Hibernation of Advanced Railroad Trains (ArrT)

Schritt Drei – .....and Communicate them (aCt)

Tracer Leaflet / LAN Party #3

This hibernation report describes the tracer that will be used at the LAN Party #3. However, the description is done from a current point of view – now at the beginning of the SPARK project – and will probably be updated to version 3.1, shortly before the LAN Party #3 will actually happen.

PROBABLY SOME PARTS OF THIS PAPER ARE CURRENTLY OUT OF DATE

1 Subsystems, Tracer Instances and Interfaces

1.1 Subsystems

Basically a subsystem is an abstract idea to group the source code of the software into parts that are functionally coherent.

The relation between subsystems is a "needs" relation. Either subsystem A "needs" subsystem B or subsystem B "needs" subsystem A or there is no relation between subsystem A and subsystem B.

A subsystem that "needs" another subsystem may host a tracer instance that "uses" an interface that is "provided" by a tracer instance that is hosted by the other subsystem.

E.g.:

- the MIDAS Base (MIB) needs the SMS Framework,
- a MIDAS Object (MOB) needs the MIDAS Base (MIB) or
- an SMS Module needs an SMS Frame.

1.2 Tracer Instances

Most of the tracer output is related to tracer instances:

- traceLevel=2: free info text related to a tracer instance
- traceLevel=2: message received/sent from/to another instance (via network sensor)
- traceLevel=2: event received/sent from/to another instance (within one scene instance)
- traceLevel=2: new state (new value of a state variable within the instance)
- traceLevel=2: start timer/stop timer/timer expired (within the instance)
- traceLevel=2: instance started/stopped (from another instance)
- traceLevel=3: free debug text related to a tracer instance

A tracer instance is the destination and origin of traced events (events within(!) scene instances) and of traced messages (events and states between(!) scene instances via network sensors).

A tracer instance can host a state variable or a timer.

A tracer instance can be started or stopped by another tracer instance.

1.3 Interfaces

An interface is a set of rules for information interchange. We say an instance "provides" an interface, if the rules are "mandated" by that instance. Other instances – which "obey" the rules – are said to "use" the interface.

E.g.:

- the SscBase "provides" the uiControl interface, which is "used" by an SMS frame
- the SscDispatcher "provides" the iiDisp interface, which is "used" by SscBase and by some <SscExt>s to instantiate and initialize the SSC Dispatchers
- on the other hand, the SscDispatchers "use" the iiControl interface, which is "provided" by SscBase. In this case the iiControl interface is used to dynamically register at the SscBase.

Following interfaces are defined by the SMUOS Framework:

- Dispatcher Stub provides
  - eiConsole – used by SSC Base, may be used by any MOB and by any SSC extension
- CommControl provides
  - niControl – network interface used by SscBase
- SscBase provides
  - iiControl – internal interface used by module coordinator and by SSC dispatcher
  - eiControl – external interface used by MIB, may be used by any MIDAS Object, may be used by any SMUOS extension
  - uiControl – user interface may be used by any SMS Frame
- SscBase requires
  - miControl – must be provided by any SSC extension
- SSC dispatcher provides and uses
  - niDisp – network interface used within SSC dispatcher
- SSC Dispatcher provides
  - eiDisp – external interface used by dispatcher stub
  - iiDisp – internal interface used by SSC Base, may be used by any SSC extension
- Module Coordinator provides
  - eiMod – external interface used by MIB, may be used by any MIDAS Object, may be used by any SMUOS extension
  - uiMod – user interface may be used by any SMS Module
- Module Coordinator requires
  - miMod – must be provided by any MC extension

- MIB provides and uses
    - iiMib – internal interface provided by Mib Core and OSM, used by any MIB
    - niMib – network interface used within MIB
- MIB provides
    - eiMib – external interface may be used by any MOB
- SMUOS Framework requires
    - uiObj – must be provided by every MIDAS Object
    - miModel – must be provided by every SMS Model
    - miModule – must be provided by every SMS Module

2 A Short Idea about the SMUOS Framework

The subsytems, tracer instances and interfaces of the SMUOS Framework have been developing over a long time. Here i try to explain a few of the crucial ideas to understand what's going on in the SMUOS Framework.

First idea was to split SrrTrains layouts into modules. Thus it would be possible to provide modules from different authors and to create SrrTrains layouts from parts of different authors.

Models should inhabit the modules and avatars should be able to meet within this scene.

The second paradigm was to keep the networking issues as much as possible "within the models". General networking infrastructure should be kept in what we call "the frame" and this infrastructure should be made accessible by the models via what we call "common parameters" (commParam).

This second paradigm led to the idea of the "MIDAS Objects" (originally they were called "SRR Objects"). Module authors and model authors should not care about the networking issues, they should just use the MIDAS objects to provide the interactivity, animation and simulation for the models, where the networking issues should be encapsulated within the MIDAS Objects.

Thus only the frame author and the programmer of the MIDAS objects would be affected by networking issues.

Hence the SMUOS Framework (former "Base Module of the SRR Framework) was split into the "SmsFramework" subsystem and the "MidasBase" subsystem (MIB), where MIDAS Objects could be built on top of the MIB.

A special case is the "CentralController" subsystem, which could be kept within the "SmsFramework" subsystem.

