Hibernation of Advanced Railroad Trains (ArrT)

Schritt Eins – Ordnung machen und sich entspannen

MIDAS Best Current Practices

Dieser Hibernation Report ist ein "Snapshot" und wird also nicht weiter upgedatet werden.

1 Übersicht "Wie bastle ich ein MIDAS Objekt"

Um eigene MIDAS Objekte zu basteln, muss man sich an einige Regeln halten. Einerseits können MIDAS Objekte die unteren Schichten des SMS Frameworks **benützen**, also

- den Modulkoordinator MC (über das externe Interface eiMod),
- den Simple Scene Controller SSC (über das externe Interface eiControl) und
- den SSC Dispatcher DISP (über das externe Interface eiDisp).

Andererseits verlangt das SMS Framework, dass MIDAS Objekte gewisse Dienste **anbieten**,

- das ist die Definition des User Interfaces uiObj.

Bei all diesen Aufgaben hilft die MidasBase dem Programmierer des MIDAS Objekts (MOB).

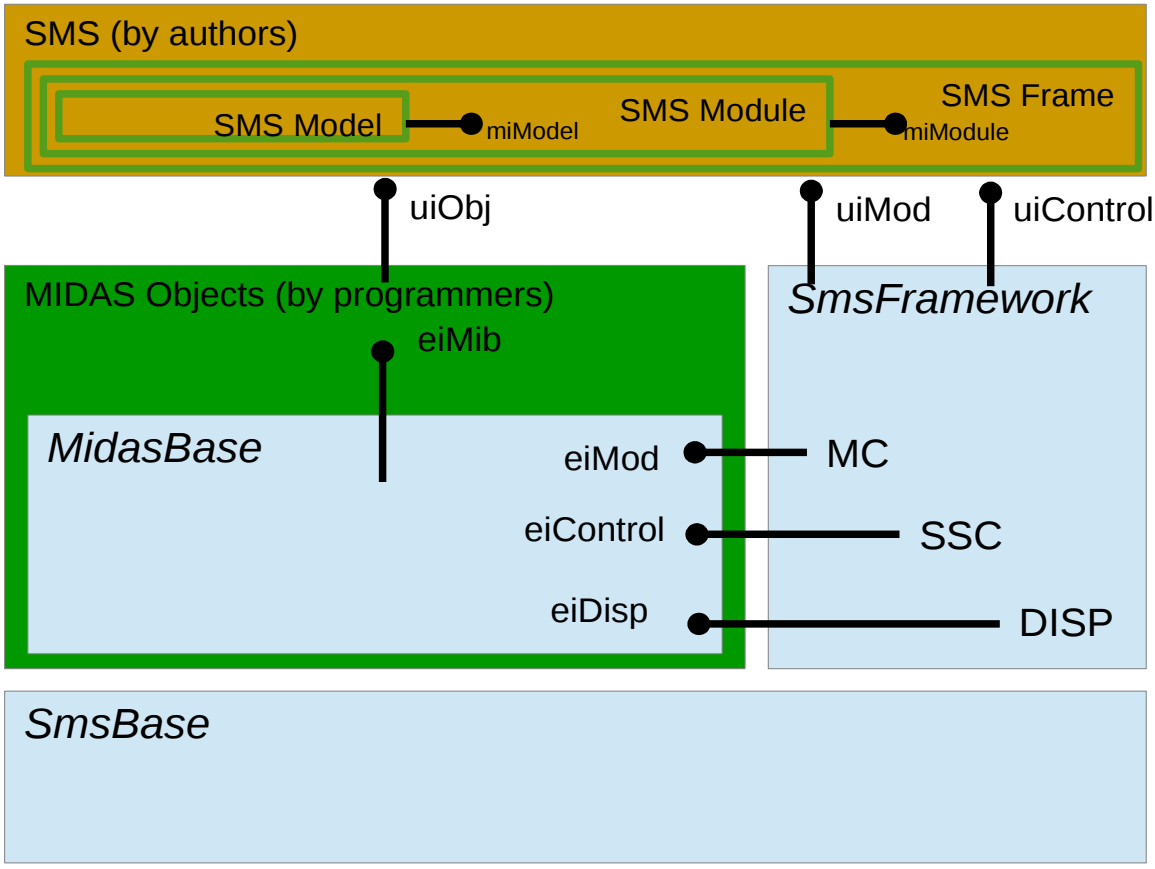

Abbildung 1: MidasBase als Bindeglied zwischen SMS Framework und MIDAS Objekt

2 Was zum Kuckuck sind MIDAS Objekte?

Na gut, da gibt es einerseits die historische Erklärung:

- Im Anfang entschied ich mich das SIMUL-RR Projekt (damals hieß es noch nicht SrrTrains v0.01) auf den Netzwerksensor aufzubauen
- Ich musste um die Netzwerksensoren zweier Hersteller eigene Prototypen herumbasteln, da diese von den beiden Firmen je unterschiedlich definiert worden waren, ich aber Szenen basteln wollte, die auf mehr als einem Web3D Browser laufen sollten (wozu denn sonst den ganzen Overhead mit einem ISO Standard, wenn ich mal fragen darf)
- Diese selbstgebastelten Prototypen bekamen im Laufe der Zeit immer mehr Funktionen, sodass aus ihnen ein eigenes Konzept wurde
- Sie wurden die sogenannten **SRR Objekte** (Simulated Railroad Objects)
- Dann irgendwann machte ich aus dem Base Module des **SRR Frameworks** (das war eine Ansammlung von Prototypen, die die SRR Objekte unterstützen sollten) ein eigenes Projekt und führte einige Umbenennungen durch:
    - aus dem "SRR Controller" wurde der "Simple Scene Controller"
    - aus dem "SRR Module Coordinator" wurde der "SMS Module Coordinator"
    - aus dem Base Module des SRR Framework wurde das SMUOS Framework
    - dabei wurden auch die SRR Objekte umbenannt auf MIDAS Objekte

OK, aber wenn ich nun so ein MIDAS Objekt erwerbe, was kann ich denn damit tun?

- Nun, angenommen, Du bist ein Web3D Autor und möchtest das Modell eines Autos bauen
- Weiter angenommen, dieses Modell soll in einer Multiuserszene von einem anderen Autor lauffähig sein
- DANN wirst Du doch nach Möglichkeiten suchen, die gesamte Multiuserproblematik auszulagern. Du wirst doch Wege erachten einfach das Modell eines Autos zu basteln ohne Rücksicht, ob die Szene multiuserfähig ist oder nicht
- DANN wirst Du doch nach fertigen Mechanismen suchen, um den Antrieb, die Lenkung und die Bremse des Autos zu simulieren, und Du wirst Dir die ganze Mathematik eigentlich ersparen wollen
- UND GENAU DAS bieten die MIDAS Objekte
- MIDAS Objekte sind hochspezialisierte Kraftpakete, die die gesamte Mathematik der Multiusersimulation im Griff haben
- MIDAS Objekte lassen sich aber nicht rendern, sie bleiben unsichtbar, unhörbar, unfühlbar, die Äußerlichkeiten, das "look and feel" Deiner Modelle liegt nach wie vor komplett in Deiner Hand
- MIDAS Objekte sind die "invisible engines" Deiner multiuserfähigen, interaktiven und animierten Modelle

3 Verhalten von MIDAS Objekten

3.1 Initialisierung mit den Common Parameters

Wegen des "MMF-Paradigmas", an das wir uns halten wollen, besteht die Szene aus einzelnen Modulen (das kann man sich so ähnlich vorstellen wie Landschaftskacheln) und diese Module werden von Modellen "bewohnt".

Die Modelle wiederum enthalten die **MIDAS Objekte (MOBs)**.

Aber es gibt auch ein – zugegebenerweise ein bisschen abstraktes – "Drumherum", ein "Etwas", das benötigt wird, um die Module zusammenzuhalten und in einer Szene "aufzuhängen".

Dieses "Etwas" nennen wir den "Rahmen".

Jetzt wissen wir auch, warum das Paradigma "MMF-Paradigma" heisst, denn es geht um **M**odelle in **M**odulen in einem **F**rame (Rahmen).

Da der Rahmen etwas Grundlegendes ist, enthält er auch einen grundlegenden Teil des SMS Frameworks, nämlich den **Simple Scene Controller (SSC)**.

Auf den Simple Scene Controller können alle Teile des SRR Frameworks und auch die MIDAS Objekte zugreifen.

Damit alle Teile der Szene auf den SSC zugreifen können, publiziert dieser nach seiner Initialisierung einen Pointer auf die **Common Parameters (commParam)**.

Um ein MIDAS Objekt zu initialisieren, leitet man ihm diesen Pointer weiter.

Mit Hilfe dieses Pointers

- kann jedes MIDAS Objekt auf die Felder des SSC zugreifen,
- kann sich jedes MIDAS Objekt für die Ereignisse anmelden, die der SSC über die commParam broadcastet.

Auf manche MIDAS Objekte muss der SSC direkt und einzeln zugreifen. Das heisst, er benötigt Pointer auf diese Objekte. Das wird im Kapitel 3.4 "Announcement beim Simple Scene Controller" beschrieben.

