SrrTrains v0.01 (pre-alpha)
Neues Konzeptpapier

Oktober 2015, C.Valentin

Inhaltsverzeichnis

0. Links....4
1. DIGITS und SIMUL-RR – wie alles begann....4
    1.1. DIGITS....4
    1.2. SIMUL-RR....5
    1.3. Weitere Herangehensweisen....5
2. Das Rollercoaster Projekt – Gleisgeometrie und Animation....6
3. Ziele sind nötig – das eierlegende-Wollmilchsau-Problem....8
4. Der Simple Scene Controller (SSC)....9
    4.1. Der Netzwerksensor....9
    4.2. Client Based Server Software (CBSS)....11
    4.3. Der "BIMPF Approach"....12
    4.4. Der Simple Scene Controller....13
5. Module und Objekte....14
    5.1. Einleitung....14
    5.2. Das MMF-Paradigma....14
    5.3. Initialization und Attachment....15
    5.4. Die MAM und der MOC....16
    5.5. Die OBCO Rolle....17
    Weitere Stichworte....17
6. Der Tracer....19
7. Das Konsoleninterface....20
8. Dynamische Modelle....21
9. Die erste LAN Party und der "BIMPF Approach"....22
10. Dynamische Module....23
11. Die MIDAS Base (MIB)....24
12. Der "SMUOS/C3P Approach" – Ausblick in die Zukunft....25
    12.1. Die SMUOS Komponente....25
    12.2. Das Collaborative 3D Profile (C3P)....25
    12.3. Mögliche Schritte, um SMUOS/C3P zu erreichen....30
13. DIGITS, SMUOS and the Theory of Relativity....31
    13.1. Use Cases, The Scope of a SMUOS/SMS....31
    13.2. New Software Structure of a SMUOS/SMS....32
Anhang A – Weiterführende Konzepte....33
    A.1. Handover....33
    A.2. Moving modules....33
    A.3. Real multibrowser capability....33
    A.4. Authoring support....33
    A.5. Rebase SRR Framework to additional MU Systems....33
Anhang B – Ein möglicher Projektplan....34
Anhang C – Die Modes of Operation (MOOs)....40
    C.1. Die MOOs des SSC....40
    C.2. Die MOOs des MC....40
    C.3. Die MOOs eines Objektes....41
Anhang D – Die OBCO State Machine....43
Anhang E – Analogien zwischen 3D Szene und Realität (Engl.)....44
Anhang F – Semaphoren und Controller Roles (Englisch)....45
    F.1. References....45
    F.2. Summary....45
    F.3. Motivation....45

F.4. Proposal....46
F.5. Examples....47
Anhang G – Trying a Prophecy about SrrTrains v0.01 – Principle Timeline....50

0. Links

1. SrrTrains "Hibernation Page" (currently the master page):
   http://letztersein.wordpress.com/srrtrains-v0-01/
2. The Simulated Railroad Framework Project
   http://simulrr.sourceforge.net/
3. The Simple Multiuser Online Scenes Project
   http://smuos.sourceforge.net/

1. DIGITS und SIMUL-RR – wie alles begann

1.1. DIGITS

Die Erfindungsmeldung "DIGITS" aus dem Jahre 2002

1.1.1. Digitus = der Finger

Die Frage aller Fragen: Wieviele Finger haben UFO-Piloten?

1.1.2. Digit = Ziffer (Teil einer Telefonnummer)

Telefonnummern sind geographisch angeordnet (zumindest war das so in Zeiten des Festnetzes).

DIGITS beschäftigt sich mit geographischer Infrastruktur.

1.1.3. DIGITS = Distributed Internet Geographic Information Transmission Service

Was wäre, wenn man den Begriff "ins Internet einsteigen" wortwörtlich auffassen würde?

Hat die Idee des Holodecks eine tatsächliche Zukunft?

