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

This file valid for step 0033.11
Issue Date: tbd.

Naming Rules, MMF Paradigm

1 Synopsis

The Naming Rules and the MMF Paradigm are *the* concepts of SMS. Everybody, who develops Simple Multiuser Scenes, e.g. using the SMUOS Framework or using the SRR Framework, should be aware of the Naming Rules and of the MMF Paradigm.

2 Purpose of Naming Rules and MMF Paradigm

The whole concept of SMS is about how to put the bricks together for Simple Multiuser Scenes.

Each instance of an SMS can be decomposed into instances of facilities that have been downloaded from some resource via the Internet. We denote these instances as "Virtual Life Facilities" (VLFs).

Now when the instance of the SMS – the personal scene instance (PSI) – provides virtual senses and skills to its user, then this process can be interpreted as the exchange of events and messages among the VLFs.

Events are pieces of information that are exchanged among the VLFs within one and the same scene instance, where messages are pieces of information that are exchanged between VLFs of the same facility, but of different scene instances.

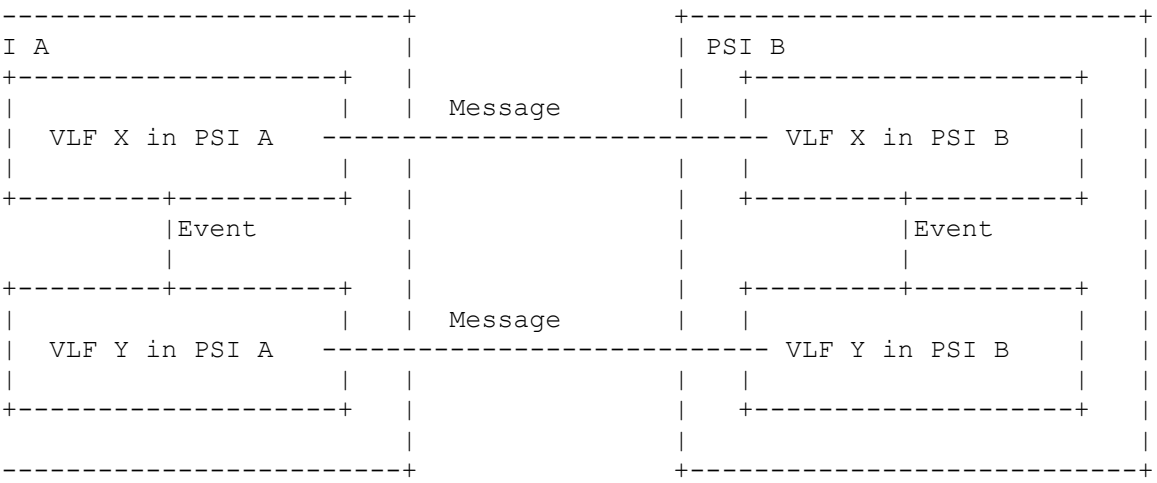

Figure 1. The exchange of messages is "hidden" within facilities

So we can state the author of the scene – who puts the facilities together – is only involved with the exchange of events. He is already used to it, if he already has done some work with VRML/X3D.

On the other hand, the exchange of messages to make the scene MU capable is "hidden" within the implementation of "MU capable facilities".

One drawback: it is not COMPLETELY hidden. The messages need some systematic addressing scheme to be able to traverse the Internet and find their destination VLF within the target PSI. This addressing scheme is the topic of the present paper.

3 External View

3.1 The MMF Paradigm (Models/Modules/Frame)

A Virtual World – or more general a Virtual Universe – can be interpreted as a set of models (e.g. houses, cars, horses, avatars and other confined structures) visiting or "inhabiting" a subset of an unconfined pool of "landscape elements".

Usually flight simulators use the term "tiles" for such "landscape elements".

The concept SMS derives from the hobby project SrrTrains v0.01, which considered simulated model railroads and hence decomposed the landscape into "modules" according to the "modular" approach for layouts of model railroads.

The concept SMS uses the term "module" intentionally, stressing the fact that modules might become something more general than tiles are. A tile is a part of the surface of a planet or of a moon, where a "module" has the potential to become a "confined (small) part of the universe".

Last but not least, there is a part of the scene – something "invisible and hard to explain", which holds the simulation together. We call this part of the scene "the frame".

Consequently an SMS consists of

- 1 Frame
- at least 1 "top level" module
- all modules can be "visited" / "inhabited" by models (and avatars)
- a model can contain "dependent" modules (*TODO11*)

Each module establishes a local coordinate system.

Each model can be attached to one module.

If the model is attached to a module, then the model can be rendered relative to that module. If the model is not attached to a module (if it is detached), then it cannot be rendered.