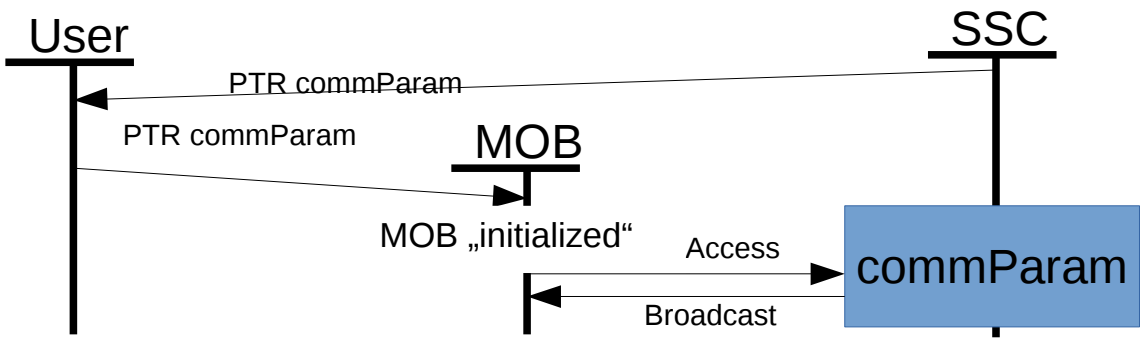

Abbildung 2: Access to the SSC by an initialized MOB

3.1.1 Fehlerfall: Fehler während der Initialisierung

Wie wir im Kapitel 3.3 "Die Felder "initialized" und "attached" sowie "getScript"" sehen werden, gibt ein MIDAS Objekt nach seiner Initialisierung ein Feed Back mit dem Feld "initialized".

Wenn nun während der Initialisierung ein Fehler auftritt, sendet das MOB in diesem Feld den Wert NULL. Trotzdem gilt die Initialisierung als durchgeführt (das MOB befindet sich im Status "fail-initialized") und wird durch eine DeInitialisierung rückgängig gemacht.

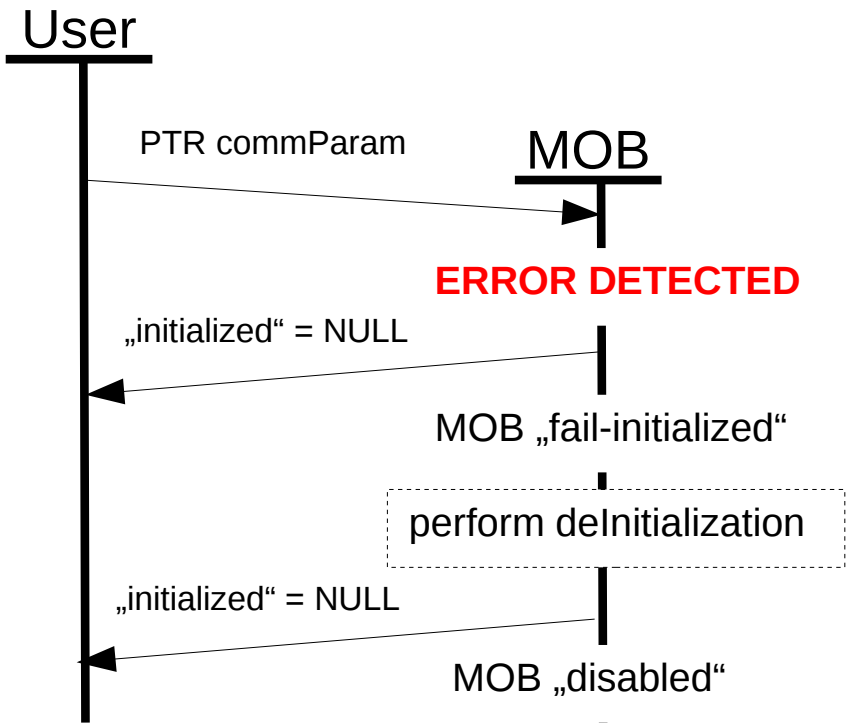

Abbildung 3: Initialization: error during initialization

3.2 Attachment mit den Modulparametern

Jedes Modul enthält eine Instanz des **Modulkoordinators (MC)**. Dieser wird nach dem SSC initialisiert, indem man ihm einen Pointer auf die **Common Parameters (commParam)** überreicht und announced sich dann beim SSC (damit der SSC einzeln und direkt auf alle MCs zugreifen kann).

Nachdem das Modul auf diese Weise den Zustand "attached" erreicht hat, veröffentlicht der MC einen Pointer auf die **Modulparameter (modParam)**.

Diesen Pointer leitet man an ein **MIDAS Objekt (MOB)** weiter, damit man es am MC attacht.

