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

This file valid for step 0033.10
Issue Date: 2017-03-17

MIDAS Objects

1 Synopsis

The MIDAS Objects are the parts of the SRR/SMUOS Framework that are intended to be used in the models of SrrTrains layouts / Simple Multiuser Scenes.

A lot of basic information about models and objects is already available in the paper 013_ModelsAndObjects.

The SRR/SMUOS Framework is accompanied by a set of example MIDAS Objects. Some of them can be used without a SMUOS extension (so-called "basic" MIDAS Objects), some of them need one of the example SMUOS extensions ("Beamer Manager", "Key Manager", "Train Manager").

MIDAS Objects that need the "Train Manager" extension, are also called "SRR Objects".

2 The Purpose of the MIDAS Objects

MIDAS Objects provide MU capable interactive simulation/animation services that can be used by model authors to provide animated, interactive, MU capable and interoperable VR/AR models, which may render real-life objects (RLOs).

All the MU stuff is encapsulated, only the necessary eventOuts, eventIns and fields are visible at the user interface of the MIDAS Objects, hiding the differences between single-user scenes and multi-user scenes.

MIDAS Objects do NOT create any renderable graphics NOR do they create sounds. They are just the "invisible engines" of the models, which calculate quantities.

The VRML/X3D paradigm for animation and simulation, i.e. concatenating

    EnvironmentSensor --> TimeSensor --> Interpolator --> Target Node

is enhanced/modified by replacing the Time Sensor by a MIDAS Object:

    EnvironmentSensor --> MIDAS Object --> Interpolator --> Target Node

and by encapsulating all these nodes into a "model".

2.1 An Example

Let's have a look at the "binary switch" MIDAS Object. That MIDAS Object is virtually the simplest of all MIDAS Objects, while it shows up all essential topics that make up a MIDAS Object:

- client software, running once in each of 1 to N scene instances
- server software, running once in only one dedicated scene instance
- a global state, maintained by the server software and distributed to all clients upon each change
- a local reaction in each client, when the global state has been received
- each client can send change requests to the server in order to change the global state
- a network sensor, which connects clients and server (via a Collaboration Server)

