserved states and events.

If a network sensor has a customer specific state or event that collides with one of those, then we can distinguish them by prefix:

user.<reservedFieldname> not equal system.<reservedFieldname>

3.2.6.1 Reserved State "obco", Reserved Event "requestObCo"

For the handling of Use Case UC 3.d. - Customized Client Side Calculations – we need to distinguish all scene instances from each other. Therefore, the CP **assigns a unique SFInt32 sessionId to each and every scene instance**.

Now we can assign the "Controller Role" (i.e. Object Controller – ObCo) to each and every NSN.

With the reserved state "obco" and the reserved event "requestObCo" we can handle the ObCo role, as follows. Those have also to be recognized and specially handled by the CP.

(currentState.obco = = sessionId) <==> this scene instance has the controller role for this object
event.requestObCo = = sessionId <==> the controller role is requested for sessionId

3.3 Detailed Syntax of the PDUs

This chapter holds a structural description of the PDUs of the ALP, with special reference to the transport of ALP over RTP and SCTP. A general description independent of the transport protocols is FFS.

3.3.1 Parameters and their Meaning

3.3.1.1 <streamName>, <networkSensorId> and <fieldname> (3x SFString)

The <fieldnames> of the NSN are set by the scene author in order to define states and events of each NSN.

E.g.

- **eventIn** **SFBool** **evt_touched**
  sends a network event with **<fieldname> == "touched"**
- **eventOut** **SFFloat** **position_changed**
  defines a state with **<fieldname> == "position"** and receives changes of the state from the network
- **eventIn** **SFFloat** **set_position**
  (re-)sets the value of a state with **<fieldname> == "position"**
- **eventIn** **SFFloat** **add_position**
  defines a change request for state **<fieldname>=="position"**
  sends the value to be added to the state

States and events **are identified by <streamName> + <networkSensorId> + <fieldname>** (three SFString values).

The <streamName> and the <networkSensorId> of each NSN are set tby the scene author in order to organize the states and events with respect to models, avatars and other entities.

An Example:

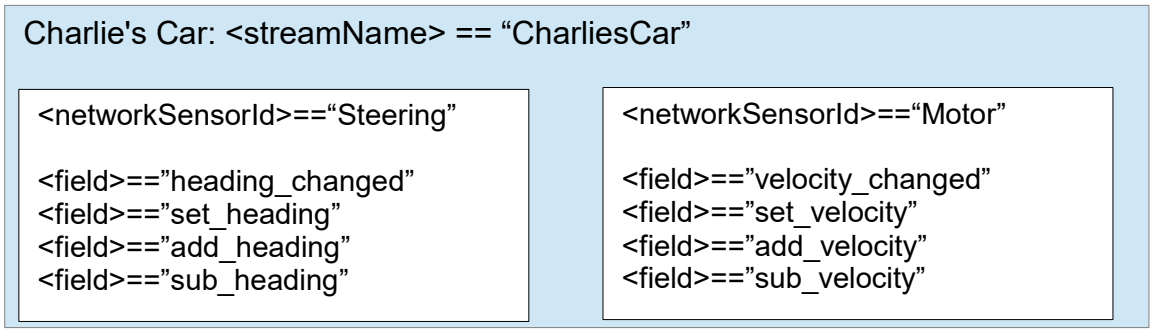

Figure 16: Example of a model with <streamName>, <networkSensorIds> and <fields>

Description:

Each network sensor needs to be identified. If a PDU is received from the network, the Web3D Browser must somehow know, which of the NSNs is the intended destination of the content.

We suggest to use a "two-level identification":

- Each **model**, each **avatar** and each **other entity** is one "cluster" of shared state, which may be realized by one or more NSNs.
  This cluster is identified by a <streamName>.
- Within the cluster, **each single NSN is identified by a <networkSensorId>**.

Within one multiuser session, the tuples "<streamName> + <networkSensorId>" must be distinct for each NSN and they must be identical in all scene instances.

The <streamName> is the basis for multiplexing the data of more than one model over more than one channel of the lower layers (transport protocols).

In the case of the present paper, the media stream of each model is mapped to one SCTP stream.

The SCTP stream IDs are 16 bit numbers, the mapping between streamName and SCTP stream ID is established during initialization of the NSN.

3.3.1.2 Username and Token (2x SFString)

Username and Token can be used as final credentials in a LI-R (Login Request). In the case of the present paper, both are received via some means of the external session control protocol (e.g. SIP).