Mit Hilfe dieses Pointers

- kann jedes MIDAS Objekt auf die Felder seines MCs zugreifen
- kann sich jedes MIDAS Objekt für die Ereignisse anmelden, die der MC über die modParam broadcastet

Auf manche MIDAS Objekte muss der MC einzeln und direkt zugreifen. Das heisst, er benötigt Pointer auf diese Objekte. Das wird im Kapitel 3.5 "Announcement beim Module Coordinator" beschrieben.

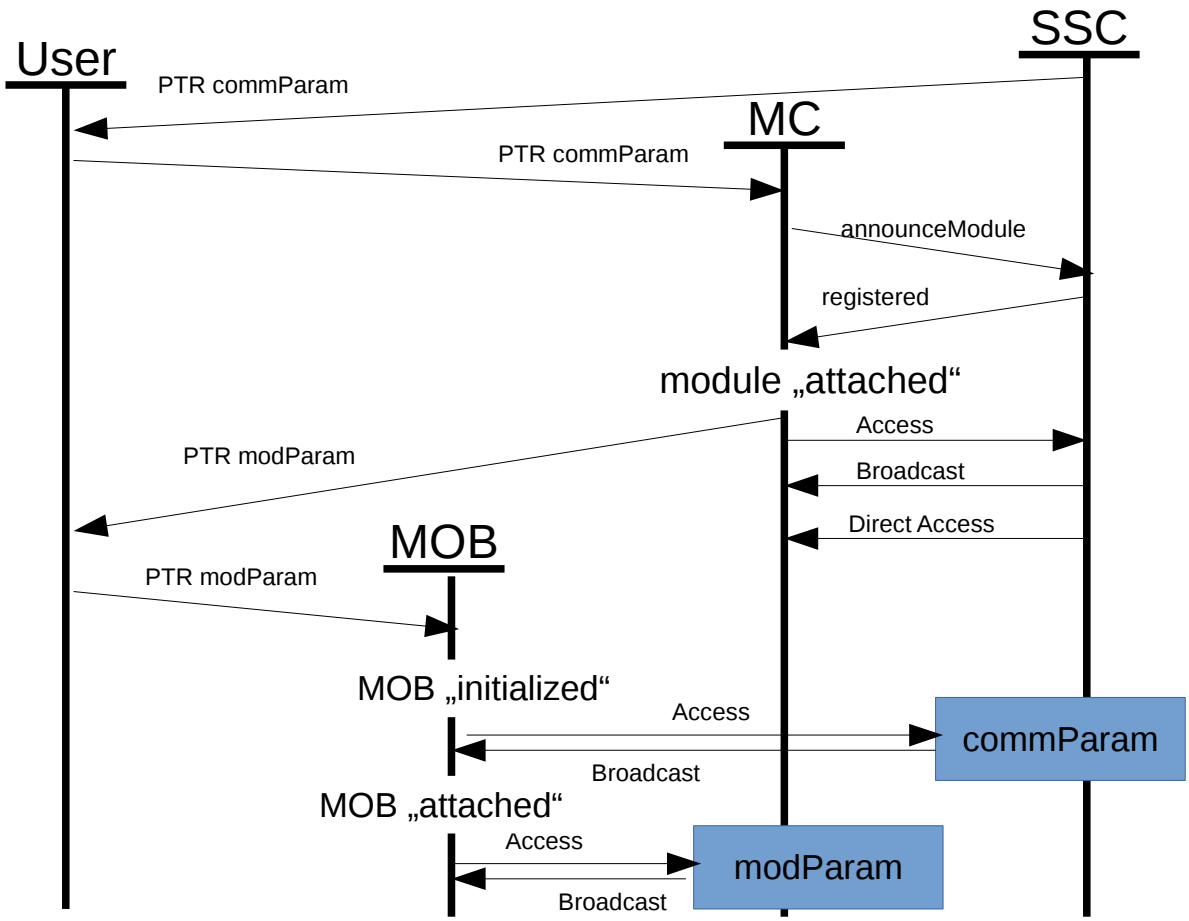

Abbildung 4: Access to the MC by an attached MOB

3.2.1 Fehlerfall: Fehler während der Initialisierung

Wie wir im Kapitel 3.3 "Die Felder "initialized" und "attached" sowie "getScript"" sehen werden, gibt ein MIDAS Objekt nach seiner Initialisierung ein Feed Back mit dem Feld "initialized" und nach seinem Attachment ein Feed Back mit dem Feld "attached".

Wenn nun während der Initialisierung im Rahmen eines Attachments ein Fehler auftritt, sendet das MOB im Feld "initialized" den Wert NULL. Trotzdem wird auch das Attachment weitergeführt (das MOB befindet sich dann im Status "fail-attached") und danach durch ein DeAttachment und eine DeInitialisierung wieder rückgängig gemacht.