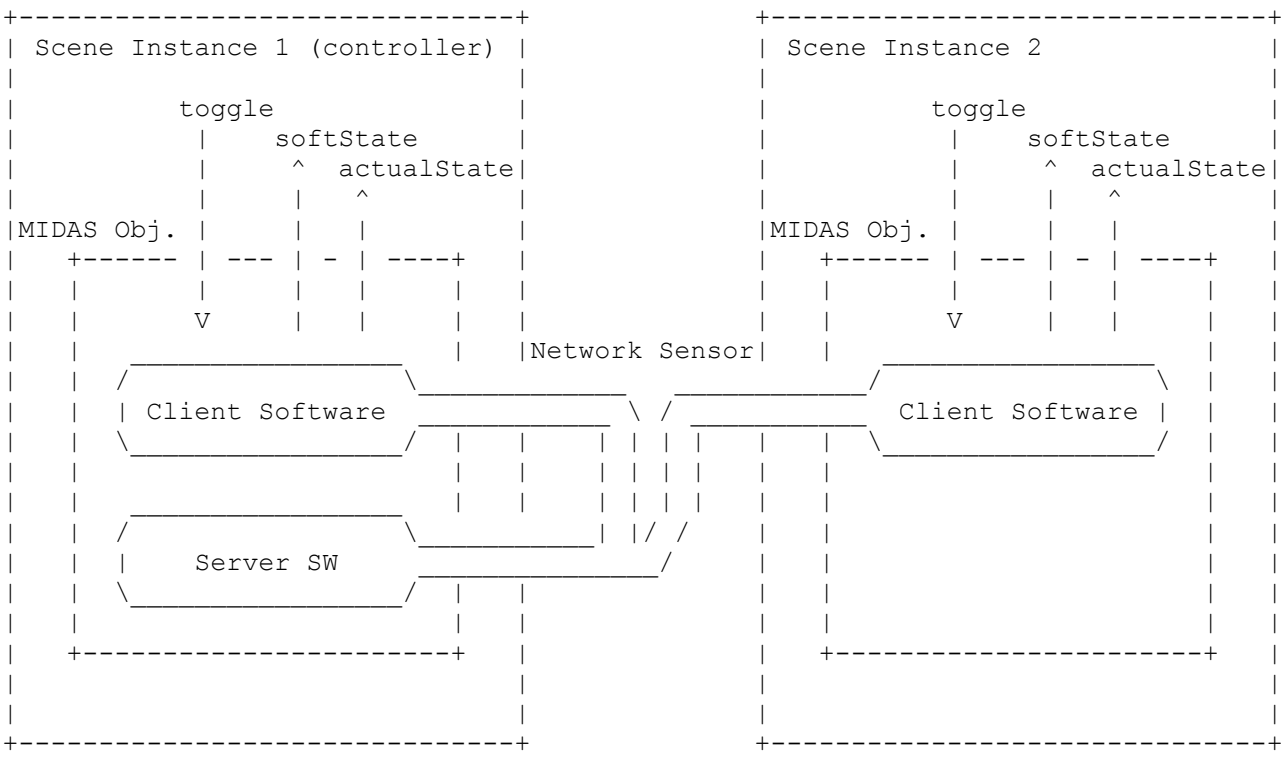

Figure 1: an example MIDAS Object in an example MU session ("binary switch")

The following figure shows an example timeline of the parameters of a "binary switch" MIDAS Object in 2 scene instances:

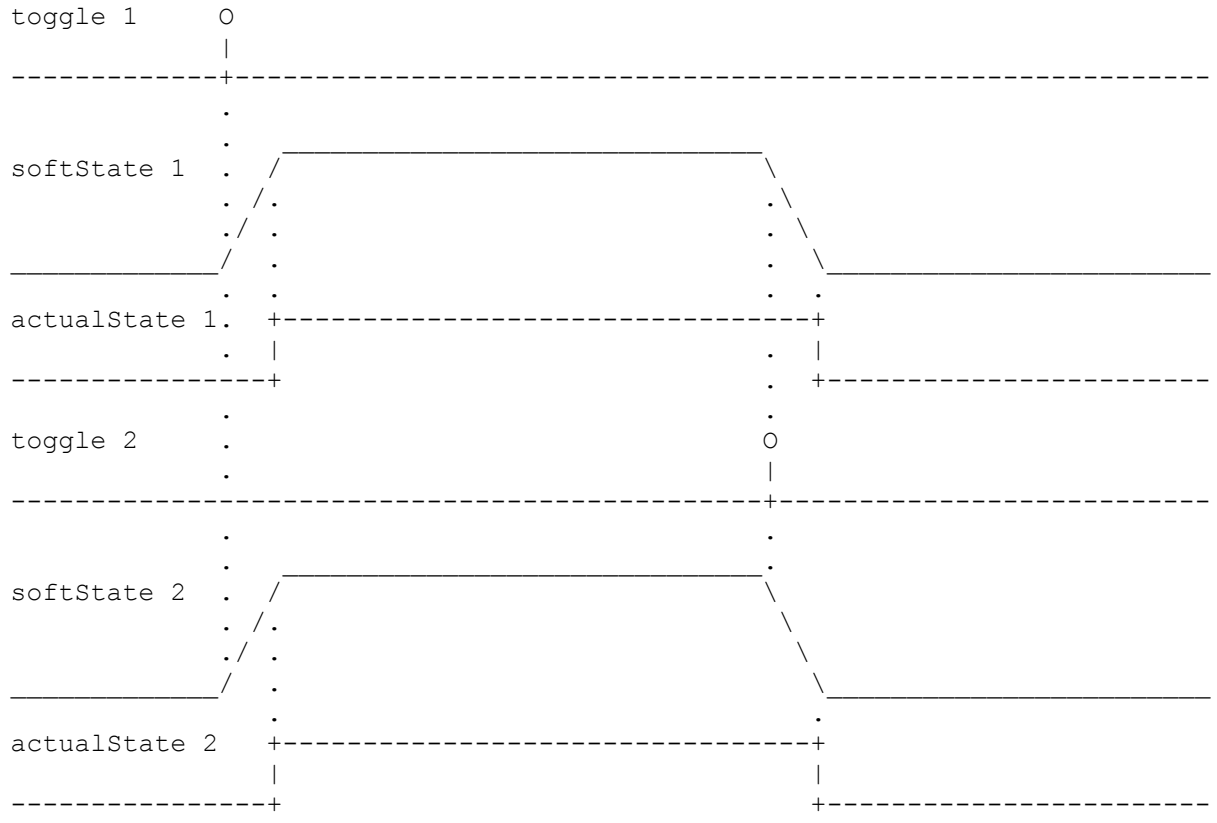

Figure 2: Example timeline of "binary switch" (network delays neglected)

3 External View

A MIDAS Object exposes its external interface (user interface) "uiObj" to the surrounding model, module or frame.

Some fields of this interface are common to all MIDAS Objects. That fields are described in chapter 5 of the paper 013_ModelsAndObjects:

- objId (SFString)
- universalObjectClass (SFNode) – for unbound objects (*not yet implemented*)
- parentObj (SFNode)
- commParam (SFNode)
- modParam (SFNode)
- enabledOut (SFBool)
- initialized (SFNode) – for unbound objects (*not yet implemented*)
- attached (SFNode) – for unbound objects (*not yet implemented*)
- disable (SFTime) – for unbound objects (*not yet implemented*)

Other fields are specifically designed for each MIDAS Object.

Additionally, each MIDAS Object listens to fields of the module parameters

- modParam.activate (SFBool) – module coordinator reports activation
- modParam.deactivate (SFBool) – module coordinator reports deactivation
- modParam.takeMOC (SFBool) – module coordinator reports having got MOC role
- modParam.grantMOC (SFBool) – module coordinator reports having lost MOC r.
- modParam.disable (SFBool) – module coordinator reports disabling module
- modParam.sessionIdLeft (SFInt32) – a remote session has left the game
- modParam.sessionIds (MFInt32) – all sessions, in which the module is active

that propagate central information to all MIDAS Objects of a module.

3.1 MOO Diagram (MOOD)

After the surrounding module or model of a MIDAS Object has been loaded (in the case of bound or unbound MIDAS Objects) or after the frame has been loaded (in the case of astral MIDAS Objects) and hence the MIDAS Object has been loaded, it is in the mode of operation (MOO) "LOADED".

Before first initialization of the MIDAS Object with the module parameters "modParam" or with the common parameters "commParam", the user does some preparation (see below) and herewith assigns an object ID to the MIDAS Object.

If the user assigns a "universalObjectClass" to the object in MOO "LOADED", then the object becomes an unbound object.

Otherwise, the object will become an astral object or a bound object, depending on whether it will be initialized with commParam or with modParam, respectively.

So the input fields "commParam", "modParam" and "disable" trigger what we call "MOO Changes", shortly spoken:

- any object is changed to MOO V with "disable"
- astral objects are changed to MOO I with "commParam"
- bound objects are changed to MOO II with "modParam"
- unbound objects are changed to MOO III with "commParam"
- unbound objects are changed to MOO IV with "modParam"
- if MOO V is reached, any input is ignored (no more MOO change possible)
- unbound objects can change between MOO III and MOO IV arbitrarily

Chapter 5 of the paper 013_ModelsAndObjects holds a comprehensive listing of all possible MOOs and MOO changes for objects.

3.2 State Event Matrix

Following events in following states are described in following use cases:

Event \ State	MOO "LOADED"	MOO I/II/III/IV	MOO V
objId	Preparation	-	-
universalObjectClass	Preparation	-	-
commParam	MOO Changes	MOO Changes	-
modParam	MOO Changes	MOO Changes	-
specific_fields	SpecificUseCases	SpecificUseCases	-
activate	-	ObjectActivity	-
deactivate	-	ObjectActivity	-
takeMOC	-	ObjectActivity	-
grantMOC	-	ObjectActivity	-
sessionIds	-	ObjectActivity	-
sessionIdLeft	-	ObjectActivity	-
disable	DisableObject	DisableObject	-

Table 1: State Event Matrix of any MIDAS Object

3.3 Use Cases

The State Event Matrix displays, which use case is available via which field of the MIDAS Object's prototype in which state.