Both are SFString values.

3.3.1.3 Expires (SFInt32)

The CP tells the Scene Instance, whether a LI-R was successful and how long the Scene Instance may wait, before it refreshes the Login.

Expires is a value in seconds (e.g. 600 means 10 minutes).

If Expires equals zero, then the Scene Instance must interpret the LI-G as Login Reject and shall not try another login with the same credentials.

3.3.1.4 Template (an unordered list of Structures)

When sending an STS to the CP during initialization, each NSN collects the information from all its state definitions. The information from each "eventIn <Type> <fieldname>_changed" is added as a row to an unordered list of structures.

An Example:

```
<STS>
        <streamName>CharliesCar</streamName>
        <StateDeclaration networkSensorId = "Steering" type = "SFFloat" fieldname = "heading"/>
        <StateDeclaration networkSensorId = "Motor" type = "SFVec3f" fieldname = "velocity"/>
</STS>
```

State Declarations of more than one NSN may be added to one STS, as long as all belong to the same <streamName>.

These state declarations are used by the CP to allocate memory and store the global TOS for each <streamName>.

Global TOS = Global Template of States.

3.3.1.5 currentState (an Ordered List of Structures)

As an answer to an STS PDU or to an SPQR PDU, the CP sends a SUN (streamName, currentState) PDU to the requesting Scene Instance, and – in case of a Purge – to all other Scene Instances, too.

The currentState contains one row for EACH row of the global TOS, also the values of the states.

An Example:

```
<SUN>
        <streamName>CharliesCar</streamName>
        <State networkSensorId = "Steering" fieldname = "obco" value = "26778"/>
        <State networkSensorId = "Steering" fieldname = "heading" value = "NULL"/>
        <State networkSensorId = "Motor" fieldname = "obco" value = "26778"/>
        <State networkSensorId = "Motor" fieldname = "velocity" value = "NULL"/>
</SUN>
```

These states have not yet been initialized, therefore the currentState (full state) contains some NULL values.

The "obco" state has been inserted by the CP. It defines the Scene Instance that owns the controller role. The "obco" value is never NULL.

3.3.1.6 NewState (an Ordered List of Structures)

A newState is contained in a SURE or in a SUN PDU.

It NEVER contains states that have not been initialized (i.e. it never contains NULL values).

```
<SUN>
        <NewState networkSensorId = "Steering" fieldname = "obco" value = "26778"
        <NewState networkSensorId = "Steering" fieldname = "heading" value = "0.56"/>
        <NewState networkSensorId = "Motor" fieldname = "obco" value = "26778"
        <NewState networkSensorId = "Motor" fieldname = "velocity" value = "12 34.6 0"/>
</SUN>

<SURE>
        <NewState networkSensorId = "Steering" fieldname = "obco" value = "26778"
        <NewState networkSensorId = "Motor" fieldname = "obco" value = "26778"
</SURE>
```

Note: if the SUN contains a <streamName>, then the state is a currentState. If the SUN does not contain a <streamName>, then the state is a newState

3.3.1.7 events, changeRequests (ordered Lists of Strcutures)

```
<BEV> | <REV>
        <Event networkSensorId = "Steering" fieldname = "touched" value = "true"/>
        <Event networkSensorId = "Motor" fieldname = "touched" value = "true"/>
</BEV> | </REV>

<SCR>
<Event networkSensorId = "Steering" fieldname = "heading" operation = "add"value = "10.0"/>
<Event networkSensorId = "Motor" fieldname = "velocity" operation = "sub" value = "1 0 0"/>
</SCR>
```