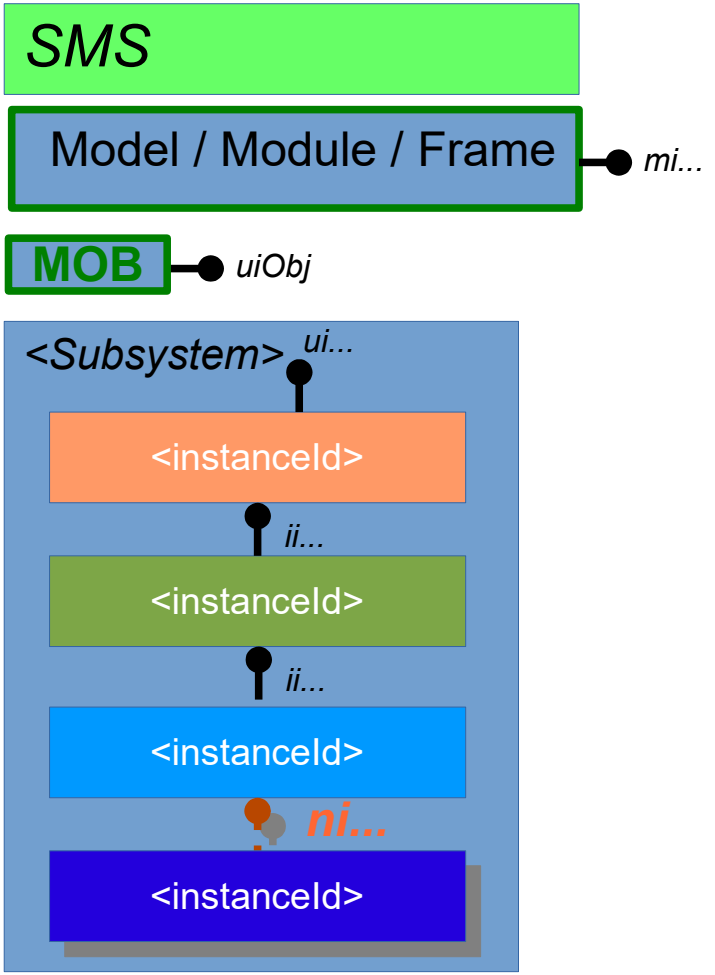
However, the intention is to move the software from the "CentralController" subsystem into the collaboration server in a later stage, therefore we keep it in a separate subsystem already.

3 Tracer Leaflet

The following pages are intended to be handed out as "tracer leaflet" to all participants of LAN Party #3.

The tracer leaflet should help the tester to interpret tracer output.

It should give a basic understanding of SMUOS subsystems, tracer instances and interfaces.



...Simple Multiuser Scene using the SMUOS Framework

...Element provided by an author. It provides a minimum interface (mi...)

...MIDAS Object provided by a programmer

...Subsystem of the SMUOS Framework

...containing an object related instance (red) that provides a user interface (ui...) to be used by 3rd party authors

...containing a module/UOC related instance (green) that provides an internal interface (ii...)

...containing a scene related instance (blue), which is a client instance (it uses a network interface ni...)

...containing a scene related instance (blue), which is a server instance (it has a shadow)

