Hibernation of Advanced Railroad Trains (ArrT)

Schritt Zwei – Elaborate Visions ..... / Leave Hibernation (EViL)

New Instance Naming

Dieser Hibernation Report ist ein "Snapshot", er wird also nicht mehr upgedatet werden!

Er zeigt die "New Naming Rules" nach ihrer Implementierung zu Beginn des Step 0033.11.

1 Problemstellung

Bei den ersten Überlegungen zum Beginn der Implementierung von UBOs (Unbound Objects) bin ich über einige Probleme "gestolpert", die ich jetzt lösen möchte.

1. Bei der Implementierung des SSC Core in Step 0033.10 hatte ich bereits vorgesehen, dass man SSC Extensions ineinander verschachteln kann, jedoch hatte ich nicht daran gedacht, die Verschachtelung von der Existenz von UOCs abhängig zu machen.
   Jetzt wäre es aber sinnvoll, für UBOs eine "Klassenhierarchie" zu entwickeln, wo UOCs andere UOCs als "Basisklasse" haben (Details im Kapitel 1.1).
2. Es gibt mehrere "Schienen" der Namensgebung, nämlich die sog. "Tracerinstanzen", die sogenannten "Well Known Ids" und die "Stream Names".
   Es wäre sinnvoll, die Tracerinstanzen von den Well Known IDs abzuleiten und die Stream Names mit den Tracerinstanzen gleichzuziehen, damit man nur mehr eine "Schiene" hat.

1.1 Klassenhierarchie für Unbound Objects (UBOs) – Ideen

Weiterhin soll der Simple Scene Controller Base (SSC Base) die Universal Object Class (UOC) "Base" zur Verfügung stellen.

Wenn man eine SSC Extension implementiert, dann sollte man sich jetzt entscheiden müssen, welche UOC durch diese SSC Extension "refined" wird, die Parent SSC Extension könnte ja mehrere UOCs definieren.

Eine Klassenhierarchie könnte also aussehen wie folgt:

SSC Base	**Klasse**	**SSC Extension**	**Klasse**	**SSC Extension**	**Klasse**
Ssc	Base	SscTrainManager	Trains	FlyingTrainMgr	FlyingTrains

Damit wäre **"Ssc.Base.SscTrainManager"** ein Identifizierer der Train Manager Extension.

"**Ssc**" würde SSC Base und "**Ssc.Base.SscTrainManager.Trains.FlyingTrainMgr**" würde den Flying Train Manager identifizieren.

Der Flying Train Manager würde auch Funktionen des Train Managers verwenden, weshalb die Klasse "Base.Trains.FlyingTrains" von der Klasse "Base.Trains" hergeleitet würde.

Die Klasse "Base" wird nicht für UBOs verwendet werden sondern nur für Astrale Objekte (ASOs).

Die "Klassen" (UOCs) werden nur für UBOs und ASOs verwendet, nicht jedoch für Bound Objects (BDOs). BDOs werden nicht über UOCs definiert, sondern über Module (eben die Module, an die sie gebunden sind).

2 Hintergrund – MMF Paradigma

Ich habe das MMF Paradigma jetzt schon so oft erklärt, trotzdem möchte ich es für den unbedarften Leser hier nochmals tun, damit dieser "Hibernation Report" für sich selbst stehen kann.

Also, entsprechend der "Modulbauweise" von Modellbahnen, will ich die Landschaft der virtuellen Modelleisenbahn in sogenannte "Module" zerlegen und auf den Modulen sollen sich "Modelle tummeln". Module sollten bei Bedarf "nachgeladen" werden oder auch wieder "entladen".

Weiters brauchen wir ein "abstraktes Etwas", eine "gewisse Infrastruktur", die von allen Modulen und Modellen verwendet werden kann, um dem User ein virtuelles Erlebnis garantieren zu können. Dieses "abstrakte Etwas" bezeichne ich als den "Rahmen" (engl. "Frame").

In erster Linie ist es also so, dass jedes Modell einem Modul zugeordnet wird – denn ein Modul spannt ein lokales Koordinatensystem auf, zu dem relativ ein Modell dargestellt (gerendert) werden kann. Das ist in folgender Abbildung 1 dargestellt:

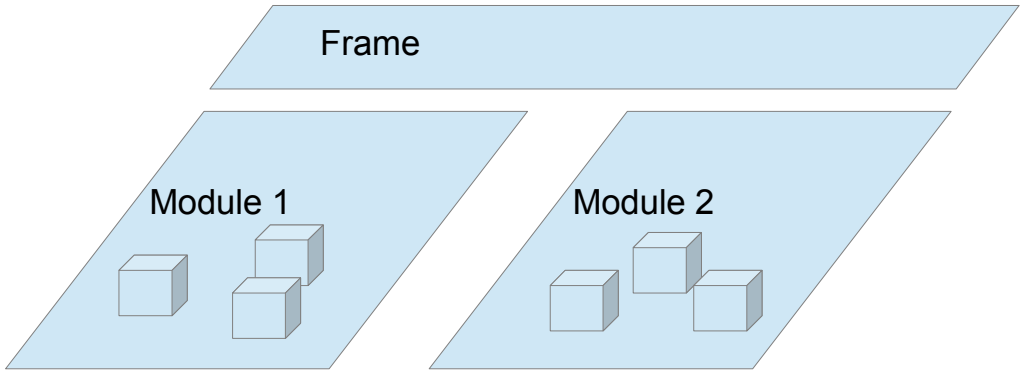

Abbildung 1: Objekte, die an ein Modul gebunden sind (Bound Objects - BDOs)