3.4 BCP Scenarios in Case of ALP over RTP

3.4.1 Login to the CP

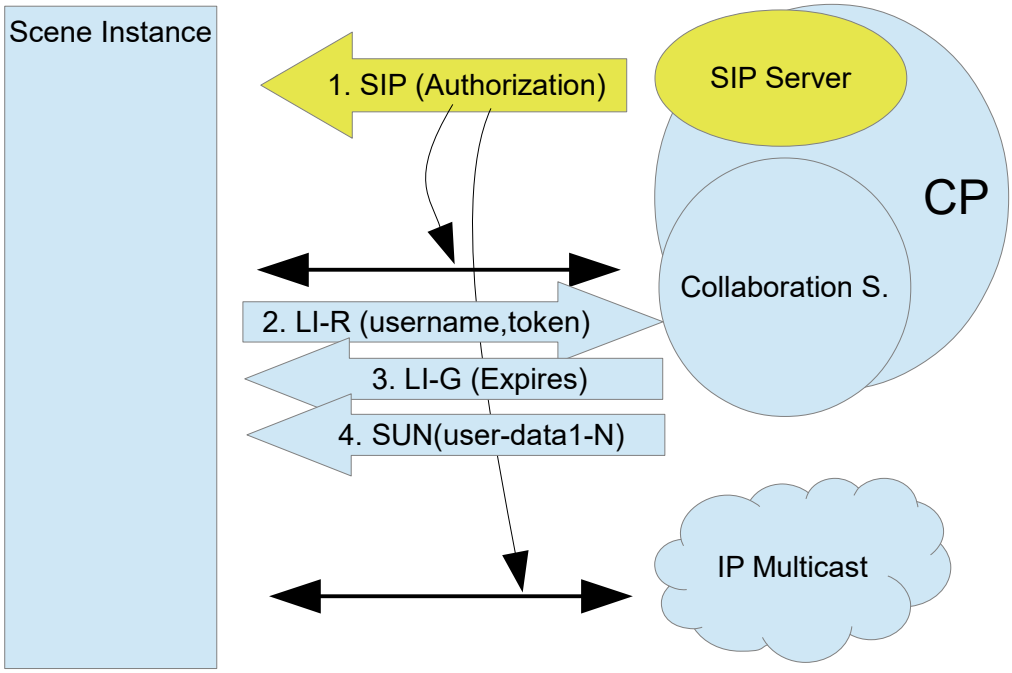

Figure 17: Login with the CP, Authentication by SIP Server

3.4.2 Initialization of a Stream (of all NSNs of a Model)

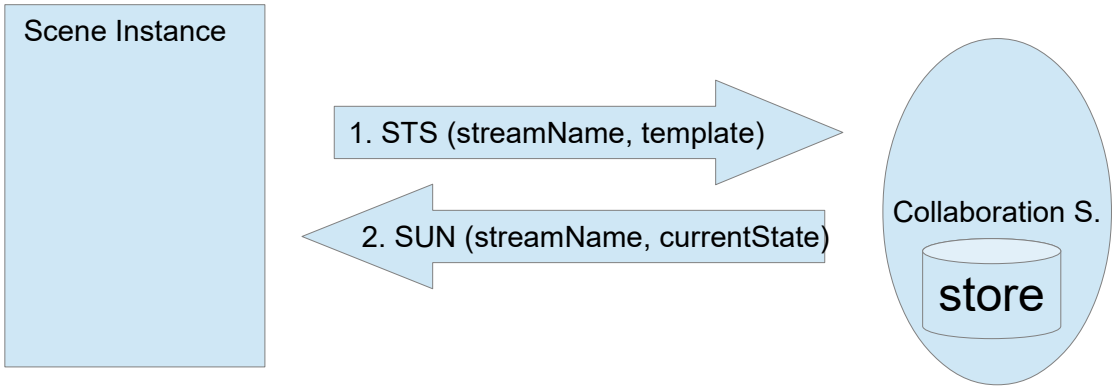

Figure 18: Initialization of the NSNs and of the SCTP stream of a streamName

The initialization of the SCTP stream might be followed by a (Re-)Set of the state, if the scene instance has got a valid "initialization decision". This may happen, e.g. if "currentState" was empty or if "currentState.obco = = me" was true. See Figure 20 in chapter 3.4.4 .