Figure 2 shows an example, where a model contains a dependent module, which contains three models in turn.

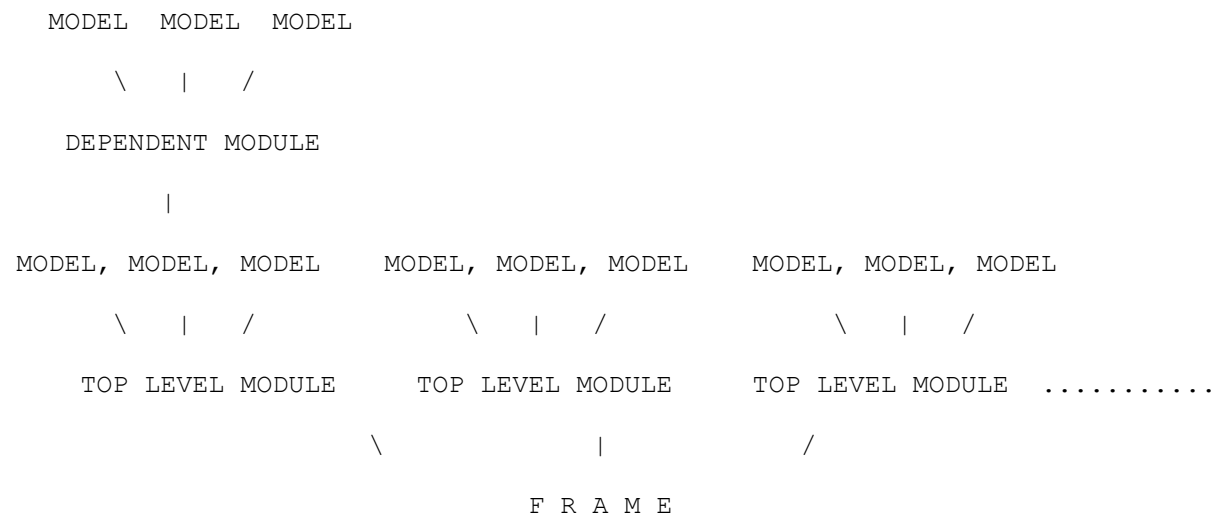

Figure 2. Enhanced MMF Paradigm (eMMF Paradigm - *TODO11*)

3.2 Module Names and Object IDs

In the original concept of SrrTrains each object was attached to one module.
Hence each object could be uniquely identified by

- <ObjInstance> = <DispInstance>-<objId>

where each module needed a so-called "dispatcher", which was used to dispatch commands from the console interface to the objects.

Each registered module could be identified by the "dispatcher instance"
- <DispInstance> = Bdo.<moduleName>

Module names are strings that are built by the characters
'A'-'Z', 'a'-'z', '0'-'9', '_' and '.' *).

This definition already indicates that "bound objects" (Bdo) are not the only possible kind of objects (see next chapter).

Within a module (or within a UOC, see next chapter), each object must be distinguished from each other. Object identifiers are strings that are unique within a module (or within a UOC).

Object IDs are strings, that are built by the characters
'A'-'Z', 'a'-'z', '0'-'9', '_' and '.' **).

*) Dependent modules inherit their module name partly from the top level module and from the <objId> of the objects that contain the dependent module
**) An object that is contained in a parent object, inherits a part of its <objId> from the parent object, i.e. <objId>=<parentObjId>.<objLocalId>.

3.3 UOC Names

Not every object is identified by a module (see "bound objects" in the chapter above).

Some objects exist outside of all modules – so called "astral objects" – and some objects may be able to change the module – "unbound objects" (UBOs).

Hence we need "something that groups objects, independent of the <moduleName>".
This concept was found in the concept of the Universal Object Class (UOC).

The SRR / SMUOS Framework provides an X3D Prototype, which is called the "Simple Scene Controller (SSC)".
This SSC consists of an "SSC Base", which supports the UOC "Base", but which does not support UBOs, and of "SSC Extensions".
Each provider of an "SSC Extension" can decide to support one or more UOCs, e.g. the UOC with <uocName>="Base.Trains" or the UOC with <uocName>="Base.Keys".

Each astral object (ASO) and each unbound object (UBO) are then identified by

- <ObjInstance> = <DispInstance>-<objId>
- <DispInstance> = Uoc.<uocName>

So the <ObjInstance> does not change, when the object changes the module.

UOC Names are strings, that are built by the characters
'A'-'Z', 'a'-'z', '0'-'9', '_' and '.'.

3.4 Extended Object IDs

The extended object ID is used in the tracer (to identify the tracer instances) and as a part of the identifiers within network traffic (which are called stream names in case of the SMUOS Framework).