Wo endet der Use Case und wo beginnt die "geographische Infrastruktur"? Muss man Objekte nicht so gestalten, dass sie sich leicht vom einen ins andere verschieben lassen, sobald sie abgeschrieben sind?

Angenommen, wir hätten eine verteilte Datenbank, die interaktive, animierte 3D-Landkarten enthält, müsste man nicht über folgende Keys auf die Datenbank zugreifen?

- Virtual Roaming Area (ein Polygon in WGS84-Koordinaten)
- Level of Detail (in welcher Stufe der mehrstufigen Datenbank beginne ich mit der Abfrage?)
- Reality (eine flexible Möglichkeit, das Ergebnis zu modifizieren, z.B. Für mehrsprachige Ortstafeln oder angepasste Werbeeinschaltungen)

Siehe die nächste Figure 1

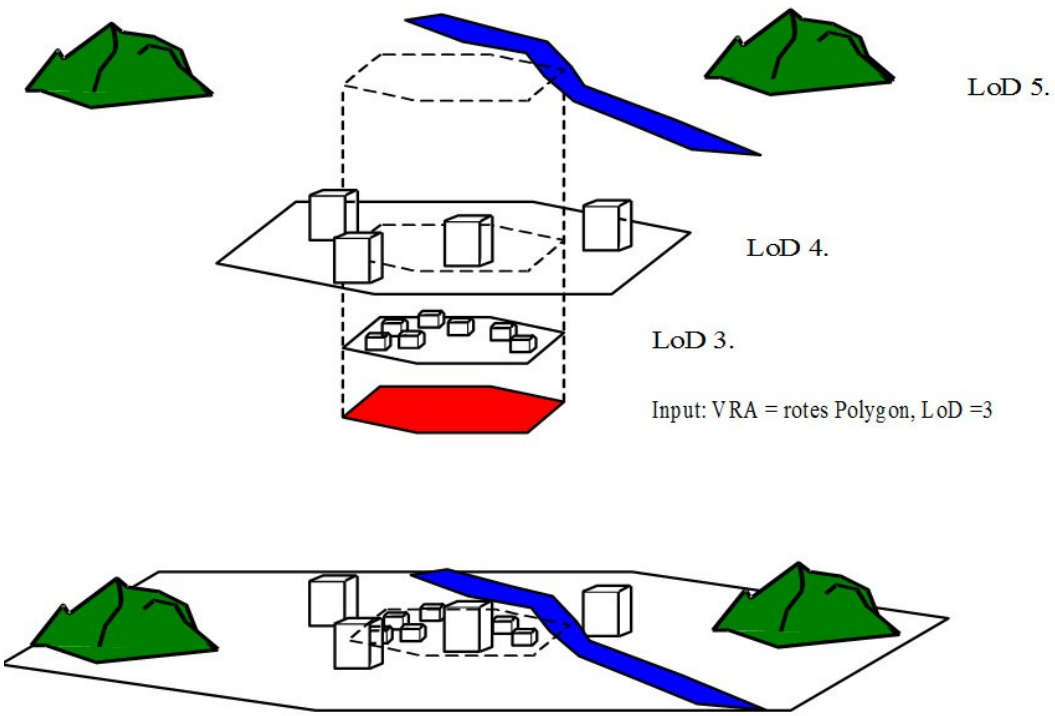


Figure 1: Zusammensetzung eines Ausschnitts der Welt nach VRA und LoD

1.2. SIMUL-RR

Die Innovationsidee "SIMUL-RR" aus dem Jahre 2007/2008

Eine multiuserfähige virtuelle Modelleisenbahn für Hobbyisten, am besten verknüpft mit einer realen Modelleisenbahn.

1.3. Weitere Herangehensweisen

Glaubst Du an Engel?

Hat die Liebe einen Platz im Universum?