3.4.3 Event Distribution (UC 3.a.)

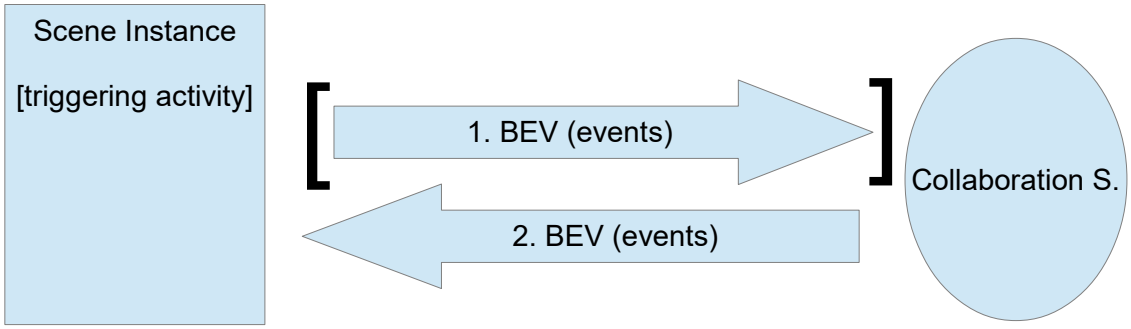

Figure 19: Distribution of Events (UC 3.a.) [with triggering activity]

3.4.4 (Re-)Setting a State (UC 3.b.)

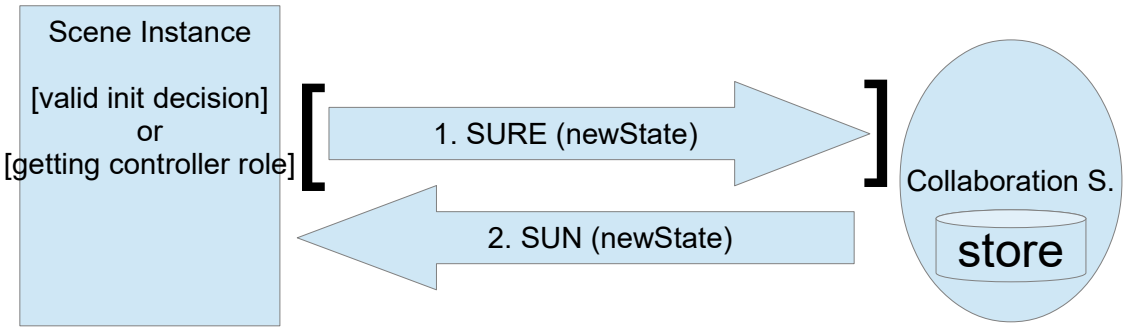

Figure 20: (Re-)setting a state [with valid init decision or after taking the controller role]

For "valid init decision" refer to chapter 3.4.2 , for "getting the controller role" refer to chapter 3.4.8 .

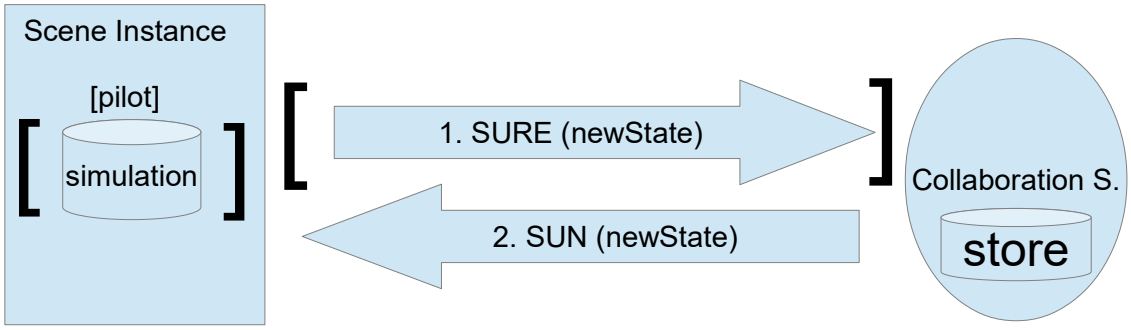

Figure 21: (Re-)Setting a State (UC 3.b.) [with pilot role]

