Simulated Railroad Framework, http://simulrr.sourceforge.net
Synopsis: 000_Synopsis

This file valid for step 0033.10
Issue Date: 2017-05-06

The Simple Scene Controller

1 Synopsis

The Simple Scene Controller is the part of the SRR/SMUOS Framework that is intended to be used in the frame of an SrrTrains layout / Simple Multiuser Scene.

The present paper describes the "SSC Base" and the "Beamer Manager" extension and the "Key Manager" extension of the Simple Scene Controller.

The "Base" of the Simple Scene Controller consists of two X3D prototypes, which are contained in the files SscBase.x3d ("client" and "server") and SscBaseNs.x3d ("network sensor").

The "Beamer Manager" extension consists of an X3D prototype, which is contained in the file SscBeamerManager.x3d ("client").

The "Key Manager" extension consists of two X3D prototypes, which are contained in the files SscKeyManager.x3d ("client" and "server") and SscKeyManagerNs.x3d ("network sensor").

Additionally, all parts of the SSC use the "SSC Core" X3D prototype, which is contained in the file SscCore.x3d and all parts of the SSC may use the "SSC Dispatcher", which consists of two X3D prototypes, contained in the files SscDispatcher.x3d ("client" and "server") and SscDispatcherNs.x3d ("network sensor").

MIDAS Objects and all parts of the SSC may use the "SSC Dispatcher Stub", which is contained in the file SscDispatcherStub.x3d ("server").

2 The Purpose of the Simple Scene Controller

The Simple Scene Controller helps with the coordination of the SMUOS Framework.

The Simple Scene Controller provides an interface to
- initialize the SMUOS Framework in multi-user-mode or in single-user-mode
- add/remove avatars
- report avatar position/orientation to the frame in relative coordinates
- request the controller role
- register/deregister modules explicitely
- load/unload dynamic modules
- activate/deactivate modules
- request MOC roles
- set trace levels and output tracer output
- display carried keys **)
- put carried keys into key containers or locks **)
- reset all keys **)
- send/receive console commands/responses
- display all beamer destinations *)
- bind one beamer destination *)
- join a remote user
- output short status messages

*) realized by the SSC "Beamer Manager" SscBeamerManager.x3d
**) realized by the SSC "Key Manager" SscKeyManager.x3d

3 External View

When the <ProtoInstance> "SscBase" and all <ProtoInstances> of the SSC Extensions have been loaded, then the Simple Scene Controller is in mode of operation "LOADED" (MOO "LOADED").

After some preparation, the frame sends the init=true event to the SSC Base and triggers the initialization of the Simple Scene Controller.

After successful or unsuccessful initialization the SSC Base outputs the SFNode event "commParam", that points to the common parameters. The common parameters are hold as fields of a <Script> node, which is contained in the SSC Base. They are described in chapter 5.1.

The "commParam" event can be immediately <ROUTE>d to the module coordinators of all static modules (via interface miModule) to trigger their initialization.

If a frame contains some software to load/unload dynamic modules, then it must store the "commParam" pointer for the later initialization of the dynamic modules (that will be performed, after a module will have been loaded).

Many use cases of the Simple Scene Controller are available only after the Simple Scene Controller has been successfully initialized.

3.1 MOO Diagram (MOOD)

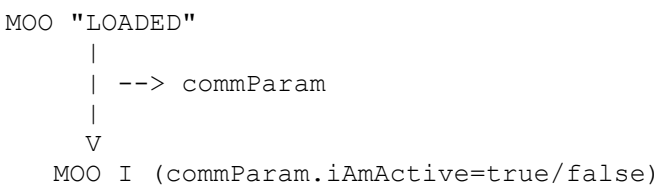

Figure 1: State Diagram of the Simple Scene Controller

3.2 State Event Matrix

Following events in following states lead to execution of following use cases:

Event \ State	MOO "LOADED"	MOO I (activated)	MOO I (inactive)
multiuserRequest	Preparation	-	-
sessionId	Preparation	-	-
useConnection	Preparation	-	-
sscExtensions	Preparation	-	-
init	Initialization	-	-
sessionIdLeft	-	RemoteUserLeft	-
addAvatar	AddAvatar	AddAvatar	-
removeAvatar	RemoveAvatar	RemoveAvatar	-
controllerRequest	RequestController	RequestController	-
registerModules	-	DynModRegistration	-
deregisterModules	-	DynModRegistration	-
activateRequest	-	ModuleActivity	-
deactivateRequest	-	ModuleActivity	-
mocRequest	-	ModuleActivity	-
traceLevel*Request	SetTraceLevels	SetTraceLevels	-
putKeys **)	-	CarriedKeysSupport	-
resetKeys **)	-	CarriedKeysSupport	-
consoleCommand	-	ConsoleInterface	-
bindBeamerDestination *)		Beamer	-
joinAvatar		Beamer	-

Table 1: State Event Matrix of the Simple Scene Controller

3.3 Use Cases

The State Event Matrix displays, which use case is available via which field of the prototypes SscBase, SscKeyManager and SscBeamerManager in which mode of operation (MOO), either MOO "LOADED" or MOO I (initialized). An additional differentiation is done based on commParam.iAmActive (activated/inactive).

The following sections explain each use case in detail

3.3.1 Preparation

Before the SSC is initialized, the following fields must be set:

- mandatorySscBaseExtensions (MFNode)
- optionalSscBaseExtensions (MFNode)
  The frame instantiates all SSC extensions (see 051_Extensibility for an explanation of SSC extensions) and tells the SSC Base their addresses so that he can initialize them

In case of multi-user-mode, following fields must be set additionally:

- multiuserRequest
  set to TRUE to switch to multi-user-mode
- sessionId
  non-negative integer to identify the scene instance within the multiuser session (in future releases, the value "0" might have a special meaning)
- useConnection
  value to refer to the <NetConnection> node of BS Contact, needed by the Event Stream Sensors

3.3.2 Initialization

With an event init=true, the frame triggers the initialization of the SSC Base and of all SSC extensions that are referenced by the fields "mandatorySscBaseExtensions" and "optionalSscBaseExtensions".

The SSC Base will answer with an event commParam != null, where the field commParam.iAmActive (SFBool) will indicate whether the initialization was successful or not.

"commParam" can be directly routed to the miModule interfaces of the top level modules and to the uiObj interface of astral objects.

3.3.3 RemoteUserLeft

With an event sessionIdLeft = <sessionId> the frame informs the SSC that a remote user has left the multiuser session.

The SSC will update the commState and re-arrange the controller role (if necessary).

3.3.4 AddAvatar/RemoveAvatar

Usually the multiuser system informs the frame, when a user has joined/has left the multiuser session.

This information is used by the frame to load/unload avatars and to forward their addresses to the SSC in order to keep the avatar containers up to date (please refer to 351_AvatarContainer for a description of avatar containers).

The events "addAvatar" and "removeAvatar" are also acceptable, when the SSC has not yet been initialized.

A remote user can be joined by entering the new position/orientation ("joined Position", "joinedOrientation") and afterwards entering the sessionId of the remote user ("joinAvatar").

3.3.5 RequestController

With an event controllerRequest = true the frame can request the controller role for his scene instance.

3.3.6 DynModRegistration

The frame can explicitely register/deregister dynamic modules with the help of the fields "registerModules" and "deregisterModules".

"registerModules" will only allocate a moduleIx (that will be used in all scene instances for this module name). The actual loading and initialization of the module shall be performed by the frame.

A dynamic module may be loaded and initialized without explicit registration, an implicit registration will be done during module initialization (as it is done for static modules). In this case, the frame does not know the moduleIx prior to module initialization but only afterwards.

"deregisterModules" will deregister the module and detach the module coordinator in all scene instances. The SSC will inform the frame about the deregistration of the modules ("registeredModules" field).

The actual unloading of the module has then to be performed by the frame.

DEREGISTERING STATIC MODULES (THAT CANNOT BE UNLOADED) IS BAD PRACTISE. The SRR/SMUOS Framework cannot distinguish static from dynamic modules.

A frame can locally unload one instance of a dynamic module (refer to chapter "Use Case DisableModule" in 201_ModuleCoordinator).