Ziemlich von Anfang an gab es auch ein Objekt, das keinem Modul zugeordnet war, das also auch ohne Modul existieren konnte, das war der "Avatar Container". Später habe ich solche Objekte dann als "Astrale Objekte (ASOs)" bezeichnet. Diese sind nur dem Rahmen zugeordnet (Abbildung 2):

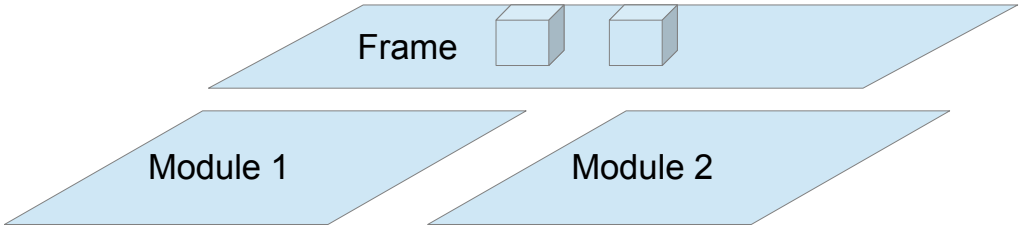

Abbildung 2: Objekte, die an kein Modul gebunden sind (Astral Objects - AOBs)

Da astrale Objekte keinem Modul zugeordnet sind, können sie auch nicht gerendert werden. Entsprechend meiner Nomenklatur ("Objekt" ist entweder "Modell" oder "MIDAS Objekt") können also nur MIDAS Objekte als astrale Objekte fungieren, Modelle können das nicht (ein Modell, das nie gerendert werden kann, ergibt keinen Sinn).

Nun gibt es noch die ungebundenen Objekte (UBOs) – das heisst, es WIRD sie geben – die im Normalfall einem Modul zugeordnet sind, damit man sie rendern kann, die aber auch das Modul wechseln können (was ich als "Handover" bezeichne) und sie können sogar vorübergehend als astrale Objekte fungieren – z.B. wenn man ihnen "das Modul unter dem Hintern wegzieht".

Der Übergang von einem Modul zum Frame wird als "DeAttachment" bezeichnet – das Objekt ist dann "detached" – und der Übergang vom Frame zu einem Modul ist ein "Attachment".

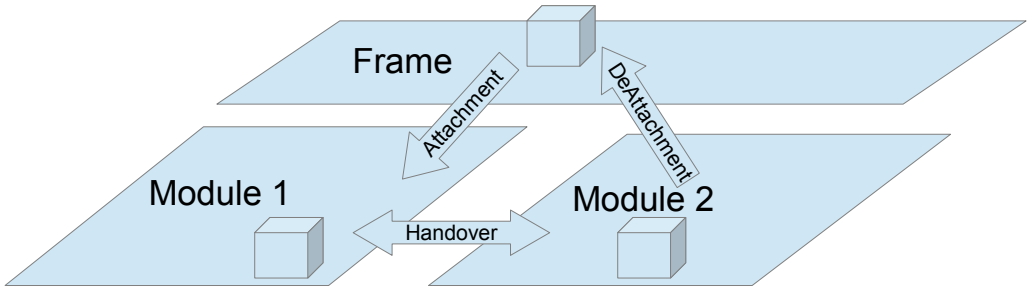

Abbildung 3: Ungebundene Objekte (Unbound Objects - UBOs)

Aus all diesen Überlegungen ergibt sich notwendigerweise die folgende Architektur für das SRR Framework:

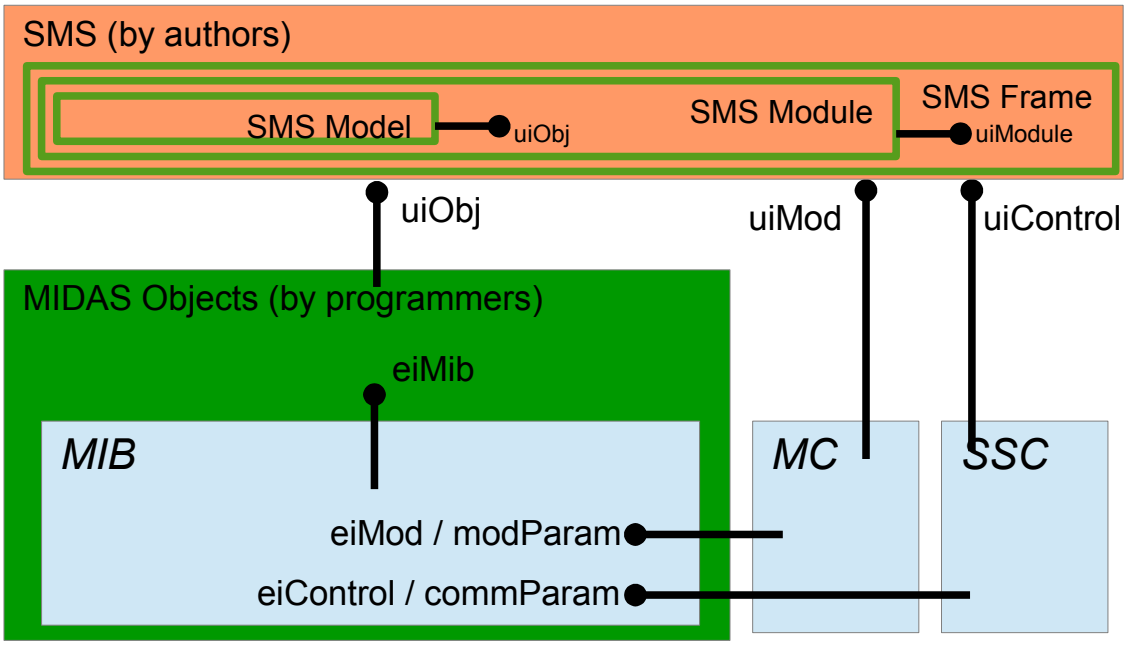

Abbildung 4: Die Teile des SRR Framework: SSC, MC und MIB

Der **Simple Scene Controller (SSC)** existiert genau einmal (pro Szeneninstanz) und wird innerhalb des Rahmens verwendet, um zentrale Eigenschaften der Simulation zu steuern.

Er wird vom Rahmen über das User Interface @uiControl verwendet und bietet seine Dienste auch den anderen Teilen der Szene über das externe Interface @eiControl an. Wir bezeichnen @eiControl deswegen als externes Interface, weil es von außerhalb des Subsystems "SimpleSceneController" verwendet werden kann.

Der **Module Coordinator (MC)** existiert genau einmal pro Modul (und pro Szeneninstanz). Er wird innerhalb jedes Moduls verwendet, um Modulparameter und alle Objekte des Moduls zu koordinieren.

Er wird vom Modul über das User Interface @uiMod verwendet und bietet seine Dienste auch den anderen Teilen des Moduls, also auch allen Objekten – nicht jedoch dem Rahmen – über das externe Interface @eiMod an. Wir bezeichnen @eiMod deswegen als externes Interface, weil es von außerhalb des Subsystems "ModuleCoordinator" verwendet werden kann.

Die **MIDAS Base (MIB)** hilft Programmierern dabei, MIDAS Objekte zu entwickeln.

Ein oder mehrere MIDAS Objekte können verwendet werden, um ein Modell zu instrumentieren. Sie bieten ihre Dienste über das User Interface @uiObj an, das immer einen gewissen Grundstock an Funktionalitäten bieten muss (sonst wäre das Objekt kein MIDAS Objekt), das aber je nach Typ das MIDAS Objekts beliebige zusätzliche Funktionen bieten kann (je nachdem, was der Programmierer programmiert hat).

Dieser Hibernation Report beschäftigt sich nun damit, welche Software-Instanzen es im SSC, im MC und in der MIB geben wird, nachdem der Step 0033.11 des Projekts SrrTrains v0.01 fertiggestellt sein wird.

3 SimpleSceneController – Instanzen und Interfaces

Im Subsystem "SimpleSceneController" gibt es mehrere Arten von Instanzen

- Singleton Instanzen (im Rahmen der Szeneninstanz)
- Instanzen "per registriertem Modul" (im Rahmen der Szeneninstanz)
- Instanzen "per UOC" (im Rahmen der Szeneninstanz)
- Instanzen "per Objekt" (in einem Objekt – genau genommen in einer Objektinstanz)

3.1 Singleton Instanzen

Sowohl der SSC Base als auch alle SSC Extensions sind Singleton Instanzen. Sie existieren nur einmal pro Szeneninstanz und enthalten zentrale Funktionen, die im Rahmen benötigt werden.

SSC Base:

"<SscInstance>" = "Ssc"

SSC Extension:

"<SscInstance>" = "<ParentClassPath>.<wki>

Beispiel: die "<SscInstance>" für den Train Manager lautet "Ssc.Base.SscTrainManager"

SSC Instanzen bieten jeweils ein @uiControl Interface und ein @eiControl Interface an.

3.2 "Per Modul" Instanzen

Der SSC Dispatcher, genau genommen der "Module Related SSC Dispatcher" ist zwar im Rahmen der Szeneninstanz enthalten – da er auch benötigt wird, wenn das entsprechende Modul (noch) nicht geladen ist –, er wird aber "per Modul" instanziiert und für gebundene Objekte verwendet.

"<DispInstance>" = "**Bdo**.<moduleName>"

Beispiel: "**Bdo**.City"

3.3 "Per UOC" Instanzen

Jede UOC benötigt genau einen SSC Dispatcher, genau genommen einen "UOC Related SSC Dispatcher". Auch dieser liegt im Rahmen der Szeneninstanz.