The extended object ID does not change during the lifetime of an object.

Bound objects (cannot change module):
<extObjId>=Bdo.<moduleName>-<objId>
Unbound objects/Astral objects (can change modules or exist outside of modules):
<extObjId>=Uoc.<uocName>-<objId>

The SSC Base (with the UOC name "Base") contains currently one astral object with a reserved object ID, so following extended object ID is reserved:

- "Uoc.Base-DefAvaCon"....the default avatar container

3.5 Stream Names

Stream names, which are used to identify the network traffic of the VLFs, follow the basic structure:

- <StreamName> = Sms-<mainIdentifier>-<User>.<Suffix>

<mainIdentifier> uniquely identifies the VLF that uses the network sensor.

It is the intention of the prefix "Sms-" to keep the SRR/SMUOS Framework compatible with modules and models that use their own network sensors or that even use another network sensor dependent framework.

3.5.1 Network Sensors of the SSC Base and of the SSC Extensions

- "<mainIdentifier>" = "<SscInstance>" identifies the part of the SSC
- "<User>" = "Base", if it is a network sensor of SSC Base and
- "<User>" = "Ext", if it is a network sensor of an SSC extension
- "<Suffix>" can be set by the programmer arbitrarily

3.5.2 Network Sensors of the SSC Dispatchers and of the UBO Loaders

- "<mainIdentifier>" = "<DispInstance>" identifies the UOC or the module
- "<User>" = "SmsDispatcher" if it is a network sensor of the SSC Dispatcher
- "<User>" = "UboLoader" if it is a Network Sensor of the SSC UBO Loader
- "<Suffix>" can be set by the programmer arbitrarily

3.5.3 Network Sensors of the Objects (Models and MIDAS Objects)

- "<mainIdentifier>" = "<ObjInstance>" identifies the object
- "<User>" = "Obj" if it is a Network Sensor of the MIDAS object or the model
- "<User>" = "Mib", if it is a Network Sensor of MIDAS Base
- "<Suffix>" can be set by the programmer arbitrarily

3.6 Key IDs

Keys are identified by unstructured character strings. Key IDs need not be unique (same key may exist twice or more often).

3.7 Parameter Names

Parameter names are used at the console interface, together with module names, UOC names, and object IDs.

Parameter names are strings that are built by the characters 'A'-'Z', 'a'-'z', '0'-'9' and '_'.

3.7 Well-Known Extension IDs

The SSC may be extended by more than one SSC Extension

Hence the users of the SSC Extensions (i.e. the Module Coordinator Extensions and the Extension MIDAS Objects) must somehow get the possibility to address the specific extensions, which they need to access.

On the other hand, the Module Coordinators of a layout/SMS may be extended by more than one Module Coordinator Extension each.

Hence the users of the Module Coordinator Extensions (i.e. the Extension MIDAS Objects) must be able to address their extensions explicitely.

These two problems are solved by introducing
- Well-known SSC Extension IDs
- Well-known Module Coordinator Extension IDs

One layout/SMS (precisely spoken, one frame) cannot host two different SSC Extensions that use the same well-known ID.

One module cannot host two different Module Coordinator Extensions that use the same well-known ID.

The example SMUOS extensions that accompany the SRR/SMUOS Framework, use following well-known IDs:

Beamer Manager Extension:
WKI = "SmuosBeamerManager" (SSC Extension)

Key Manager Extension:
WKI = "SmuosKeyManager" (SSC Extension)

Train Manager Extension:
WKI = "SimulrrTrainManager" (SSC Extension)
WKI = "SimulrrTrainManager" (Module Coordinator Extension)

3.8 Overview about all Identifiers

3.8.1 Labels for Software Types

3.8.1.1 "<Wki>"

A well-known id identifies an SSC extension or an MC extension. <Wki>s are unique identifiers of SMUOS based software worldwide.

For example,
"<Wki>" = "SimulrrTrainManager"
identifies the SRR Train Manager, which is a part of the SRR Framework

3.8.1.2 "<Wku>"

A well-known uoc identifies a UOC relative to the SSC extension that defines the UOC.

For example, defines
"<Wku>" = "Trains"
the UOC "Trains" relative to the Train Manager SSC Extension.

"SimulrrTrainManager.Trains" is therefore a worldwide unique identification of the UOC "Trains".

Note: The "<Wku>" can not be changed, but the "<uocName>" can be different in different SMSs (the Frame Author can influence that).

3.8.1.3 "<SscClassPath>"

- "<SscClassPath>" = "Ssc.Base[.<Wki>.<Wku>]..."

The <SscClassPath> identifies a UOC within a specific hierarchy of SSC Base and SSC Extensions in an SMS.