3.3.7 ModuleActivity

Module activity is usually changed by the module itself. To achieve this, the module sends an event to the external interface uiMod of the module coordinator.

Please refer to chapter "Use Case ModuleActivity" in 201_ModuleCoordinator for a short introduction to module activity.

Additionally, the frame has the possibility to influence module activity via the fields "activateRequest", "deactivateRequest" and "mocRequest" of the external interface uiControl of the Simple Scene Controller.

3.3.8 SetTraceLevels

The frame can set all trace levels of his scene instance. Please refer to 015_Tracer for a more detailed description.

3.3.9 CarriedKeysSupport **)

Avatars (users) can carry keys and keys can be contained in key containers, locks react on carried keys and on contained keys (see also 353_KeyContainer, 354_CarriedKeysLock and 358_ContainedKeyLock).
This functionality is supported by the "Key Manager" extension of the SSC. Hence the SSC provides the fields "carriedKeys" (MFString) to display the carried keys in the frame and "putKeys" (MFString) to put keys into the bound key container. With "resetKeys" (SFBool), the frame can reset all keys (carried keys and key containers).

3.3.10 ConsoleInterface

The frame can request the handling of a console command (an SFString value, i.e. a command line of a given syntax).
Each console command will be answered with exactly one console response (an MFString value, i.e. a list of response lines), unless the console is already active (in which case the console command will be silently discarded).
The syntax of the console response is kept as simple as possible, hence easing the automatic handling by some GUI controller (MVC pattern).
Please refer to 014_ConsoleInterface for more details.

3.3.11 Beamer *)

The MIDAS Object "Beamer Destination" (see 357_BeamerDestination) can register at the "Beamer Manager" extension of the SSC. All beamer destinations are presented to the frame via the "beamerDestinations" (MFString) field at the uiControl interface and can be bound via the "bindBeamerDestination" (SFString) field. Beamer destinations can be bound via the "Beamer" MIDAS Object, too (see 356_Beamer).

Note: this function is also known as "gate".

3.3.12 StatusMessage

The Simple Scene Controller outputs short status messages via the "statusMessage" field.

4 Internal View

This chapter shows only information about the SSC Base. If you seek more information about the Beamer Manager extension or about the Key Manager extension, please refer to general information about SMUOS extensions in 051_Extensibility.

4.1 The Structure of the SSC Base

The SSC Base is an X3D prototype, which contains following nodes:

- 3 <ProtoInstance> nodes of the prototype "SmsTracer", one for the "SSC Base Client", one for the "SSC Base Server" and a "#parm Object"
- a <Script> node, which holds the "Common Parameters" and the "Classic Tracer" – see chapter 5.1.
- a <Viewpoint> (the "Internal Default Viewpoint") and the associated avatar container ("Default Avatar Container")
- a <ProtoInstance>, which adapts a native network sensor for usage within the SSC Base – see chapter 4.2
- a <ProtoInstance> of type "SscCore", which provides core functions
- a <Script> node to store the received and decoded "Communication State" (which was sent as an MFString value via the network sensor), until it has been processed completely.
- several <TimeSensor> nodes to realize timers and periodic triggers
- a <Script> node, which contains the software of the "SSC Base Client"
- a <Script> node, which contains the software of the "SSC Base Server"

The "SSC Base Server" runs in only one of the scene instances of a multiuser session (this scene instance is said to have the "controller role" then).

The "SSC Base Client" runs in all scene instances (including the one with the controller role).

The "SSC Base Server" maintains the "Communication State" commState (a rather complex MFString value) and distributes it via the network sensor after each change, so that all scene instances are kept up to date.

If an "SSC Base Client" wants to request a change of the commState, then it sends a CSCR (commState Change Request) to the "SSC Base Server".

The "SSC Base Client" maintains the "Common Parameters" commParam (an SFNode value that points to a script node within the SSC Base), whose fields can be accessed by all parts of the SRR/SMUOS Framework in the same scene instance.

The "Common Parameters" contain some information, which is relevant for all parts of the SRR/SMUOS Framework within the scope of one scene instance. Besides others this is a local copy of the "Communication State" commState – see chapter 5.1.