"<DispInstance>" = "**Uoc**.<uocName>"

Beispiel: "**Uoc**.Base.Keys"

3.4 "Per Objekt" Instanzen

Der SSC Dispatcher Stub, der es jedem Objekt ermöglicht, auf den richtigen SSC Dispatcher zuzugreifen, wird "per Objekt" und auch innerhalb des Objektes instanziiert.

"<StubInstance>" = "<extObjId>" = "<DispInstance>-<objId>"

Jeder Dispatcher Stub bietet ein @eiConsole Interface für ein bestimmtes Objekt an.

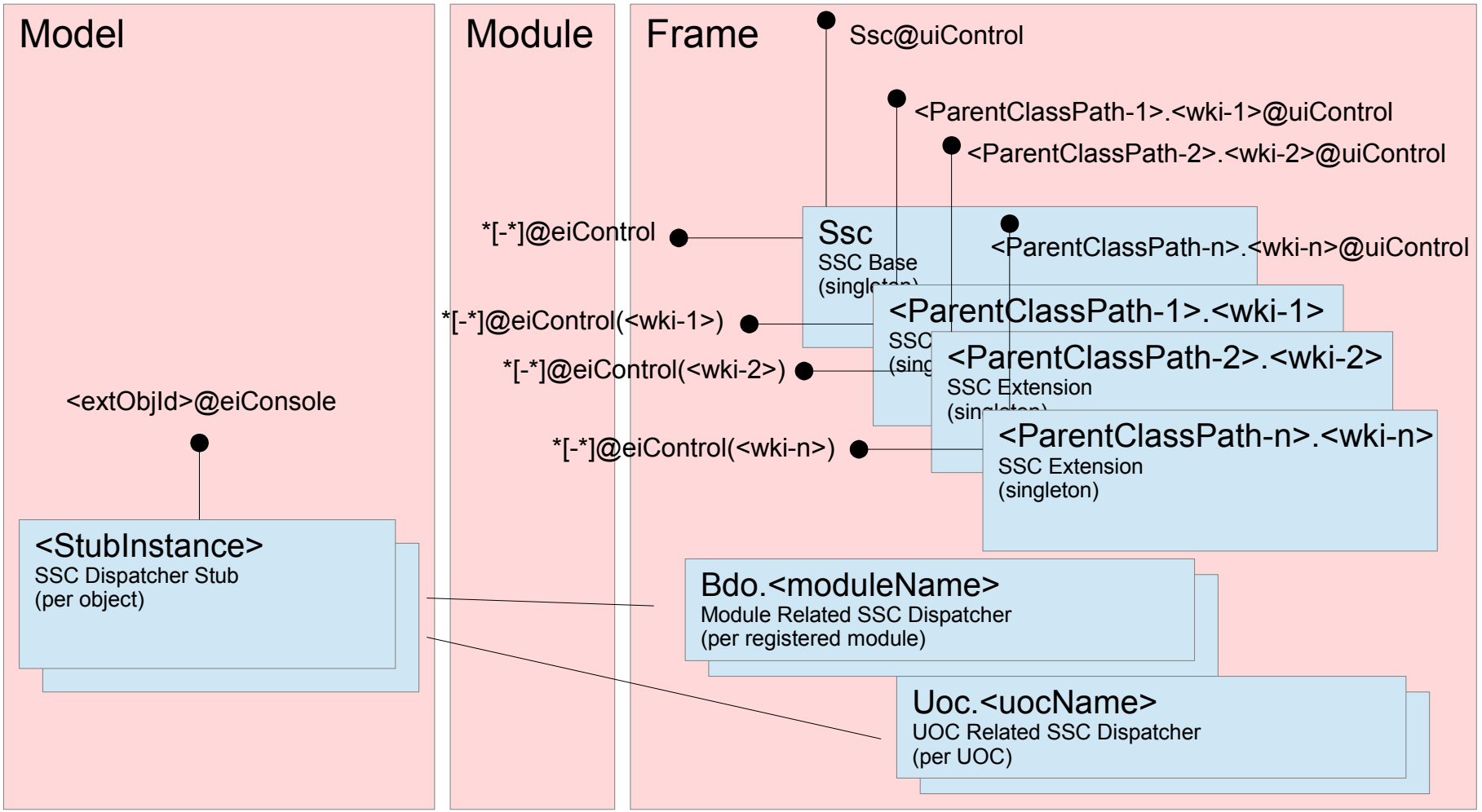

Abbildung 5: Subsystem "SimpleSceneController" - Instanzen und Interfaces

4 ModuleCoordinator – Instanzen und Interfaces

Im Subsystem "ModuleCoordinator" gibt es nur Instanzen, die

- per geladenem Modul

instanziiert werden.

Das sind also Instanzen des MC Base oder Instanzen von beliebigen MC Extensions.

"<McInstance>" = "**Mod**.<moduleName>-<McClassPath>"

Z.B. Würde man die Train Manager Extension im Modul "City" über

"<McInstance>" = "**Mod**.City-McBase.McTrainManager"

ansprechen.