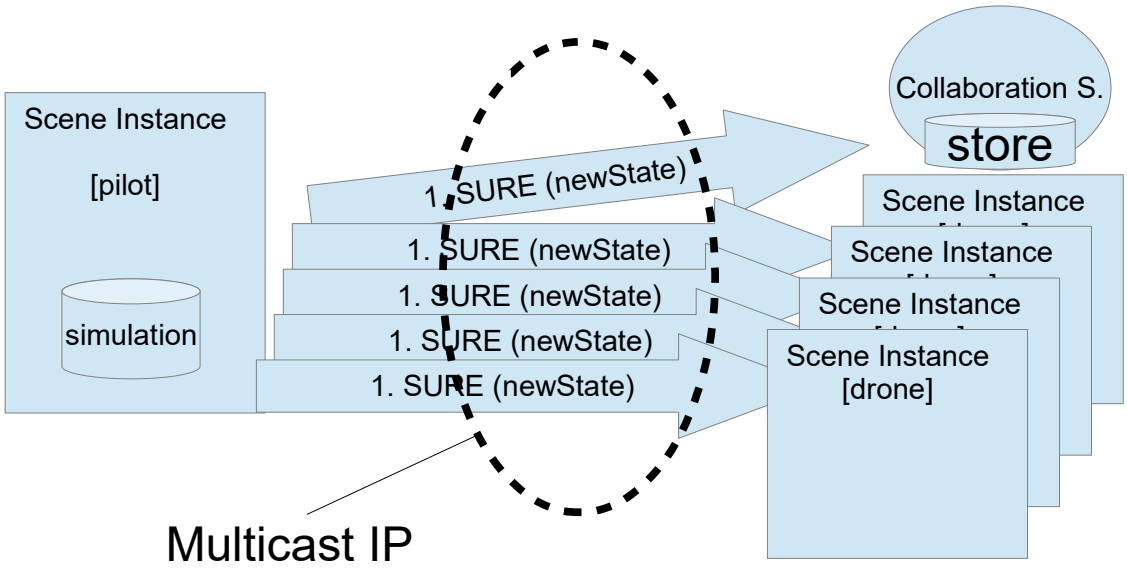

Figure 22: Multicast transport of avatar state (real-time)

3.4.5 Unloading a Model / Querying the Current State

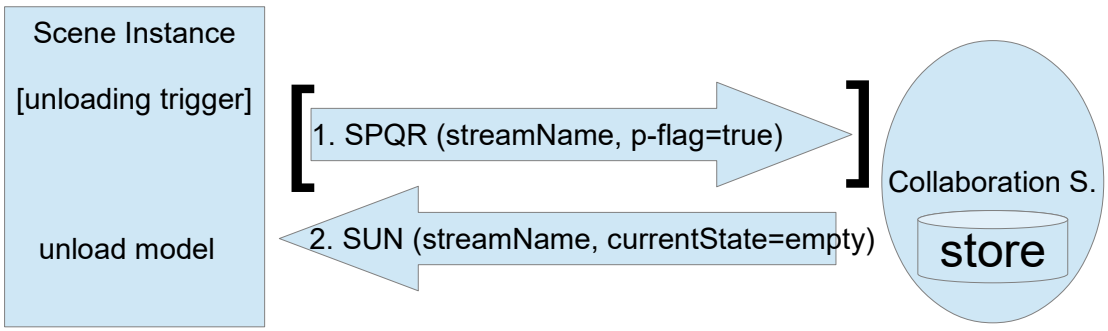

Figure 23: Emptying a state (unloading a model) [with unloading trigger]

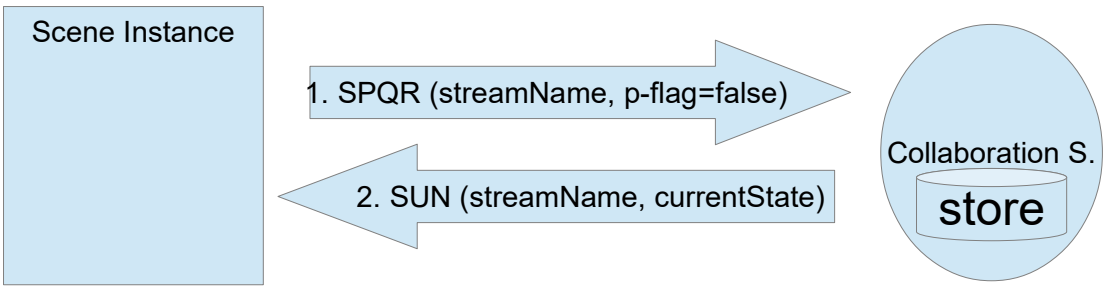

Figure 24: Querying a state by one scene instance (exclusive quest)