Figure 1: Legend for the tracer leaflet

This page intentionally left blank

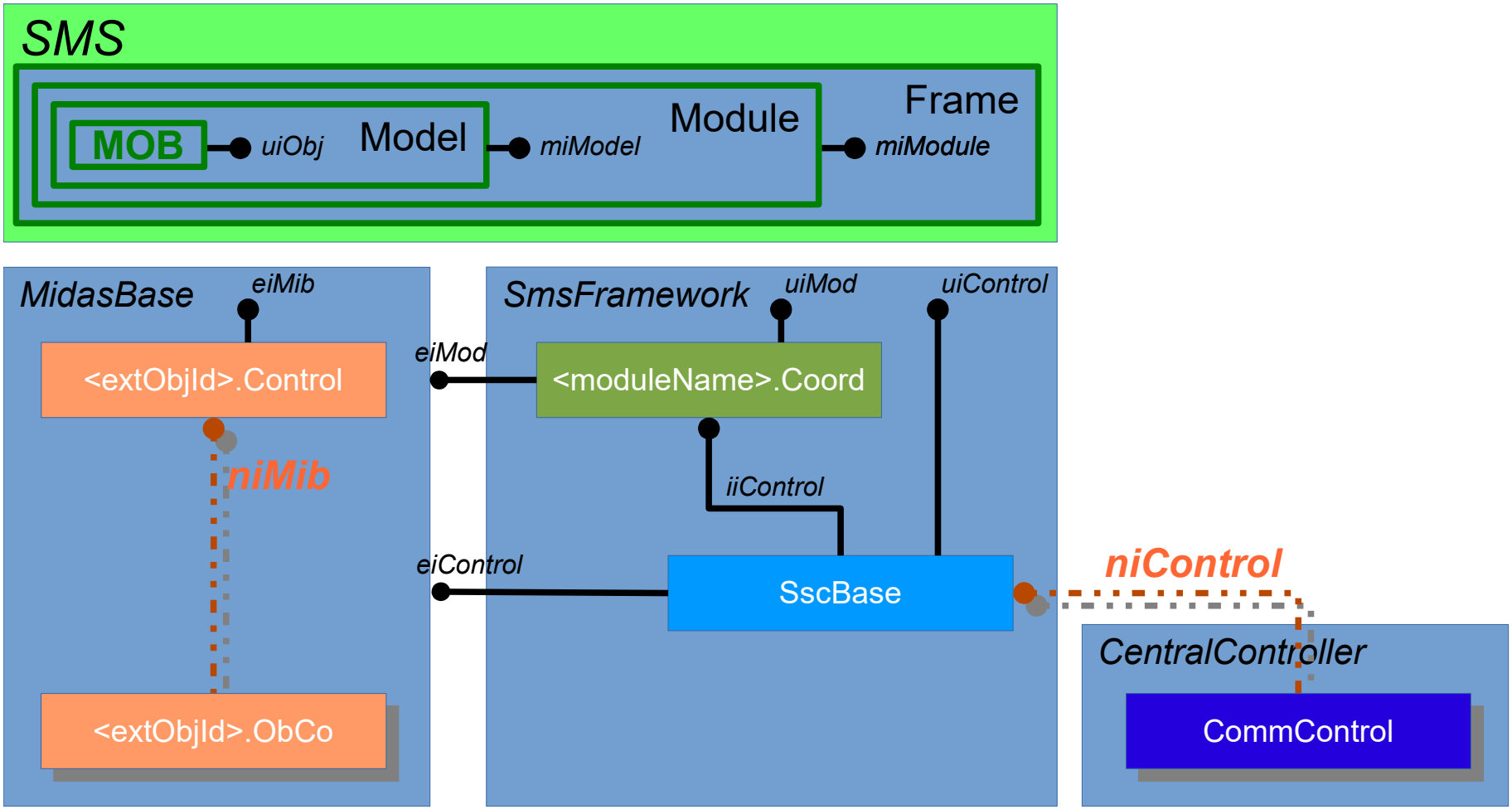


Figure 2: Basic subsystems, instances and interfaces within an SMS (w/o console interface, w/o SMUOS extensions)

3.1 How To Set Trace Levels of the SMUOS Framework

THIS TEXT IS RELATED TO Figure 2!!!

Trace Level 0 only fatal errors are output

Trace Level 1 fatal errors and errors are output (this is the default setting at all tracer instances)

Trace Level 2 fatal errors, errors and info messages are output (use this setting to narrow down an error / effect)

Trace Level 3 fatal errors, errors, info messages and debug messages are output (use this setting for the instance, where you assume the error)

Object Related Tracer Instances (red)

Set the trace level for selected object(s):

Objects = **<extObjId>=TL_overall[;<extObjId>=TL_overall]...**

Module Related Tracer Instances (green)

Set the trace level for selected module(s) – operational trace level effective after initialization, init trace level effective during initialization:

Modules = **<moduleName>=TL_oper [, TL_init][;<moduleName>=TL_oper [, TL_init]]...**

Scene Related Tracer Instances (blue)

Set the trace level for SscBase – operational trace level effective after initialization, init trace level effective during initialization:

SscBase = **TL_operation [, TL_initialization]**

Set the trace level for CommControl – operational trace level effective after initialization, init trace level effective during initialization:

CommControl = **TL_operation [, TL_initialization]**

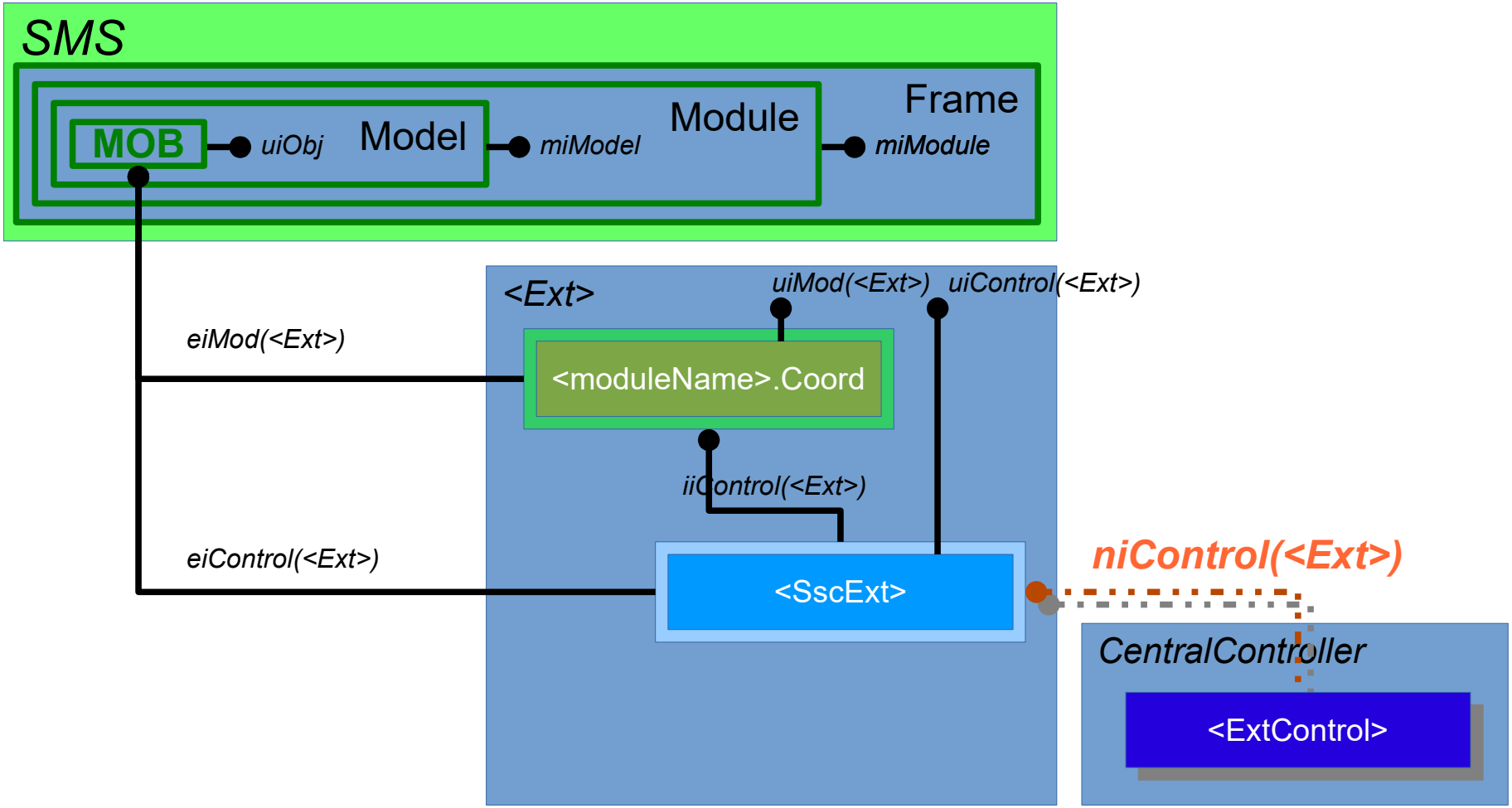


Figure 3: Basic subsystems, instances and interfaces with a SMUOS extension (this is a suggestion)

3.2 How to Set Trace Levels of the SMUOS Extensions

THIS TEXT IS RELATED TO Figure 3!!!