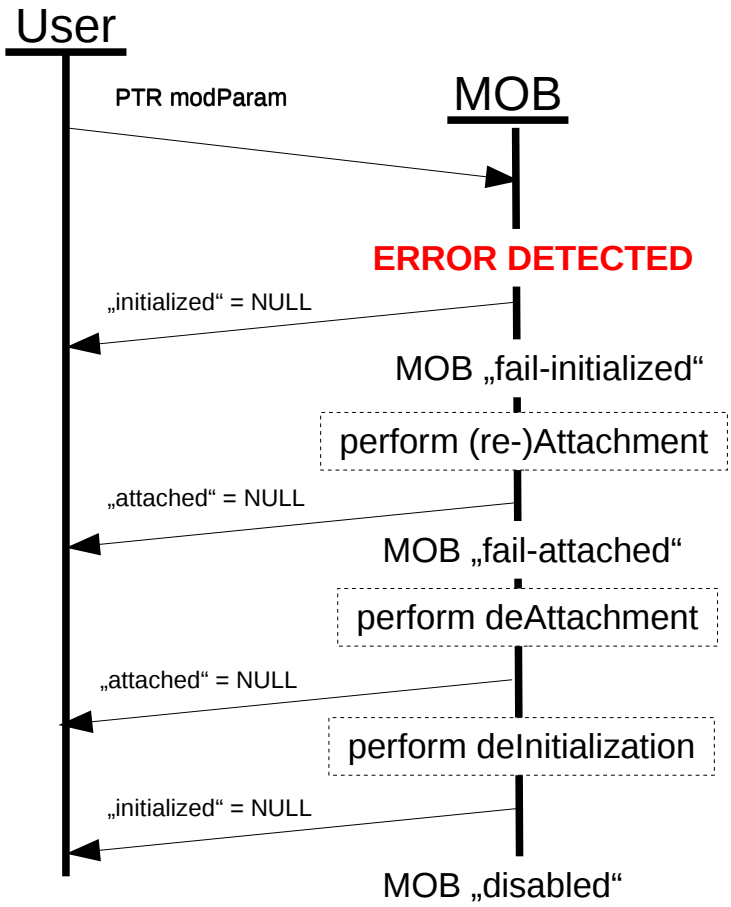

Abbildung 5: Attachment: error during initialization

3.2.2 Fehlerfall: Fehler während des Attachments

Wie wir im Kapitel 3.3 "Die Felder "initialized" und "attached" sowie "getScript"" sehen werden, gibt ein MIDAS Objekt nach seiner Initialisierung ein Feed Back mit dem Feld "initialized" und nach seinem Attachment ein Feed Back mit dem Feld "attached".

Wenn nun während des Attachments ein Fehler auftritt, sendet das MOB im Feld "attached" den Wert NULL. Trotzdem wird das Attachment weitergeführt (das MOB befindet sich dann im Status "fail-attached") und danach durch ein DeAttachment und eine DeInitialisierung wieder rückgängig gemacht.

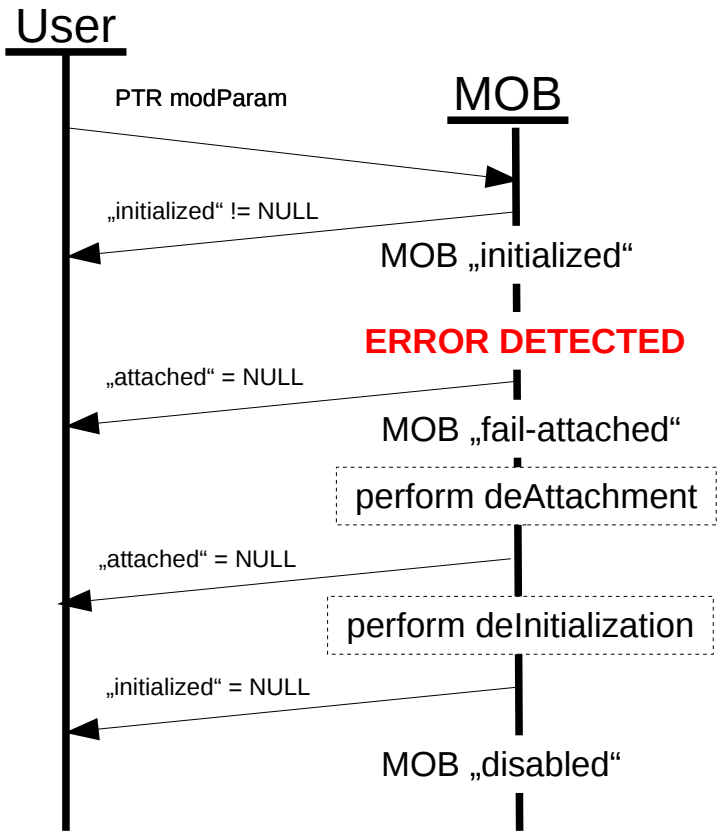

Abbildung 6: Attachment: error during attachment

3.3 Die Felder "initialized" und "attached" sowie "getScript"

3.3.1 "getScript"

Jedes MIDAS Objekt enthält (mindestens) einen <Script>-Knoten, in dem die Funktionalität des Objekts prozedural beschrieben ist.