The <SscClassPath> can be used as <ParentClassPath> in an <SscInstance> (see there).

In our example,
"<SscClassPath>" = "Ssc.Base.SimulrrTrainManager.Trains"

is the <ParentClassPath> for the SSC extension
"Ssc.Base.SimulrrTrainManager.Trains.FlyingTrainMgr" in this SMS.

3.8.1.4 "<McClassPath>"

- "<McClassPath>" = "McBase[.<Wki>]..."

The <McClassPath> identifies either the MC Base or the type of MC Extension within a specific hierarchy of MC Base and MC Extensions.

Together with the <moduleName>, the <McClassPath> in a <McInstance> is used to identify an instance of the MC Base or an MC Extension (see there).

For example, could an MC extension in our example scene the
"<McClassPath>" = "McBase.SimulrrTrainManager"
receive.

3.8.2 Labels for elements of the SMS according to the MMF paradigm

3.8.2.1 "<moduleName>"

A <moduleName> identifies a module within a Simple Multiuser Scene.

Example:
   "<moduleName>" = "City"
is the "second module" of the "official demo layout".

3.8.2.2 "<uocName>"

A <uocName> identifies a Universal Object Class (UOC) within a Simple Multiuser Scene.

Example:
   "<uocName>" = "Base" .....the UOC "Base" of the SSC Base
   "<uocName>" = "Base.Keys" .....a UOC managed by the SmuosKeyManager
   "<uocName>" = "Base.Trains" .....a UOC in the SimulrrTrainManager

3.8.2.3 "<extObjId>", "<objId>"

An <extObjId> identifies an object (that is, a MIDAS object or a model). The <objId> is the local object ID (within a UOC or within a module).

Bound objects:
   - "<extObjId>" = "<DispInstance>-<objId>" = "Bdo.<moduleName>-<objId>"
Astral objects:
   - "<extObjId>" = "<DispInstance>-<objId>" = "Uoc.Base-<objId>"
Unbound objects:
   - "<extObjId>" = "<DispInstance>-<objId>" = "Uoc.<uocName>-<objId>"

3.8.3 Designations for Software Instances within an SMS

Instances are names used in
   (1) the Tracer,
   (2) Stream Names and
   (3) the console interface
to identify VLFs of the SMS.

3.8.3.1 "<SscInstance>"

An <SscInstance> identifies either the SSC Base or an SSC Extension. It ALWAYS starts with the letter sequence "Ssc" and it does not contain the "-" character.

SSC Base:
   - "<SscInstance>" = "Ssc"
SSC Extension:
   - "<SscInstance>" = "<ParentClassPath>.<Wki>

Example: the "<SscInstance>" for Train Manager is
   "Ssc.Base.SimulrrTrainManager"

3.8.3.2 "<DispInstance>"

A <DispInstance> identifies either a UOC Related SSC Dispatcher or a Module Related SSC Dispatcher. It is also used in the included UBO Loader. It ALWAYS starts with one of the letter sequences "Bdo" or "Uoc" and it does not contain the "-" character.

Module Related SSC Dispatcher:
   - "<DispInstance>" = "Bdo.<moduleName>"
Example: "Bdo.City"

UOC Related SSC Dispatcher:
   - "<DispInstance>" = "Uoc.<uocName>"
Example: "Uoc.Base.Keys"

3.8.3.3 "<McInstance>"

An <McInstance> identifies an instance of the MC Base or of an MC Extension for a specific module. It ALWAYS starts with the letter sequence "Mod" and it consists of two parts that are separated by a "-" character.

   - "<McInstance>" = "Mod.<moduleName>-<McClassPath>"
For example, the Train Manager Extension in the "City" module would look like
   "<McInstance>" = "Mod.City-McBase.SimulrrTrainManager"

3.8.3.4 "<ObjInstance>"

An <ObjInstance> identifies an instance of a MIDAS object or of a model. It ALWAYS starts with one of the letter sequences "Bdo" or "Uoc" and it has two parts that are separated by a "-" character.

Bound objects:
- "<ObjInstance>" = "<extObjId>" = "Bdo.<moduleName>-<objId>"

Example: "Bdo.City-StationHouse.Door.Switch.Lock"

Astral objects:
- "<ObjInstance>" = "<extObjId>" = "Uoc.Base-<objId>"

Example: "Uoc.Base-DefAvaCon" is the default avatar container

Unbound objects:
- "<ObjInstance>" = "<extObjId>" = "Uoc.<uocName>-<objId>"

Example: "Uoc.Base.Trains-tgv0001"

4 Internal View

N/A

5 Additional Info

The MMF Paradigm and most of the naming rules will hold for the overall concept of SMS, however some of the naming rules will not survive the experimental SMUOS Framework. This is ffs.