Trace Level 0 only fatal errors are output

Trace Level 1 fatal errors and errors are output (this is the default setting at all tracer instances)

Trace Level 2 fatal errors, errors and info messages are output (use this setting to narrow down an error / effect)

Trace Level 3 fatal errors, errors, info messages and debug messages are output (use this setting for the instance, where you assume the error)

Key Manager Extension / Beamer Manager Extension

SscKeyManager/KeyControl and SscBeamerManager inherit their trace levels from SscBase/CommControl

Train Manager Extension

Module Related Tracer Instances (green)

Set the trace level for selected module(s) – operational trace level effective after initialization, init trace level effective during initialization:

ModulesTm = **<moduleName>=TL_oper [, TL_init][;<moduleName>=TL_oper [, TL_init]]...**

Scene Related Tracer Instances (blue)

Set the trace level for SrrControlTm – operational trace level effective after initialization, init trace level effective during initialization:

SrrControlTm = **TL_operation [, TL_initialization]**

Set the trace level for TrainControl – operational trace level effective after initialization, init trace level effective during initialization:

TrainControl = **TL_operation [, TL_initialization]**

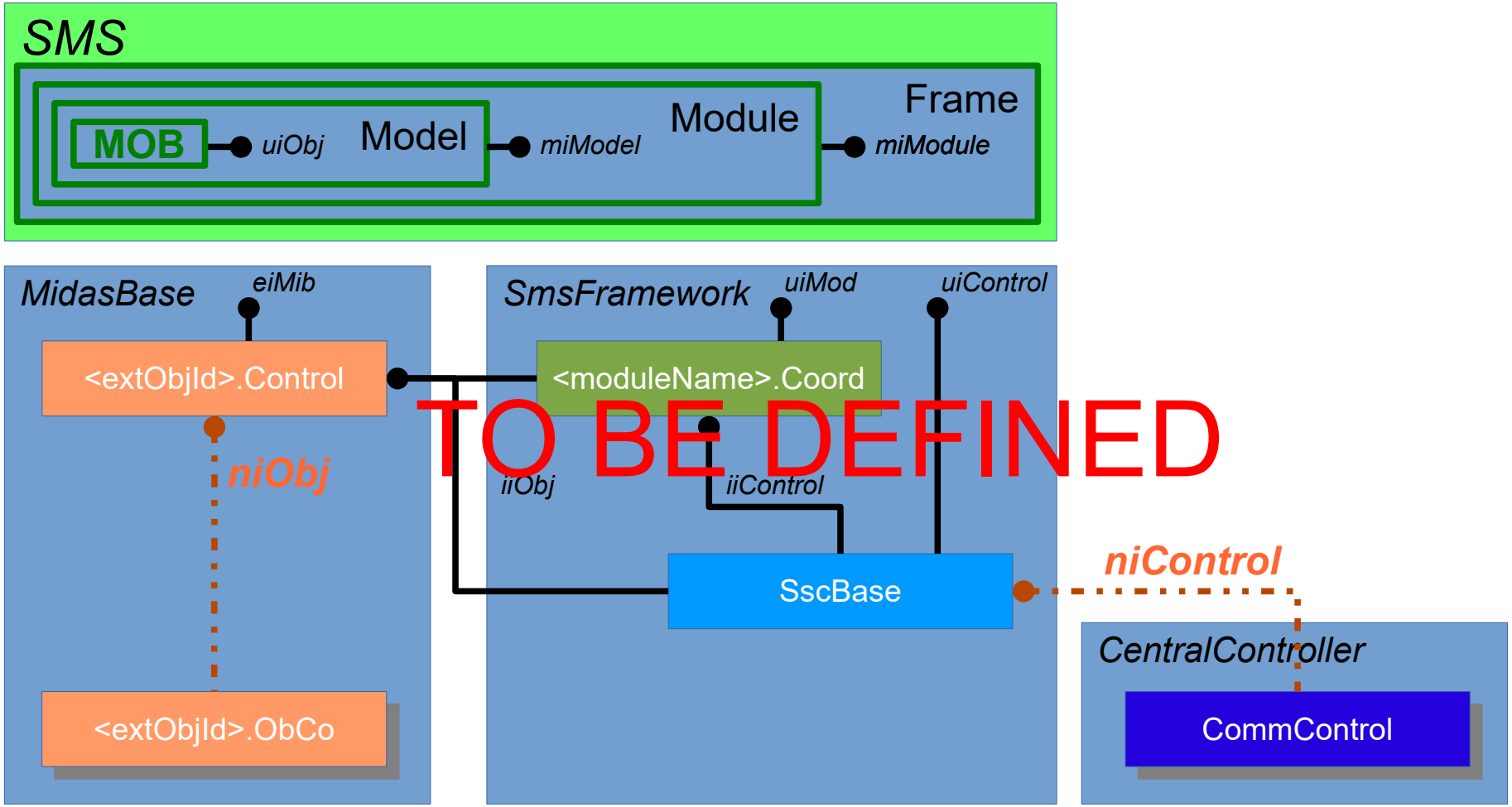


Figure 4: Subsystems, instances and interfaces for unbound models (TrainManager example)