4.1.1 The SSC Dispatchers

The SSC physically hosts a list of SSC dispatchers to support the console interface and to support universal object classes for unbound models and SSC Parameters.

The SSC dispatchers themselves contain "client" and "server" software, too.

However, SSC dispatchers do not only exist once per scene instance (as the SSC does), but once for each universal object class and once for each module.

The first n + 1 SSC dispatchers (where n is the number of UOCs) are
- one dispatcher in the SSC Base
- n dispatchers, one for each UOC, where each SSC Extension may establish zero or more UOCs,

each one identified by a UOC name (SSC Base uses the reserved UOC name "Base").

After the first n + 1 SSC dispatchers, then follows a variable list of SSC dispatchers
- one for each registered module

each one identified by a module name.

The SSC dispatcher for the SSC Base is physically hosted by an "SFNode" field of the "SSC Base Client".

The SSC dispatchers for the SSC extensions are hosted by the prototypes of the SSC extensions.

The SSC dispatchers for the modules are physically hosted by an "MFNode" field of the "Common Parameters" script. Hence the MIDAS Objects can access the SSC dispatchers via the "Module Parameters" modParam.

Having dynamic modules (that can be loaded/unloaded at runtime) is the reason for having the SSC dispatchers in the SSC (which exists from load time until shutdown of the scene). The console interface of a scene instance must be able to access a module, even if this module is currently unloaded (the SSC dispatcher will forward the console command to the scene instance that has the MOC role for this module).

4.2 The Network Sensor Prototype

The Network Sensor Prototype SscBaseNs.x3d provides a generic interface as a facade for different technologies to access the services of a central Collaboration Server (*currently only implemented for BS Contact/BS Collaborate*).

In single-user-mode, the Network Sensor Prototype provides a "short-cut", so that the SRR/SMUOS Framework is functional without CS, too (single-user-mode is currently tested and supported for BS Contact).

Following messages are exchanged via the Network Sensor Prototype of the SSC Base:

- "Access Request" (Event, SFInt32)
  During initialization, a scene instance sends an Access Request with the own sessionId to the central controller to create an entry in the commState. The scene instance waits for the answer commState, before it sends the first CSCR (see Change Requests).

- "Change Requests" (Event, MFString)
  Each scene instance can request changes of the commState (commState change requests, i.e. CSCRs) from the central controller.

- "Communication State" (State, MFString)
  The Communication State (commState) is maintained and distributed by the central controller.

- "Remove Sessions Request" (Event, MFInt32)
  If a sessionId has left the multi user session, this event will be sent to the central controller to remove the sessionId from the commState.

Since the network sensors do not currently provide addressing services, but do distribute events to all scene instances evenly, the Network Sensor Prototype provides a filter.

This filter reports the events "Access Request", "Change Requests" and "Remove Sessions Request" to the user (i.e. to the "SSC Base Server") only, when the flag commParam.iAmController is set (actually, this flag makes a scene instance being the server).

4.3 Parameters in the Communication State (commState)

- sequence
  The commState includes a sequence number that is increased with sending out of each commState. This helps in waiting for the reception of the commState (the central controller waits with sending out a new commState, until it has received the latest commState sent out).

- Access Request Answer Flags
  Each commState contains a hint, which Access Requests are answered with this commState. This helps in detecting the "initialization finished" condition.

- Avatar Container Binding Answer Flags
  see concept paper 351_AvatarContainer

- sessionIds
  A list of the sessionIds of all scene instances

- avaConBinding
  see concept paper 351_AvatarContainer

- controllerIx
  An index, that indicates, which sessionId is the one of the central controller

4.4 Communication State Change Requests (CSCRs)

When the central controller has received CSCRs, he reads the commState from the commParam, applies the CSCRs to it and sends out a new commState.

Storing the commState in the commParam will be done by the client software.

Following basic CSCRs are currently understood by the central communication controller.