Manche dieser Prozeduren müssen das MIDAS Objekt einem anderen Knoten der Szene zur Kenntnis bringen.

Dazu wäre es gut, wenn es innerhalb eines <Script>-Knotens einen Befehl gäbe, der so ähnlich wie SELF in anderen Programmiersprachen auf die umgebende <ProtoInstance> verweist.

Solch einen Mechanismus habe ich in X3D nicht gefunden, aber zumindest gibt es die Möglichkeit, einen Pointer auf den <Script>-Knoten selbst im Sinne von SELF zu verwenden.

Wir definieren also: Innerhalb des SMS Frameworks wird ein MIDAS Objekt durch einen SFNode Wert repräsentiert, der auf den im MIDAS Objekt enthaltenen <Script>-Knoten zeigt. Das gilt auch für andere <ProtoInstance>s, die im Rahmen des SMS Frameworks verwendet werden.

Am besten sieht man das an einem Beispiel: der Modulkoordinator enthält folgenden <Script>-Knoten (auszugsweise dargestellt):

```
        <Script DEF='SmsModCoordScript' directOutput='true' mustEvaluate='true'>
          <field accessType='inputOutput' name='commParam' type='SFNode' value="NULL"/>
          <field accessType="inputOutput" name="getScript" type="SFNode">
            <Script USE="SmsModCoordScript"/>
          </field>
.....
<![CDATA[
ecmascript:
.....
// Announcement protection timer expired (fires isActive = false)
function timerExpired(Value,timestamp)
{
.....
    commParam.sscBase.announceModule = getScript;
.....
}
]]>
        </Script>
```

Hier sieht man, dass der <Script>-Knoten in seinen Feldern die Werte "commParam" und "getScript" speichert. Dabei zeigt "commParam" auf die Common Parameter und "getScript" ist ein Pointer auf den <Script>-Knoten selbst.

Der Code "commParam.sscBase.announceModule = getScript" sendet also einen Pointer auf den <Script>-Knoten des Modulkoordinators zum Feld "announceModule" des <Script>-Knotens, auf den das Feld commParam.sscBase zeigt.

3.3.2 "initialized" und "attached"

Damit nun auch der Benutzer des MIDAS Objekts den Pointer auf das MIDAS Objekt speichern kann, um ihn dann später zum Beispiel gegenüber dem SSC als Identifizierung des MIDAS Objektes zu verwenden, wird definiert:

1. Ein MIDAS Objekt, das eine erfolgreiche (Re-)Initialisierung durchlaufen hat, muss am Feld "initialized" einen Pointer auf den internen <Script>-Knoten ausgeben, der auch gegenüber dem MC, dem SSC und dem DISP verwendet wird.
2. Ein MIDAS Objekt, das eine DeInitialisierung durchlaufen oder eine erfolglose (Re-)Initialisierung hinter sich hat, muss am Feld "initialized" den Wert NULL ausgeben.
3. Ein MIDAS Objekt, das ein erfolgreiches (Re-)Attachment durchlaufen hat, muss am Feld "attached" einen Pointer auf den internen <Script>-Knoten ausgeben, der auch gegenüber dem MC, dem SSC und dem DISP verwendet wird.
4. Ein MIDAS Objekt, das ein DeAttachment durchlaufen oder eine erfolgloses (Re-)Attachment hinter sich hat, muss am Feld "attached" den Wert NULL ausgeben.
5. Dieser "SELF"-Wert wird auch verwendet, wenn sich ein MIDAS Objekt beim MC, beim SSC oder beim DISP "announced" (ankündigt).

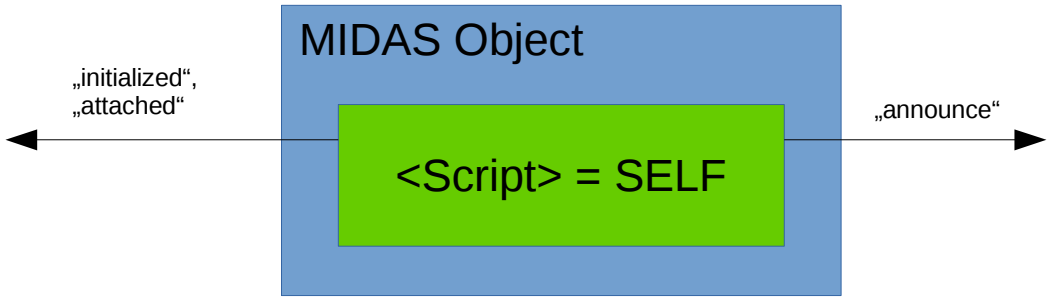

Abbildung 7: Verwendung des <Script>-Knotens für SELF

3.4 Announcement beim Simple Scene Controller

Ein MIDAS Objekt, das sich beim SSC ankündigt (announcet), sollte dies während seiner (Re-)Initialisierung durchführen.