Der MC Base im Modul "Hill" würde über

"<McInstance>" = "**Mod**.Hill-McBase"

angesprochen.

Der Modulkoordinator verwendet das @eiControl Interface des SimpleSceneController und bietet das @uiMod Interface und das @eiMod Interface

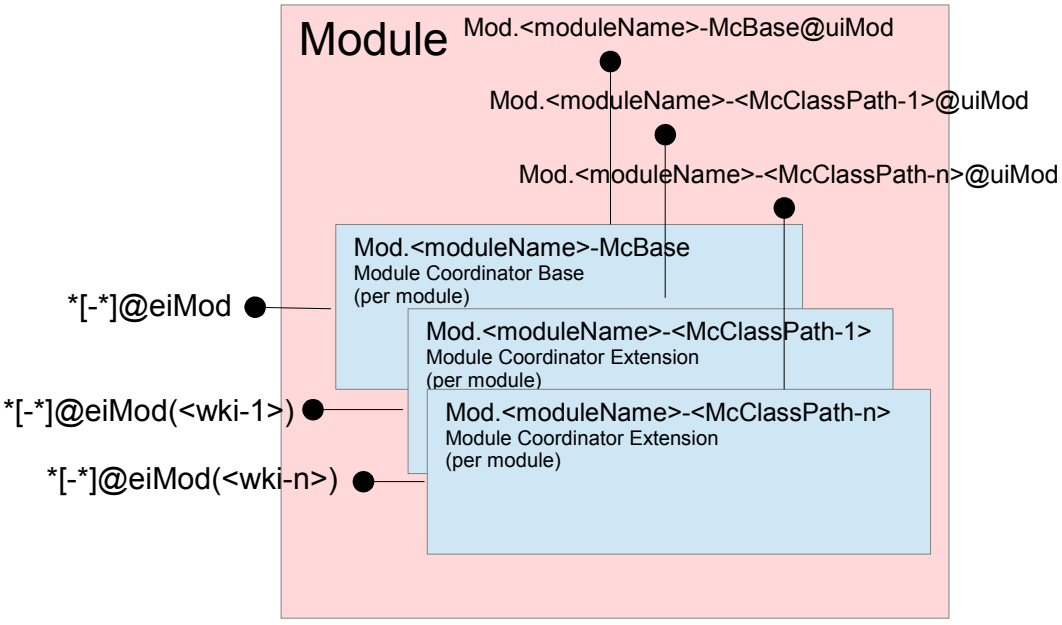

Abbildung 6: Subsystem "ModuleCoordinator" - Instanzen und Interfaces

5 MidasBase – Instanzen und Interfaces

In der MIDAS Base gibt es nur Instanzen, die "per Objekt" instanziiert werden. Diese gesellen sich zu der Instanz das SSC Dispatcher Stub, der auch "per Objekt" instanziiert wird.

Ein Objekt ist eindeutig durch eine Extended Object ID definiert, die sich wie folgt ergibt:

"<extObjId>" = "<DispInstance>-<objId>"

Die <objId> ist eindeutig innerhalb einer UOC oder innerhalb eines Moduls, wobei in der <DispInstance> durch den Präfix "Uoc." bzw. "Bdo." der Unterschied gemacht wird.

Eine UOC Related <DispInstance> wird für astrale Objekte und ungebundene Objekte verwendet, eine Module Related <DispInstance> für gebundene.

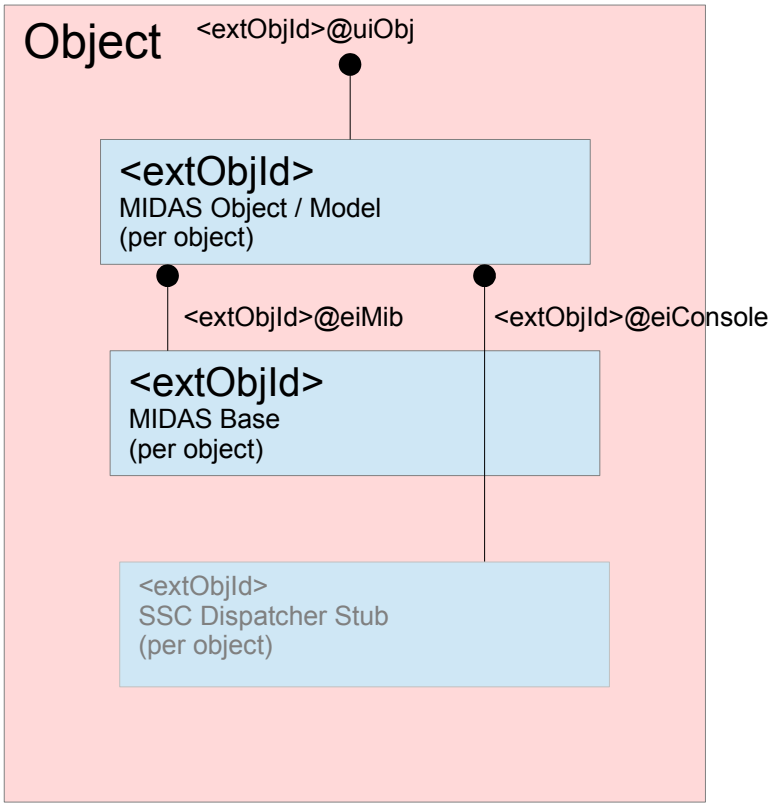

Abbildung 7: Instanzen und Interfaces, die es "per Objekt" gibt

6 Grundelemente für die "Neuen Namensregeln"

6.1 Bezeichnungen für Softwaretypen

6.1.1 "<wki>"