3.3 How Unbound Models can be Used with the Tracer

Unbound Models of the Train Manager Extension: to be defined

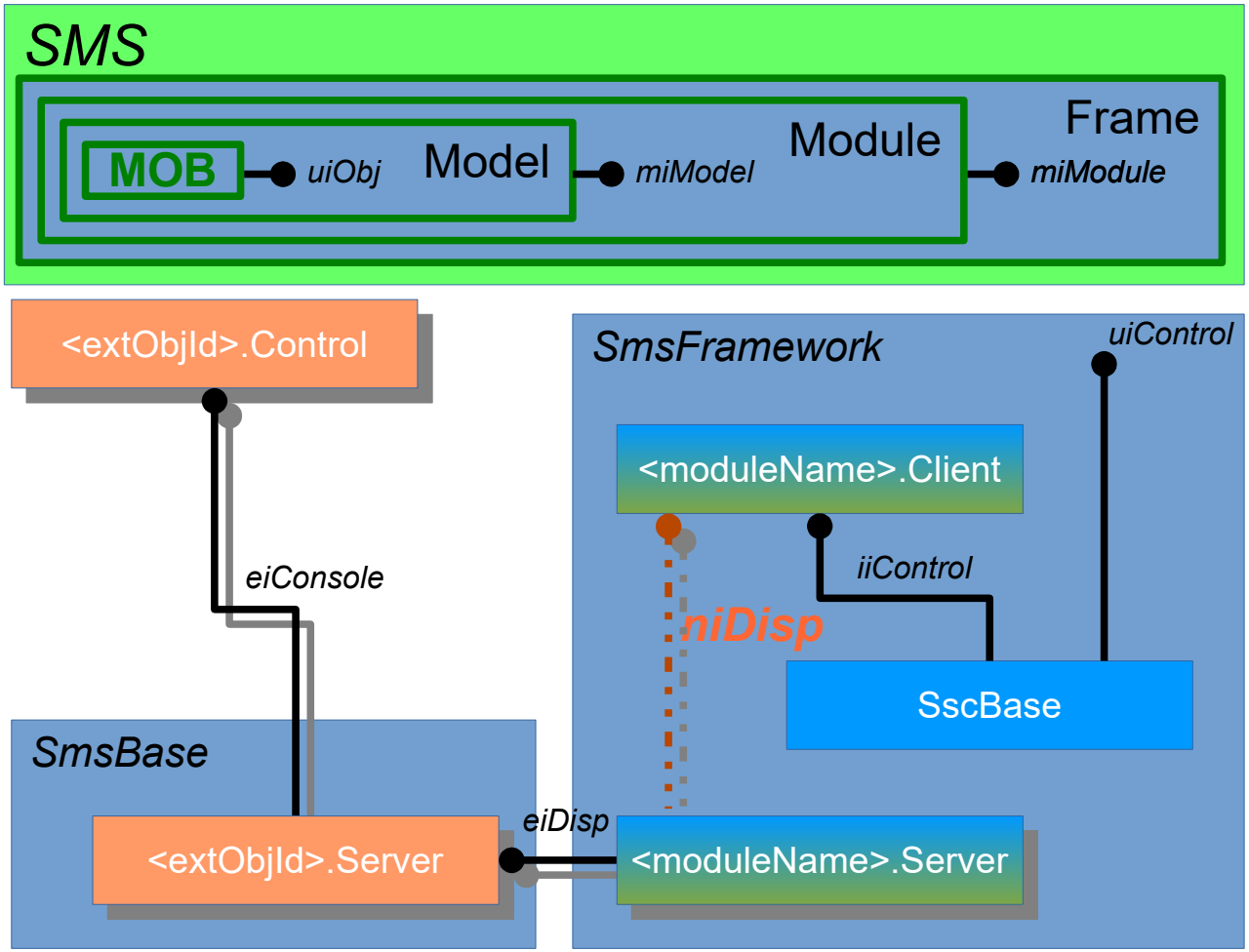


Figure 5: Console interface for bound objects

3.4 How the Console Interface Works

THIS TEXT IS RELATED TO Figure 5!!!

When addressing messages (i.e. console requests) to an object, one has to be aware an object need not be active in every scene instance.

Not even the module containing the object need to be present in every scene instance.

Hence we are forced to send a message to an object via the SSC, which IS present in every scene instance.

For this purpose, the SSC contains what we call "SSC Dispatchers", one for each module and one for each UOC.

The SSC Dispatcher has an internal network interface niDisp, which is used to address the OBCO of the addressed object.

The so-called Dispatcher Stub is a prototype that is contained in each and every object and that uses the interface eiDisp of the SSC Dispatcher to register for console requests.

<moduleName>.Client is the instance of the SSC dispatcher that is locally addressed by the requesting SSC Base.

<moduleName>.Server is the instance of the SSC dispatcher, that delivers the console request locally to the object.

<extObjId>.Server is the instance of the Dispatcher Stub, that delivers the commands via the eiConsole interface to the object.