Dadurch ist gewährleistet, dass der SSC das Objekt bereits kennt, wenn es seinen "SELF"-Pointer am "initialized"-Feld ausgibt.

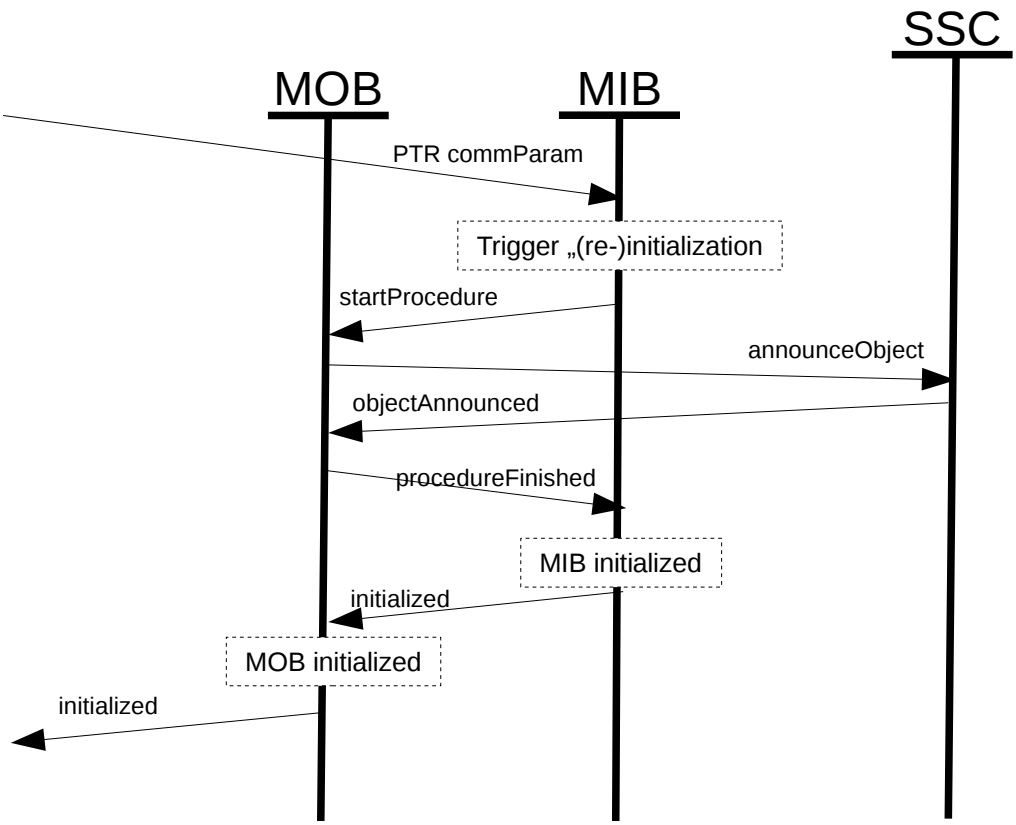

Abbildung 8: (Re-)Announcement eines Objektes während der (Re-)Initialisierung

3.5 Announcement beim Module Coordinator

Ein MIDAS Objekt, das sich beim MC ankündigt (announcet), sollte dies während seines (Re-)Attachments durchführen.

Dadurch ist gewährleistet, dass der MC das Objekt bereits kennt, wenn es seinen "SELF"-Pointer am "attached"-Feld ausgibt.