Eine "well known id" identifiziert eine SSC Extension oder eine MC Extension. <wki>s sind weltweit eindeutige Identifizierer von SMUOS basierter Software

Z.B. zeigt "<wki>" = "SscTrainManager" die Software des SSC Train Managers, der ein Teil des SRR Framework ist

6.1.2 "<wku>"

Eine "well known uoc" identifiziert eine UOC relativ zu der SSC Extension, die die UOC definiert.

Z.B. definiert "<wku>" = "Trains" die UOC "Trains" relativ zur Train Manager SSC Extension.

"SscTrainManager.Trains" ist also eine weltweit eindeutige Identifikation der UOC "Trains"

Hinweis: Die "<wku>" lässt sich zwar nicht ändern, der "<uocName>" (s.u.) kann aber in unterschiedlichen SMSn unterschiedlich sein (der Frame Author kann das beeinflussen)

6.1.3 "<SscClassPath>"

"<SscClassPath>" = "Ssc.Base[.<wki>.<wku>]..."

Der <SscClassPath> identifiziert eine UOC innerhalb einer konkreten Hierarchie von SSC Base und SSC Extensions in einer SMS.

Der <SscClassPath> kann als <ParentClassPath> in einer <SscInstance> verwendet werden (siehe dort).

In unserem Beispiel ist "<SscClassPath>" = "Ssc.Base.SscTrainManager.Trains" der <ParentClassPath> für die SSC Extension "Ssc.Base.SscTrainManager.Trains.FlyingTrainMgr" in dieser SMS.

6.1.4 "<McClassPath>"

"<McClassPath>" = "McBase[.<wki>]..."

Der <McClassPath> identifiziert entweder den MC Base oder den Typ einer MC Extension innerhalb einer konkreten Hierarchie von MC Base und MC Extensions.

Gemeinsam mit dem <moduleName> wird der <McClassPath> in einer <McInstance> verwendet, um eine Instanz des MC Base oder einer MC Extension zu identifizieren (siehe dort).

Z.B. könnte eine MC Extension in unserer Beispielszene den

"<McClassPath>" = "McBase.McTrainManager"

erhalten.

6.2 Bezeichnungen für Elemente der SMS nach dem MMF Paradigma

6.2.1 "<moduleName>"

Ein <moduleName> identifiziert ein Modul innerhalb einer Simple Multiuser Scene.

Beispiel:

"<moduleName>" = "City" ....."zweites Modul" des "offiziellen Demo Layout"

6.2.2 "<uocName>"

Ein <uocName> identifiziert eine Universal Object Class (UOC) innerhalb einer Simple Multiuser Scene.

Beispiel:

"<uocName>" = "Base".....die UOC "Base" des SSC Base
"<uocName>" = "Base.Keys".....eine UOC, die vom SscKeyManager verwaltet wird
"<uocName>" = "Base.Trains".....eine UOC mit UBO Loader im SscTrainManager

6.2.3 "<extObjId>", "<objId>"

Eine <extObjId> identifiziert ein Objekt (also ein MIDAS Objekt oder ein Modell). Die <objId> ist die lokale Objekt ID (innerhalb einer UOC oder innerhalb eines Moduls)

Gebundene Objekte:

"<extObjId>" = "<DispInstance>-<objId>" = "Bdo.<moduleName>-<objId>"

Astrale Objekte:

"<extObjId>" = "<DispInstance>-<objId>" = "Uoc.Base-<objId>"

Ungebundene Objekte:

"<extObjId>" = "<DispInstance>-<objId>" = "Uoc.<uocName>-<objId>"

6.3 Bezeichnungen für Softwareinstanzen

Instanzen sind Namen, die (1) im Tracer, (2) in Stream Names und (3) am Konsoleninterface verwendet werden, um Elemente der SMS zu identifizieren.

6.3.1 "<SscInstance>"

Eine <SscInstance> identifiziert entweder den SSC Base oder eine SSC Extension. Sie beginnt IMMER mit der Buchstabenfolge "**Ssc**".

SSC Base:

"<SscInstance>" = "Ssc"

SSC Extension:

"<SscInstance>" = "<ParentClassPath>.<wki>
Beispiel: die "<SscInstance>" für den Train Manager lautet "Ssc.Base.SscTrainManager"

6.3.2 "<DispInstance>"

Eine <DispInstance> identifiziert entweder einen UOC Related SSC Dispatcher oder einen Module Related SSC Dispatcher. Sie wird auch im enthaltenen UBO Loader verwendet

Module Related SSC Dispatcher:

"<DispInstance>" = "**Bdo**.<moduleName>"

Beispiel: "**Bdo**.City"

UOC Related SSC Dispatcher:

"<DispInstance>" = "**Uoc**.<uocName>"

Beispiel: "**Uoc**.Base.Keys"

6.3.3 "<McInstance>"

Eine <McInstance> identifiziert eine Instanz des MC Base oder eine Instanz einer MC Extension für ein konkretes Modul.