- addSession;<sessionId>
  creates a new entry <sessionId> in the commState and initializes all parameters for this scene instance

- removeSession;<sessionId>
  removes the scene instance <sessionId> from the commState

- takeController
  requests the "provisional" controller to take the controller role

- grantController;<sessionId>
  grants the controller role to scene instance <sessionId>

- bindAvatar;<extObjId>;<sessionId>
  see concept paper 351_AvatarContainer

5 Additional Info

5.1 The Common Parameters

The term "common parameters" refers to a <Script> node, which is a part of the SscBase prototype and which holds in its fields a lot of common data, that is accessible by all parts of the scene (especially by all parts of the SRR/SMUOS Framework).

During initialization, the SFNode event "commParam" is forwarded from the Simple Scene Controller to the Module Coordinators and to the astral objects.

5.1.1 Core Information

These four fields are set by the SscCore prototype.

enabled(SFBool).....remains true, as long as the SSC Base is enabled
initialized(SFBool).....set by SSC Core after initialization of the SSC. A value "true" indicates successful initialization of the SSC Base and of all mandatory SSC Extensions. Now the field "extensions" contains pointers to all successfully initialized SSC Extensions and the SSC Base can send an Access Request to the Central Controller
iAmActive(SFBool).....set by SSC Core after activation. A value "true" indicates successful activation and initialization of the SSC Base (now all common parameters are valid). Many of the functions of the SSC Base are only available, when iAmActive = "true"
extensions(MFNode).....Pointers to all successfully initialized extensions of the Common Parameters (please refer to 051_Extensibility)

5.1.2 Static Information

The common parameters contain besides others some static information, which does not change during the lifetime of the scene.

```
sscBaseVersion(SFFloat).........version of the SSC Base prototype
neededBrowsers(MFString)........Browser.name of all supported browsers
neededVersions(MFFloat).........needed version of all supported browsers
blaxxunIx(SFInt32)..............index of blaxxunCC3D within "neededBrowsers" and
                                "neededVersions"
octagaIx(SFInt32)...............index of Octaga Player within "neededBrowsers"
                                and "neededVersions"
instantIx(SFInt32)..............index of Avalon within "neededBrowsers" and
                                "neededVersions"
errorStrings(MFString)..........errorStrings[errorNo] is the last occured error
```

5.1.3 Networking Parameters

These parameters are accessible as global SSC parameters

```
maxRTT(SFFloat).................network parameter: maximum roundtrip time
randomFactor(SFFloat)...........another network parameter
```

5.1.4 Status Information

Following common parameters indicate basic status about the SSC Base.

```
browserIx(SFInt32)..............set during initialization, either blaxxunIx or
                                octagaIx or instantIx
browserVersion(SFFloat).........set during initialization, actual version of
                                used browser
useConnection(SFString).........DEF name of the <NetConnection> (BS Contact),
                                needed by the network sensors
connection(SFNode)..............Pointer to the <Connection> (Octaga), needed by
                                the network sensors
errorNo(SFInt32)................last occured error. Initialized to 0. The text
                                of the last occured error can be accessed by
                                errorStrings[errorNo]
multiuser(SFBool)...............true, if multiuser mode is actually chosen
```

5.1.5 Local Scene Information

The local scene information is derived from the commState. So the source is the reception of a "communication state" (commState) – see below.

```
ownSessionId(SFInt32)...........the sessionId, which was delivered by the
                                <BSCollaborate> node, "-1" in case of single-
                                user-mode
attachedModules(SFString).......a comma-separated list of all modules that
                                are currently attached in this scene instance
activeModules(SFString).........a comma-separated list of all modules that
                                are currently active in this scene instance
attachedModulesBoolArr(MFInt32).a list of quasi-boolean values ("0" or "1"),
                                i.e., if attachedModulesBoolArr[moduleIx] then
                                this module is attached in this scene instance
activatedModulesBoolArr(MFInt32).a list of quasi-boolean values ("0" or "1"),
                                i.e., if activatedModuleBoolArr[moduleIx] then
                                this module is active in this scene instance
mocRolesBoolArr(MFInt32)........a list of quasi-boolean values ("0" or "1"),
                                i.e., if mocRolesBoolArr[moduleIx] then this
                                module has its MOC role in this scene instance
ownMocRoles(SFString)...........a comma-separated list of module names that
                                have their MOC role in this scene instance
```

5.1.6 Communication State

The communication state is stored on the collaboration server and distributed to all scene instances as a rather complicated MFString state value, whose reception should be performed as an atomic action.