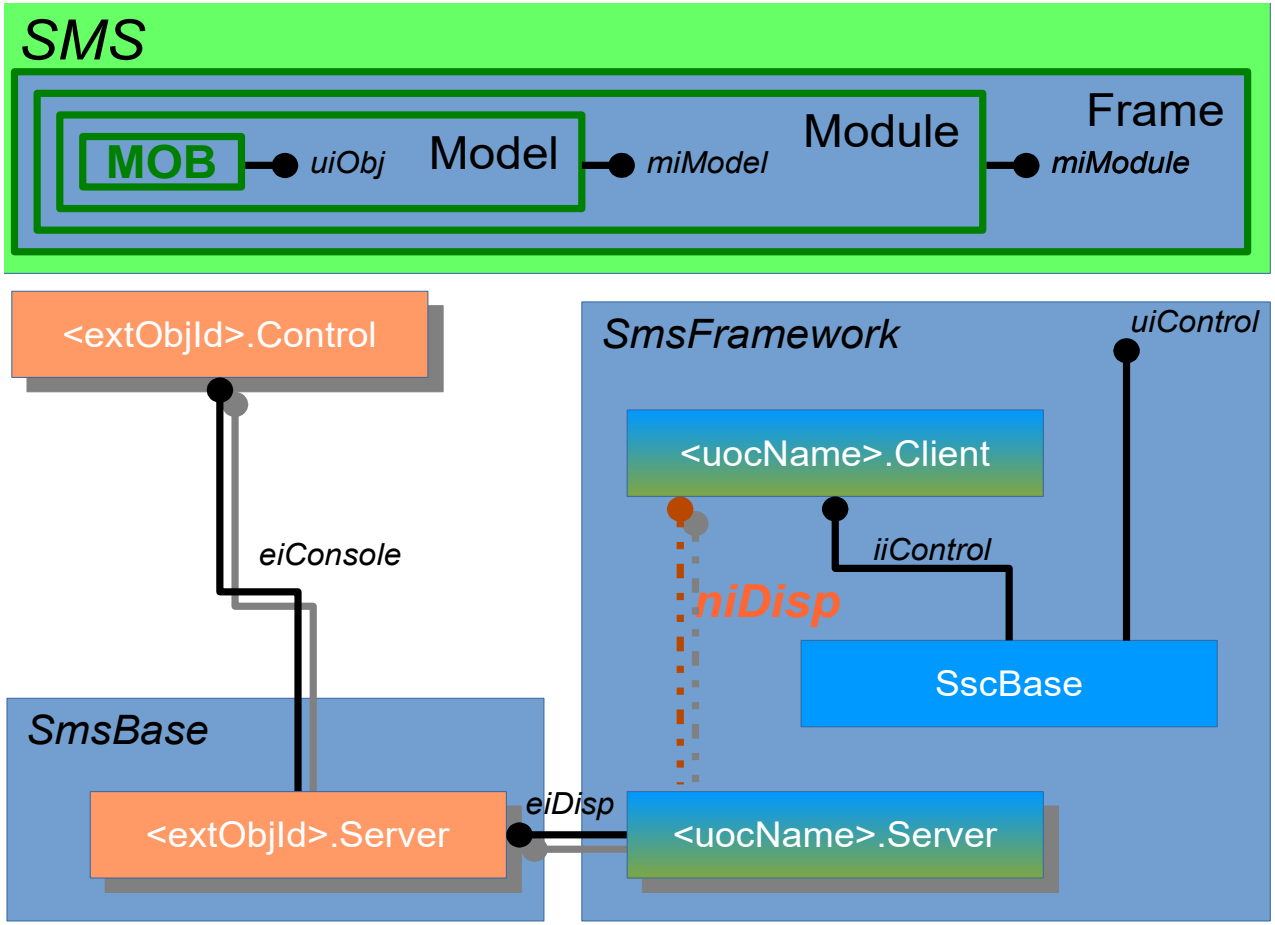


Figure 6: Console interface for unbound objects

3.5 How the Console Interface Works for Unbound Models

THIS TEXT IS RELATED TO Figure 6!!!

There is not only one SSC Dispatcher per module, but there is also one SSC Dispatcher per UOC.

Each UOC addresses a class of unbound objects.

Each SSC extension may choose to support one or more UOCs. In this case, it has to instantiate one SSC Dispatcher per UOC.