3.4.6 Server Side Calculations (UC 3.c.)

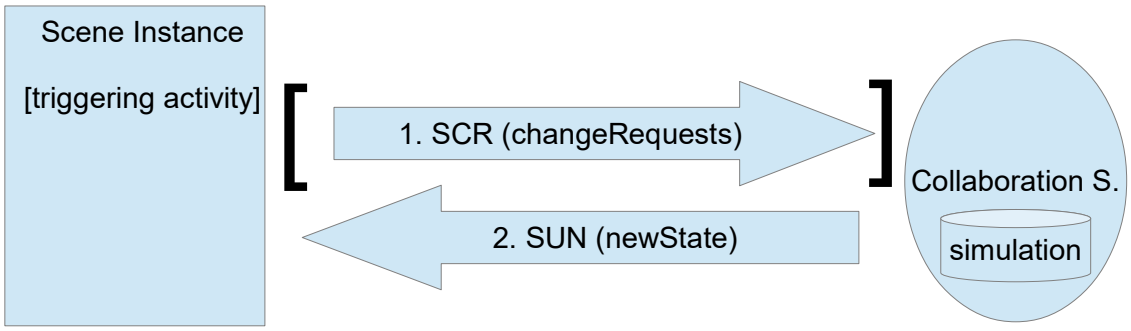

Figure 25: Server side calculations (UC 3.c.) [with triggering activity role]

3.4.7 Client Side Calculations (UC 3.d.)

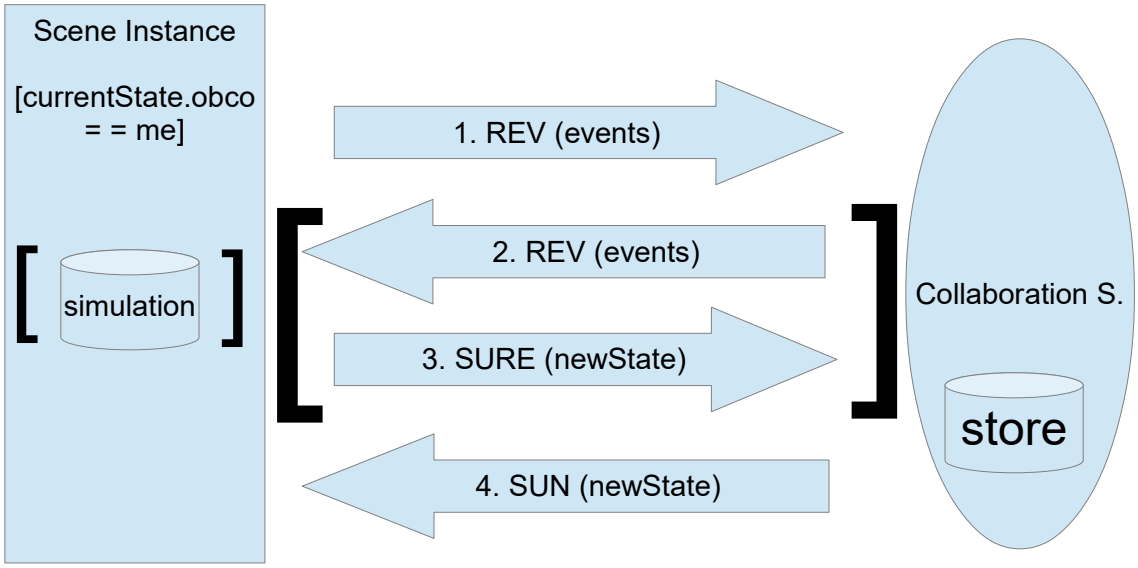

Figure 26: Client side calculation (UC 3.d.) [with controller role]