The communication state is stored in the common parameters in two sections, first the "Master Communication State", i.e. the parameters, which contain all the information, and which can be transformed 1:1 to the MFString value; second, the "Communication State Helpers", which contain redundant information, which is derived from the master communication state in each scene instance separately

5.1.6.1 Master Communication State

commStateSequence(SFInt32).....sequence number of the commState message
sessionIds(MFInt32).............ordered list of all sessionIds of all scene instances
avaConBinding(MFString).........has same dimension as sessionIds, each element contains the extObjId of the bound avatar container of that scene instance or an empty str.
controllerIx(SFInt32)...........sessionIds[controllerIx] has the central controller role
ownIx(SFInt32)..................sessionIds[ownIx] is the same as ownSessionId
moduleActivity(MFInt32).........module activity matrix
modulesAttachedState(MFInt32)...module attachment matrix
unregisteredModules(MFInt32)....all free moduleIxs
registeredModules(MFString).....module names of all registered modules (can be empty string - gaps are possible)
implicitelyRegistered(MFInt32)..."1", if module is implicitely registered

5.1.6.2 Communication State Helpers

iAmController(SFBool)...........if "true", then this scene instance has the central controller role
attachedModulesStrings(MFString)..has same dimension as sessionIds, each element contains a comma-separated list of attached modules of the addressed scene instance
activatedModulesStrings(MFString).has same dimension as sessionIds, each element contains a comma-separated list of activated modules of the addressed scene instance
mocRolesStrings(MFString).......has same dimension as sessionIds, each element contains a comma-separated list of MOC roles of the addressed scene instance

5.1.7 List of sessionIds of Sessions that Left During the Last 20 Seconds

With the help of the following parameters, the SSC Base maintains a list of sessionIds of sessions that have left during the last 20 seconds. This list is used by the MIDAS Base to perform a correct handling of assignment of OBCO roles.

leftSessionIds(MFInt32).........List of sessionIds that left during the last 20 seconds
leftSessionIdsTimestamps(MFTime)..has the same dimension as leftSessionIds. Keeps the associated timer values

5.1.8 SSC Dispatchers

sscDispatchersGroup(MFNode).....to be described
modulesOffset(SFInt32).........to be described

5.1.9 Address of the SSC Base

sscBase(SFNode).....Address of the SSC Base Client <Script>.
If a module coordinator [extension module] or a MIDAS Object wants to send an event to the SSC Base Client, then it can address the input fields of the SSC Base Client via [modParam.]commParam.sscBase.<fieldName>

3.1.10 Distribute Commands to all Avatar Containers

setPosOriReporting(SFString).....to be described
joinedPosition(SFVec3f).....to be described
joinedOrientation(SFRotation).....to be described
joinAvatar(SFInt32).....to be described

3.1.11 Tracer Support

The basic support of the Tracer is implemented directly in the common parameters.

Where we have to distinguish
- the "classic" tracer (deprecated but still supported)
- the SMS Tracer

The "classic tracer" can be directly accessed via the common parameters, without additional necessities, on the other hand it supports only one trace level, which is common to all parts of the scene.

The "SMS Tracer" is more sophisticated. It is implemented in the X3D prototype SmsTracer.x3d, but it has to be additionally supported by the common parameters.

traceLevel(SFInt32).....to be described
traceLevelSscBase(MFInt32) .....to be described
traceLevelCommControl(MFInt32)....to be described
traceLevelModules(MFString).....to be described
traceLevelObjects(MFString).....to be described
command(SFNode).....to be described
command.ErrLog(MFString).....to be described
command.InfoLog(MFString).....to be described
command.Debug(MFString).....to be described
command.ErrLogNew(MFString).....to be described
command.InfoLogNew(MFString).....to be described
command.DebugNew(MFString).....to be described
command.ErrLogNew2(MFString).....to be described
command.traceOutput(MFString).....to be described
command.traceBuffer(MFString).....to be described