The execution of use cases will lead to changes in the internal states of the MIDAS Objects.

3.3.1 Preparation

Before first time initialization of a MIDAS Object, the surrounding module/model or the frame sets the fields

- objId
- universalObjectClass (in case of an unbound object - *not yet implemented*)

The "objId" and "universalObjectClass" will not change during the lifetime of a MIDAS object.

3.3.2 MOO Changes

Before a MIDAS Object can deliver valid parameters for animation and interaction, it must be assigned an object ID (see 3.3.1 "Preparation") and it must be initialized and/or attached to a module.

Please find a comprehensive description of all MOOs and MOO changes in chapter 5 of the paper 013_ModelsAndObjects.

3.3.3 ObjectActivity

The flags "iAmActive" and "iAmObCo" are maintained by the MIDAS "base class". They depend on the parent module's activity state in the Module Activity Matrix (MAM).

Depending on the "iAmActive" flag, the MIDAS Object may avoid CPU intensive calculations, when the parent module is inactive. If "iAmActive" is true, then the local state is valid and the MIDAS Object should provide valid parameters for animation, simulation and interactivity.

Depending on the "iAmObCo" flag, the MIDAS Object routes change requests to the OBCO instance ("Object Controller") - all other instances ignore change request events. The OBCO, and only the OBCO, is responsible for calculating and distributing the global state.

3.3.4 DisableObject

Disabling a MIDAS Object can happen under several circumstances

- explicitely disabling via uiObj field "disable" (e.g. in unbound models)
- modParam.disable is/becomes true --> disable MIDAS Object together with the whole module (e.g. in dynamic modules)
- something goes wrong during a MOO Change

In all cases, the MIDAS Object will output an "enabledOut" = false event, which may be <ROUTE>d to other active nodes of the model (to their "enabled" field) in order to disable time sensors, environment sensors and so on.

3.3.5 SpecificUseCases

The use cases specific to each MIDAS Object cannot be described here, as the reader will kindly accept.

3.4 Idle Animation

MIDAS Objects should be written in a way, so that they provide animation parameters already before the first initialization (see chapter 3.3.2), i.e. before they receive the module parameters or common parameters.

This "idle animation" shall enable module/model authors to quickly detect, whether they forgot a <ROUTE> or did some similar mistake, without the necessity of inserting their model or their module into an SMS.

E.g. the "binary switch" MIDAS Object outputs a sawtooth function at the "softState" field, as long as it has not been initialized.

4 Internal View

Some functions are common to all MIDAS Objects and hence should be contained in a "base class" for all MIDAS Objects.

Unfortunately, X3D prototypes are not object oriented and hence do not support inheritance of interfaces and behaviours.

Therefore an X3D prototype "MidasBase" (or similar) is provided by the SMUOS Framework and it is used by (nearly) every MIDAS Object. Inheritance is emulated by containment, i.e. each MIDAS Object contains an instance of the prototype "MidasBase" and propagates a part of its external interface to the own external interface.

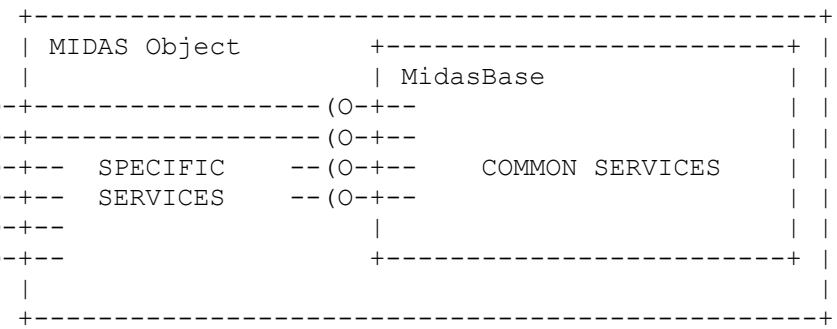

Figure 5: Architecture of a MIDAS Object

THE INTERFACE BETWEEN "MidasBase" and "MIDAS Object" IS EXPERIMENTAL AND SHOULD NOT BE CONSIDERED BEING FUTURE-PROOF.