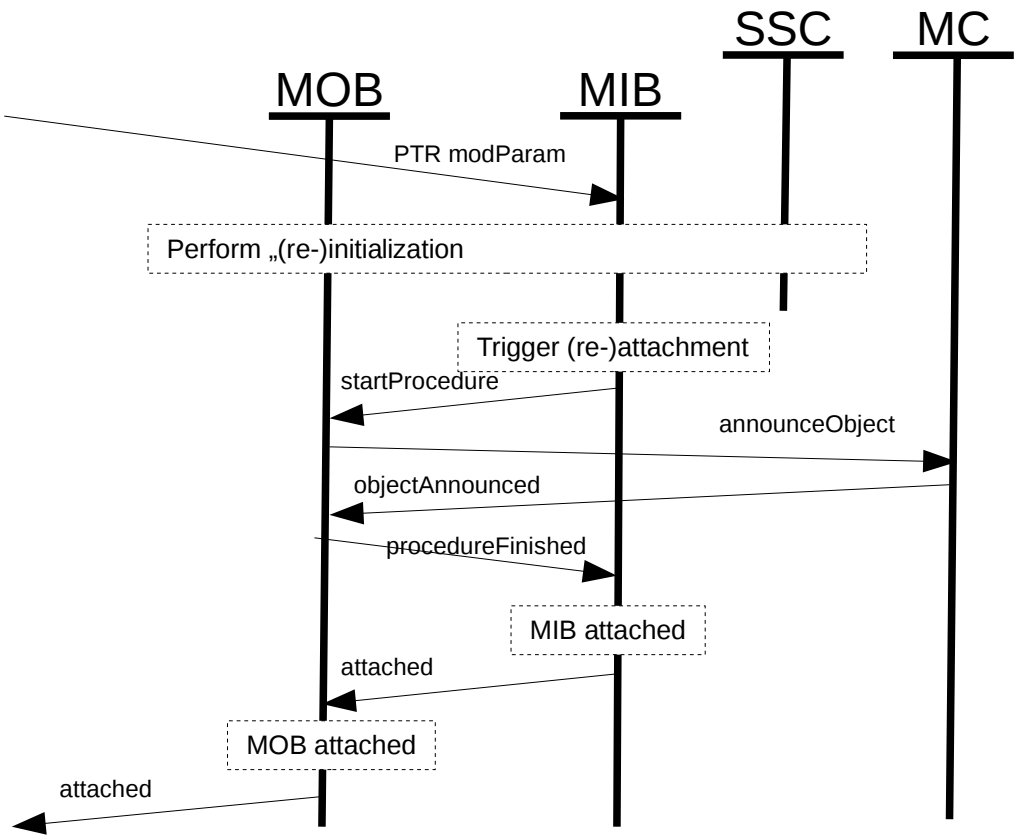

Abbildung 9: (Re-)Announcement eines Objektes während des (Re-)Attachments

3.6 Verschachtelte MIDAS Objekte

Prinzipiell sorgt die MIDAS Base (MIB) dafür, dass alle MIDAS Objekte (MOBs) eines Modells immer denselben Mode of Operation (MOO) innehaben.

Im besonderen ist dies eine Aufgabe des MibCore Prototypen:

- Dependent MOBs, die zum Zeitpunkt der "basic initialization" im Feld "dependentMobsIn" referenziert sind, werden mit addMob() hinzugefügt. Das heisst, sie werden in der "internen Liste der dependent MOBs" aufgenommen und bei der ersten Initialization oder beim Wechsel in MOO V wird dafür gesorgt, dass sie den gleichen MOO haben, wie das Elternobjekt.
- Auch mit addDependentMobsIn() kann man ein oder mehrere MOBs als dependent MOBs hinzufügen. Das ist immer dann möglich, wenn gerade keine Prozedur am Laufen ist. Nach dem Hinzufügen des MIDAS Objektes muss man am Elternobjekt einen MOO Change starten, damit das Kindobjekt denselben MOO erhält. Das geschieht am Anfang der (Re-)Initialisierung des Elternobjektes oder nach der DeInitialisierung des Elternobjektes.
- Durch das Entfernen eines oder mehrerer dependent MOBs mit removeDependentMobsIn() - was immer möglich ist, wenn gerade keine Prozedur am Laufen ist - werden diese MOBs aus der "internen Liste der dependent MOBs" entfernt und danach werden sie automatisch mit "disable" in den MOO V versetzt. Eventuelle eigene Pointer auf das MOB muss der User selbst entfernen.
- Wenn ein Kindobjekt in der "internen Liste der dependent MOBs" steht und aus irgendwelchen Gründen disablet worden ist ("enabledOut" feuert den Wert "false"), dann wird auch das Elternobjekt disablet. Beim Elternobjekt gilt dies als sogenannter "interner Trigger für ein Disabling".
- Ein "interner Trigger für Disabling" führt dazu, dass einem eventuell vorhandenen eigenen Elternobjekt sofort(!) mitgeteilt wird, sich auch zu disablen, ohne auf den Trigger "enabledOut" = "false" des Kindobjektes zu warten.
- Es gibt folgende "interne Trigger für Disabling"
  - Ein Kindobjekt feuert "enabledOut" = "false"
  - Ein Netzwerksensor meldet die fehlerhafte Initialisierung
  - Ein Kindobjekt meldet einen "internen Trigger für Disabling"
- Am Ende eines MOO Changes in den MOO V (nachdem es "enabledOut" = "false" gefeuert hat) leitet jedes MIDAS Objekt einen Trigger an den "disable" Eingang aller Kindobjekte.