"<McInstance>" = "**Mod**.<moduleName>-<McClassPath>"

Z.B. Würde man die Train Manager Extension im Modul "City" über

"<McInstance>" = "**Mod**.City-McBase.McTrainManager"

ansprechen.

6.3.4 "<ObjInstance>"

Eine <ObjInstance> identifiziert eine Instanz eines MIDAS Objektes oder eines Modells.

Gebundene Objekte:

"<ObjInstance>" = "<extObjId>"

Beispiel: "**Bdo.City**-StationHouse.Door.Switch.Lock"

Astrale Objekte:

"<ObjInstance>" = "<extObjId>"

Beispiel: "**Uoc.Base**-DefAvaCon" ist der Default Avatar Container

Ungebundene Objekte:

"<ObjInstance>" = "<extObjId>"

Beispiel: "**Uoc.Base.Trains**-tgv0001"

7 Tracerinstanzen und -interfaces

Im Tracer gibt es einige Tracepunkte, für die die Benennung von Instanzen (<instance>) extrem wichtig ist. Die wichtigsten sind "messageReceived", "sendMessage", "eventReceived" und "sendEvent".

SMS:traceLevel=2,timeStamp=<timeStamp>,**ownSessionId=<sessionId>**
SMS: **instanceId=<instance>**,subsystem=<ss>,fileName=<fileName>,ssVersion=<ssVersion>
SMS: action=***messageReceived***;**sender=<sender>**;**messageType=<mt>**;messageId=<mid>
SMS: <free text from the programmer>
SMS:END

SMS:traceLevel=2,timeStamp=<timeStamp>,**ownSessionId=<sessionId>**
SMS: **instanceId=<instance>**,subsystem=<ss>,fileName=<fileName>,ssVersion=<ssVersion>
SMS: action=***sendMessage***;**receiver=<receiver>**;**messageType=<mt>**;messageId=<mid>
SMS: <free text from the programmer>
SMS:END

Eine Message wird also immer von einer <instance> innerhalb einer <sessionId> an eine <instance> innerhalb einer <sessionId> gesendet:

<sender> = <instance>@<sessionId>
<receiver> = <instance>@<sessionId>

Ein Event kann nur innerhalb einer <sessionId> von einer <instance> an eine andere <instance> versendet werden, wobei das <field> an beiden Enden denselben Namen hat.

Events zwischen zwei Subsystemen werden über sog. Interfaces (<ifInstance>@<ifName>) geroutet.

SMS:traceLevel=2,timeStamp=<timeStamp>,ownSessionId=<sessionId>
SMS: **instanceId=<instance>**,subsystem=<ss>,fileName=<fileName>,ssVersion=<ssVersion>
SMS: action=***sendEvent***;**destination=<destination>**;**field=<field>**
SMS: <free text from the programmer>
SMS:END

SMS:traceLevel=2,timeStamp=<timeStamp>,ownSessionId=<sessionId>
SMS: **instanceId=<instance>**,subsystem=<ss>,fileName=<fileName>,ssVersion=<ssVersion>
SMS: action=***eventReceived***;**origin=<origin>**;**field=<field>**
SMS: <free text from the programmer>
SMS:END

Es gilt

<origin> = <instance>
<destination> = <instance>

wenn Origin und Destination im selben Subsystem liegen, und es gilt

<origin> = <ifInstance>@<ifName>
<destination> = <ifInstance>@<ifName>,

wenn dem nicht der Fall ist.

Das soll in den folgenden zwei Unterkapiteln klar gemacht werden.

7.1 Tracerinstanzen

7.1.1 Tracerinstanzen für SSC Base und SSC Extensions

Controller / Central Controller:

"<SscInstance>.Control".....1x per Szeneninstanz (singleton)
"<SscInstance>.CentralSrv"....1x per Multiuser Session, verwaltet CommState/Global State

7.1.2 Tracerinstanzen für SSC Dispatcher und SSC UBO Loader

Client / Server / UOC Server:

"<DispInstance>.Client".....1x per Szeneninstanz und per Modul/UOC
"<DispInstance>.Server".....1x per Szeneninstanz und Modul/UOC, verarb. SMS Requests
"<DispInstance>.UocSrv"....1x per Multiuser Session und Modul/UOC, verw. UBO State

7.1.3 Tracerinstanzen für MC Base und MC Extensions

Coordinator:

"<McInstance>.Coord".....1x per Szeneninstanz und per geladenem Modul, koordiniert