4.1 Kinds of MIDAS Objects

The SMUOS Framework distinguishes three kinds of MIDAS Objects:

- Standard MIDAS Objects
- "no-state" MIDAS Objects
- "animated" MIDAS Objects

Hence following files contain the MIDAS Base "MIB":

MibStandard.x3d + MibStandardNs.x3d +
+ SscDispatcherStub.x3d + MibStandardOsm.x3d + MibCore.x3d:
= "base class" for standard Objects (see also 311_MidasBase)

MibNoState.x3d
+ SscDispatcherStub.x3d + MibNoStateOsm.x3d + MibCore.x3d:
= "base class" for "no-state" Objects (see also 312_MidasBaseNoState)

MibAnim.x3d + MibStandardNs.x3d + MibAnimNs.x3d
+ SscDispatcherSub.x3d + MibStandardOsm.x3d + MibCore.x3d:
= "base class" for "animated" Objects (see also 313_MidasBaseAnim)

4.1.1 Standard MIDAS Objects

Standard MIDAS Objects were the first MIDAS Objects implemented. It is the purpose of standard MIDAS Objects to maintain a "global state" that can change in a discrete way. That means, the global state will stay the same most of the time (most of the frames) and it will change only seldom at discrete instances of time.

Two examples are the "binary switch" and the "key container" MIDAS Objects. The global state of the "binary switch" is the "scheduled state" (an SFBool value) and the global state of the "key container" are the "contained keys" (MFString).

Standard MIDAS Objects define a so-called "OBCO role" (object controller role). The OBCO role roughly follows the MOC role of the parent module, so that it is guaranteed only one instance of the MIDAS Object has the OBCO role at a time (on the other hand, it may occasionally happen, that a MIDAS Object has no OBCO for some small time interval).

The OBCO maintains the "global state" of the MIDAS Object, which is distributed to all scene instances upon each change via a network sensor.

Each scene instance can start local reactions upon reception of the changed global states.

Each scene instance can send "change requests" to the OBCO via a network sensor, to request changes of the global state.

4.1.2 "No-State" MIDAS Objects

"No-State" MIDAS Objects do not maintain a global state.

Examples for "no-state" MIDAS Objects are the "carried keys lock", the "beamer destination" and the "beamer", as well as the "trigger".

Ad "beamer" and "beamer destination":
The SSC holds an internal list of special <Viewpoint> nodes, an MFNode value. This list may be different in different scene instances. The "beamer destination" MIDAS Object announces its <Viewpoint> node during first time initialization to this list and deannounces it during disabling. The "beamer" MIDAS Object outputs a list of descriptions of all "beamer destinations" and takes a "description" as trigger to bind one of the <Viewpoint> nodes.

Ad "carried keys lock":
The SSC holds a list of "carried keys" (MFString). This list may be different in different scene instances (each avatar "carries" different keys).
The "carried keys lock" MIDAS Object outputs the "locked" state (SFBool), depending on the "carried keys" and depending on the "fitting keys".

4.1.3 "Animated" MIDAS Objects

"Animated" MIDAS objects do not maintain a time-discrete state, as standard MIDAS Objects do, but they maintain a (quasi-)continuous function state(t).

An example of an "animated" MIDAS Object is the "carousel drive". In this case, the function state(t) is a vector of "angle", "angular velocity" and "angular acceleration".

The quasi-continuous function state(t) is calculated in only one scene instance (in the OBCO – see chapter 4.1.1).

After a variable interval of time, the function state(t) is extrapolated to the future and state(future) is distributed as so-called "targets" to all scene instances.

The scene instances answer immediately with so-called "reports" to enable a controlling loop for all scene instances and to enable an estimation of the round trip times (RTTs) between the OBCO and each scene instance.

5 Additional Info

THE INTERFACE BETWEEN "MidasBase" and "MIDAS Object" IS EXPERIMENTAL AND SHOULD NOT BE CONSIDERED BEING FUTURE-PROOF.