3.4.8 Taking the Controller Role

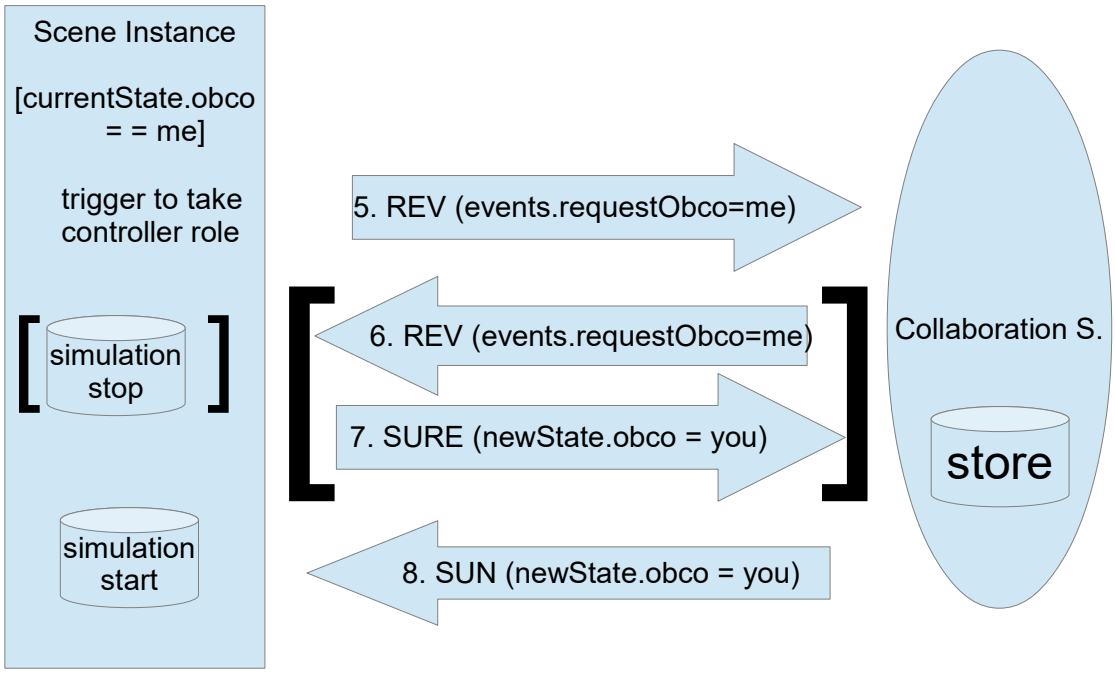

Figure 27: Taking the Controller Role (Obco) [with old controller role]

After getting the controller role, the new OBCO (Re-)Sets the state to be sure all participants are up to date. See Figure 20 in chapter 3.4.4 .

3.5 How to Solve Inconsistencies in NSN Field Names

The names of the arbitrarily defined fields in the NSNs need not be identical among all scene instances. E.g. during an activity of upgrading a game on severel clients, the versions of the game might be different on several scene instances and hence the field names could be inconsistent.

Nevertheless, different versions of a game should be able to interoperate, of course.

Rules of Thumb

1. Network Events
   (a) The Collaboration Server does not store information about any event. Events in a BEV or REV PDU are just passed through or routed, respectively. They are identified by <streamName> + <sensorId> + <fieldname>[2]. Their <fieldvalue> and <fieldtype> is transmitted, too.
   (b) If a scene instance receives a network event for a field <streamName> + <sensorId> + <fieldname> that is not defined in the scene instance, then the received network event shall be silently discarded. A log may be written
   (c) "changeRequests" in an SCR PDU are handled like "newState" in SURE with the only difference, they are identified by <streamName> + <sensorId> + <fieldName> + <operation>. The <fieldvalue> and <fieldtype> describe the prameter of the operation.
2. Network States (Shared States)
   (a) The Collaboration Server stores a template of states (TOS) for each known combination of <streamName> + <sensorId> within the scope of each multiuser session.
   (b) The STS PDU transmits a local TOS for each NSN of a stream (identified by <streamName> + <sensorId>).
   (c) This local TOS is created by the NSN by combinig the <fieldname> and the <fieldtype> of each "<fieldname>_changed" field of the NSN into a row of a local TOS table, which is sent with the STS PDU
   (d) When receiving the template from the STS PDU, the Collaboration Server adds and/or replaces rows in its persistently stored "global" TOSs for the <streamName> in question. The STS can never trigger deletion of rows from the global TOSs.
   (e) When creating a new row in a TOS (upon STS), then the value of the state is initialized with <null> at the Collaboration Server.
   (f) When purging a stream with the SPQR (p-flag=true), then all rows of all templates for that stream (i.e. for all <sensorIds> of that <streamName>) are deleted
   (g) The "currentState" – which is always transmitted in a SUN PDU – may have been triggered by an STS or by an SPQR. It always contains ALL rows according to the global TOSs at the Collaboration Server, in particular some of the states might have the value <null>. After a purge the number of rows will be zero.
   (h) The "newState" never transmits states of value <null>. It contains a subset of all states related to all rows of the local or global TOSs. It is transmitted in SURE or SUN PDUs.

2 Note: if an event is identfied by the fields "evt_touched", "touched_evt", "ff_touched" or "touched_ff" then the <fieldname> is "touched". The <fieldnames> "requestObco" and "obco" are reserved for events and states, resp.