7.1.4 Tracerinstanzen für Objekte

Client Software / Object Controller / Console Server

"<ObjInstance>.Control".....1x per Szeneninstanz und geladenem Objekt
"<ObjInstance>.ObCo".....1x per Multiuser Session und Objekt, verwaltet Global State
"<ObjInstance>.Server".....1x per Szeneninstanz und Objekt, verarb. SMS Requests

7.2 Tracerinterfaces

7.2.1 @commParam Interface

[-]@commParam	SscBase verteilt Trace Levels und Events an Avatare
<mcInstance>@commParam	McBase received Trace Levels vom SscBase
<objInstance>@commParam	MoosAvatarContainer received Events vom SscBase

7.2.2 @eiControl Interface

[-]@eiControl	SscBase empfängt Request von irgendwo [und antwortet]
<mcInstance>@eiControl	Austausch zwischen SscBase und McBase
<objInstance>@eiControl	Austausch zwischen SscBase und einigen MIDAS Objekten: MoosAvatarContainer, MoosCreator, MoosSwitchA, MoosSwitchB

7.2.3 @commParam(<wki>) Interface

[-]@commParam(<wki>)	SscBeamerManager verteilt Beamer Destinations
<mcInstance>@commParam(<wki>)	wird nicht verwendet (receive)
<objInstance>@commParam(<wki>)	MoosBeamer (received Beamer Destinations)

7.2.4 @eiControl(<wki>) Interface

[-]@eiControl(<wki>)	wird nicht verwendet (Request von irgendwo)
<mcInstance>@eiControl(<wki>)	Austausch zwischen SrrControlTm und SrrModCoordTm
<objInstance>@eiControl(<wki>)	Austausch zwischen SscBeamerManager / SscKeyManager / SrrControlTm und einigen MIDAS Objekten MoosBeamer / MoosBeamerDestination / MoosKeyContainer / MoosLockA / MoosLockB / SrrReplicator

7.2.5 @modParam Interface

Module.<moduleName>-*@modParam	McBase verteilt MAM
Module.<moduleName>-<objId>@modParam	MibAnim / MibNoStateOsm/ MibStandardOsm empfangen die MAM

7.2.6 @eiMod Interface

Module.<moduleName>-*@eiMod	McBase empfängt Request von irgendwo [und antwortet]

7.2.7 @modParam(<wki>) Interface

Wird zur Zeit nicht verwendet

7.2.8 @eiMod(<wki>) Interface

<objInstance>@eiMod(<wki>) Austausch zwischen SrrModCoordTm und den MIDAS Objekten SrrBasicTrackSection / SrrBasicTurnout2Way / SrrTrackEdge / SrrTrackNode

7.2.9 @eiConsole Interface

<objInstance>@eiConsole Austausch zwischen SscDispatcherStub und den Benützern SscBase / SscKeyManager / MoosCreator / MoosDriveA / MoosKeyContainer / MoosLockB / MoosSwitchA / MoosSwitchB

7.2.10 @eiMib Interface

<objInstance>@eiMib benützung von MIB durch ein MOB

7.2.11 @uiObj Interface

<objInstance>@uiObj wird angeboten von:
MibCore / MoosAvatarContainer / MoosBeamer / MoosBeamerDestination / MoosCreator / MoosDriveA/ MoosKeyContainer / MoosLockA / MoosLockB / MoosSwitchA / MoosSwitchB / MoosTrigger / SrrBasicTrackSection / SrrBasicTurnoutr2Way / SrrReplicator / SrrTrackEdge / SrrTrackNode
wird benützt von
SscBase (Avatar Container) / Model / Module / Frame

7.2.12 @uiMod Interface

<mcInstance>@uiMod wird angeboten von:
McBase / SrrModCoordTm
wird benpützt von:
Module

7.2.13 @uiModule Interface

Module.<moduleName>@uiModule wird benützt von:
SmsModuleLoader / Module Wrapper / Frame

7.2.14 @uiControl Interface

<sscInstance>@uiControl wird benützt von: Frame

8 StreamNames

Stream Names haben die generelle Form

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

wobei <Suffix> vom Programmierer frei gewählt werden kann (alles ausser Punkte und Gedankenstriche)

<User> wird vom SMUOS Framework vorgegeben und wird zusammen mit dem <mainIdentifier> in den nächsten Kapiteln erklärt

8.1 SSC Base und SSC Extensions

"<mainIdentifier>" = "<SscInstance>"

"<User>" = "Base", wenn es ein Network Sensor des SSC Base ist und

"<User>" = "Ext", wenn es ein Network Sensor einer SSC Extension ist

8.2 SSC Dispatcher und SSC UBO Loader

"<mainIdentifier>" = "<DispInstance>"

"<User>" = "SmsDispatcher", wenn es ein Network Sensor des SSC Dispatcher ist und

"<User>" = "UboLoader", wenn es ein Network Sensor des SSC UBO Loader ist

8.3 Objekte

Streamnames:

"<mainIdentifier>" = "<ObjInstance>"

"<User>" = "Obj", wenn es ein Network Sensor des MIDAS Objektes oder des Modells ist

"<User>" = "Mib", wenn es ein Network Sensor der MIDAS Base ist

9 Konsoleninterface

<DispInstance>-<objId>-<parameterName>