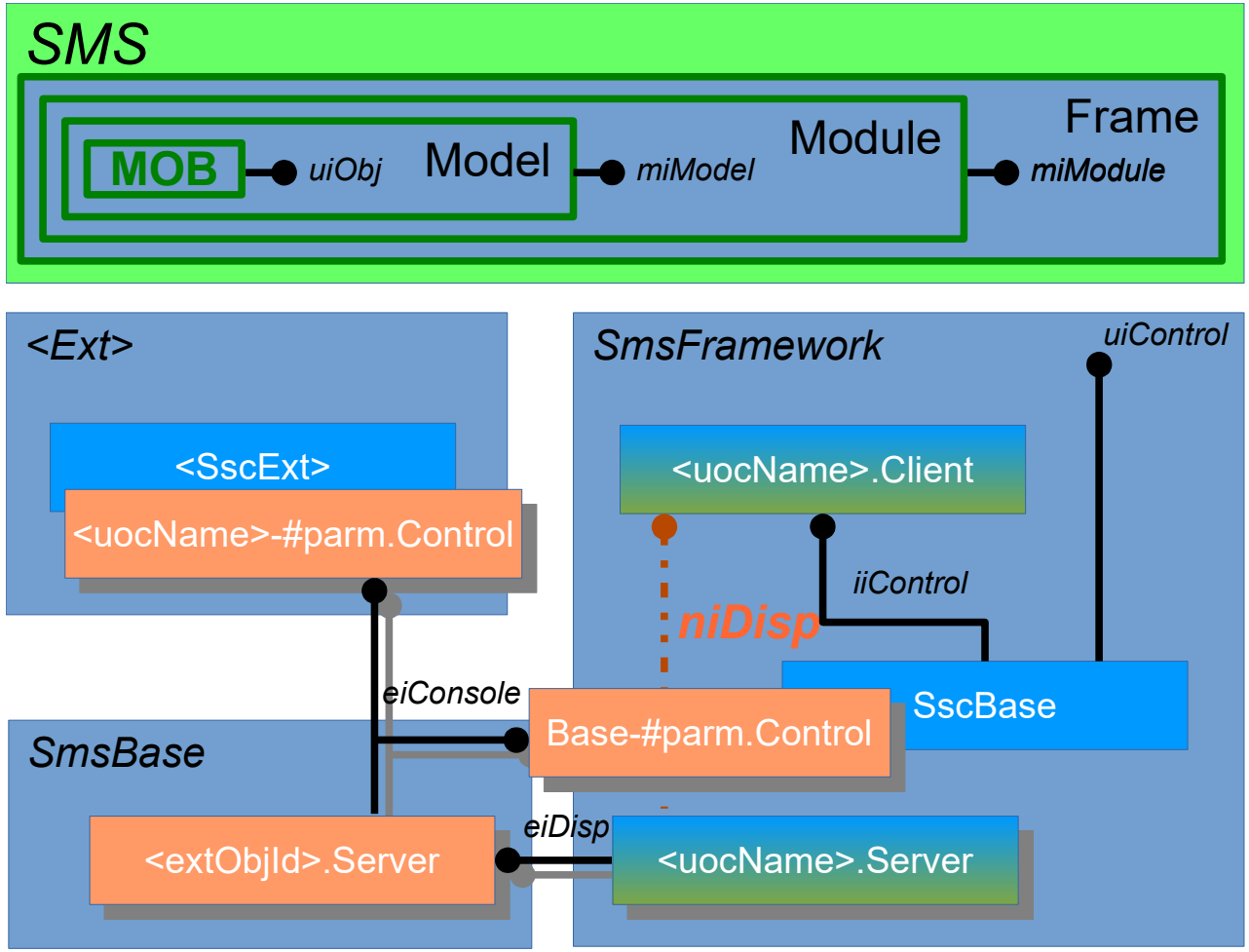


Figure 7: Console interface for SSC parameters

3.6 How the Console Interface Works for SSC Parameters

THIS TEXT IS RELATED TO Figure 7

Each SSC Dispatcher instantiates one instance of the dispatcher stub.

Hence each UOC related SSC Dispatcher may support SSC parameters, if the SSC Base (which actually does) or the SSC extension uses the provided eiConsole interface.

3.7 The Content of the sourceforge Projects, as Seen by the Tracer

This chapter provides two tables with a basic view to the

1. tracer instances within the official demo layout,
2. tracer instances within the example track geometry,
3. tracer instances within the Train Manager Extension,
4. tracer instances within the Key Manager Extension,
5. tracer instances within the Beamer Manager Extension,
6. tracer instances within the example basic MIDAS objects and
7. tracer instances within the SMUOS framework,

as provided by the projects http://simulrr.sourceforge.net and http://smuos.sourceforge.net.

Subsystem	File(s)	Instance(s)	Use Case(s)
Official Demo Layout			
MyFrame	ConsoleHud.x3d	**MyConsoleHud**	Frame / SSC
	ControlHud.x3d	**MyControlHud**	
	Main_bscontact.x3d Main_instant.x3d Main_octaga.x3d	**MyController**	
MyFirstModule	Hill.x3d	**<moduleName>.MyController** default: Hill	Module / MC
MySecondModule	City.x3d	**<moduleName>.MyController** default: City	
MyThirdModule	Mountains.x3d	**<moduleName>.MyController** default: Mountains	
MyFourthModule	Dunes.x3d	**<moduleName>.MyController** default: Dunes	
SimpleSetupPoint	SimpleSetupPoint.x3d	**<extObjId>.MyController**	Bound Object
Example Track Geometry (tg/)			
SrrTrackSectionA	SrrTrackSectionA.x3d	**<extObjId>.MyController**	Bound Object
SrrTrackSectionB	SrrTrackSectionB.x3d	**<extObjId>.MyController**	
SrrTurnoutLeftA	SrrTurnoutLeftA.x3d	**<extObjId>.MyController**	
SrrTurnoutLeftB	SrrTurnoutLeftB.x3d	**<extObjId>.MyController**	
SrrTurnoutRightA	SrrTurnoutRightA.x3d	**<extObjId>.MyController**	
SrrTurnoutRightB	SrrTurnoutRightB.x3d	**<extObjId>.MyController**	

Subsystem	File(s)	Instance(s)	Use Case(s)
SrrTrackGeometryABI	SrrTrackGeometryABI.x3d	**<extObjId>.MyGeometry**	
Train Manager Extension (tmm/)			
TrainMobs.Replicator	SrrReplicator.x3d	**<extObjId>.Control**	Bound Object
TrainMobs.BasicTurnout2Way	SrrBasicTurnout2Way.x3d	**<extObjId>.Control**	
TrainMobs.BasicTrackSection	SrrBasicTrackSection.x3d	**<extObjId>.Control**	
TrainMobs.TrackEdge	SrrTrackEdge.x3d	**<extObjId>.Control**	
TrainMobs.TrackNode	SrrTrackNode.x3d	**<extObjId>.Control**	
TrainFramework	SrrModCoordTm.x3d	**<moduleName>.Coord**	Module / MC
	SrrControlTm.x3d SrrControlTmNs.x3d	**SscTrainManager**	Frame / SSC
CentralController		**TrainControl**	
Key Manager Extension (sms/)			
KeyMobs.CarriedKeysLock	MoosLockA.x3d	**<extObjId>.Control**	Object
KeyMobs.KeyContainer	MoosKeyContainer.x3d MoosKeyContainerNs.x3d	**<extObjId>.Control**	
		<extObjId>.ObCo	
KeyMobs.ContainedKeyLock	MoosLockB.x3d MoosLockBNs.x3d	**<extObjId>.Control**	
		<extObjId>.ObCo	
KeyFramework	SscKeyManager.x3d	**<uocName>-#parm.Control**	SSC Parameter
	SscKeyManager.x3d SscKeyManagerNs.x3d	**SscKeyManager**	Frame / SSC
CentralController		**KeyControl**	
Beamer Manager Extension (sms/)			
BeamerMobs.BeamerDestination	MoosBeamerDestination.x3d	**<extObjId>.Control**	Object
BeamerMobs.Beamer	MoosBeamer.x3d	**<extObjId>.Control**	
BeamerFramework	SscBeamerManager.x3d	**SscBeamerManager**	Frame / SSC

Table 1: Subsystems and tracer instances provided by simulrr.sourceforge.net

Subsystem	File(s)	Instance(s)	Use Case(s)
Example Basic MIDAS Objects (sms/)			
BasicMobs.AvatarContainer	MoosAvatarContainer.x3d	**<extObjId>.Control**	Object Astral Object
BasicMobs.Trigger	MoosTrigger.x3d MoosTriggerNs.x3d	**<extObjId>.Control**	Object
BasicMobs.BinarySwitch	MoosSwitchA.x3d MoosSwitchANs.x3d	**<extObjId>.Control**	
		<extObjId>.ObCo	
BasicMobs.NwaySwitch	MoosSwitchB.x3d MoosSwitchBNs.x3d	**<extObjId>.Control**	
		<extObjId>.ObCo	
BasicMobs.CarouselDrive	MoosDriveA.x3d	**<extObjId>.Control**	
		<extObjId>.ObCo	
BasicMobs.Creator	MoosCreator.x3d	**<extObjId>.Control**	
		<extObjId>.ObCo	
SMUOS Framework (sms/)			
MidasBase	MibAnim.x3d MibAnimNs.x3d	**<extObjId>.Control**	Object
		<extObjId>.ObCo	
	MibStandard.x3d MibStandardNs.x3d	**<extObjId>.Control**	
		<extObjId>.ObCo	
	MibNoState.x3d	**<extObjId>.Control**	
	MibCore.x3d	**<extObjId>.Control**	
SmsFramework	McBase.x3d	**<moduleName>.Coord**	Module / MC
	SscDispatcher.x3d SscDispatcherNs.x3d	**<moduleName>.Client**	Frame / SSC
		<uocName>.Client	
		<moduleName>.Server	
		<uocName>.Server	
	SscBase.x3d	**Base-#parm.Control**	SSC Parameter
	SscBase.x3d SscBaseNs.x3d	**SscBase**	Frame / SSC
CentralController		**CommControl**	
SmsBase	SmsDispatcherStub.x3d	**<extObjId>.ObCo**	Object SSC Parameter
	SmsTracer.x3d	**-----**	-----

Table 2: Subsystems and tracer instances provided by smuos.sourceforge.net

4 The Prototypes SmsTracer and SmsDispatcherStub

4.1 Overview

Besides all the other functionalities the SMUOS framework provides two very basic prototypes,

1. the SMS Tracer prototype "SmsTracer" and
2. the SMS Dispatcher Stub prototype "SmsDispatcherStub".

The dispatcher stub uses the eiDisp interface to register for information that is provided by some user.

Any user can send information to a MIDAS object or to an SSC parameter, given that object has initialized its dispatcher stub and hence the dispatcher stub has announced at the SSC dispatcher.

Therefore the user addresses the destination object by its <extObjId> and releases the information to the SSC, using either of the interfaces uiControl, iiControl or eiControl.

The SMS Tracer is used by all prototypes of the SMUOS framework (including the dispatcher stub) to output errors, info and debug logs to the Web3D Browser console.

Additionally, the logs are output at the uiControl interface and can be processed by the frame of the SMS.

We have tried hard to keep the tracer output very systematic, therefore enabling the (semi-) automatic processing of logs for graphical display.

4.2 Parameters of a Tracer Instance

Each instance of the SmsTracer prototype holds following vital parameters:

4.2.1 Parameters Identifying the tracer instance

Following parameters identify the tracer instance and are always present in the tracer output:

- instanceId (SFString) – ID of the tracer instance (see chapter 3 for examples)
- subsystem (SFString) / fileName (SFString) – subsystem and file, where the tracer is instantiated (see chapter 3 for examples)
- ssVersion (SFString) – version of the subsystem

4.2.2 Parameters Identifying the Use Case

The SmsTracer prototype may be used for each of several use cases:

- tracer instance in the frame or in the SSC
- tracer instance in an astral object (this is an exceptional case)
- tracer instance in a module or in the MC
- tracer instance in a bound object
- tracer instance in an SSC parameter object (#parm) or in an unbound object

The use cases are identified as follows:

Use Case	universalObjectClass	objId	modParam
Frame / SSC	-	-	-
Astral Object	-	present	-
Module / MC	-	-	present
Bound Object	-	present	present
Forbidden Use Case	present	-	don't care
Unbound Object / SSC Parm	present	present	don't care

Table 3: How to identify the use case of a tracer instance

Following parameters are present in the tracer output dependent on the use case:

Use Case	Additional Output Parameters (except instanceId, subsystem, fileName and ssVersion)
Frame / SSC	none
Astral Object	objId=<object ID of the astral object>
Module / MC	moduleName=<name of the module>
Bound Object	objId=<object ID of the bound object> moduleName=<name of the module>
Forbidden Use Case	N/A
Unbound Object / SSC Parm	objId=<object ID of the object> moduleName=<name of the parent module, if available>

Table 4: Important output of a tracer instance by use case

Note: Sometimes we denote an "aggregated" use case "Object". This use case means a tracer instance may be used either in bound objects or in unbound objects, depending on the case.