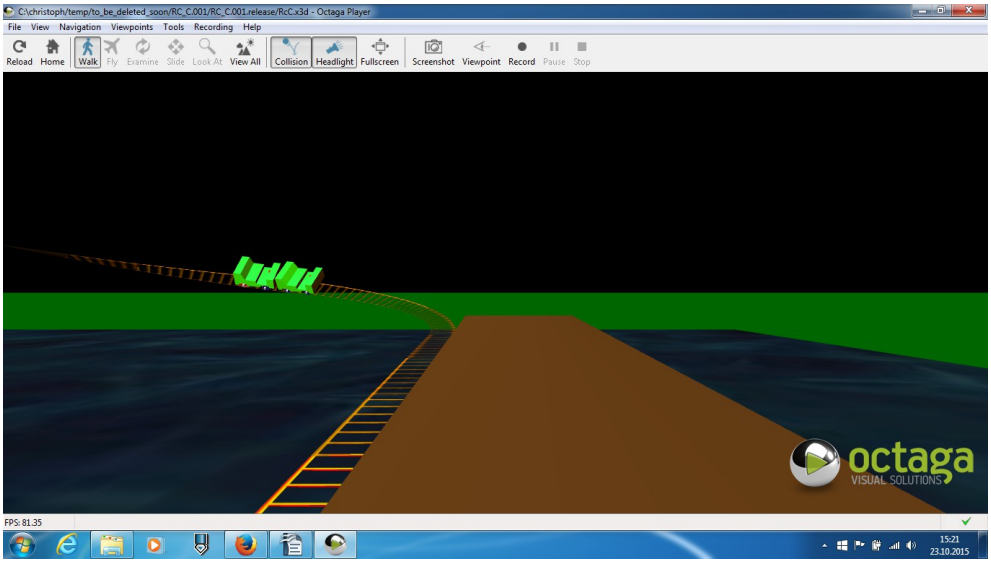
Warum man sich mit Ideen beschäftigen sollte, für die die Geldwelt nichts übrig hat, obwohl die Ideenwelt eindeutig in diese Richtung tendiert.

Die Ideenwelt ist der Geldwelt um Jahrhunderte voraus.

2. Das Rollercoaster Projekt – Gleisgeometrie und Animation

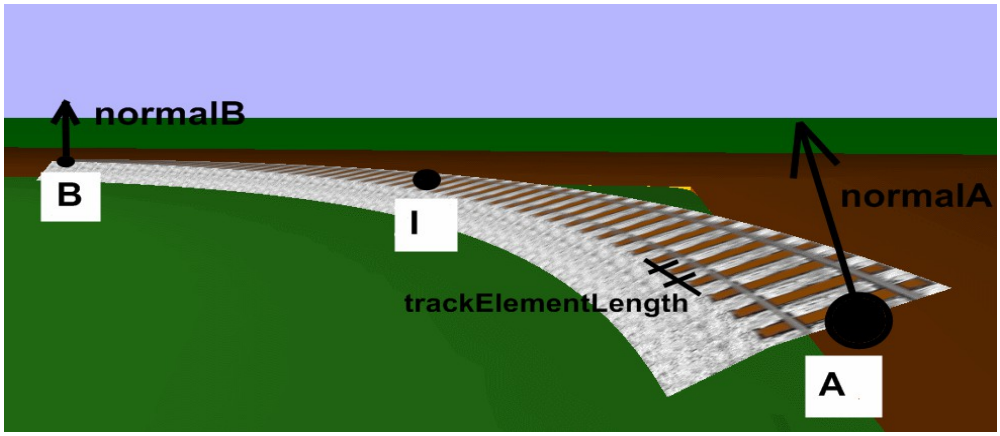
Die engere Wahl für eine Technologie für das Simulated Railroad Projekt fiel auf X3D, also konnte ich beginnen mich einmal mit Gleisgeometrie und "Animation entlang von Schienen" zu beschäftigen.

Da ein Rollercoaster offensichtlich stärkere Anforderungen an die Gleisgeometrie stellt als eine Eisenbahn, begann ich mit dem "Rollercoaster Projekt" ca. zu Ostern 2008.

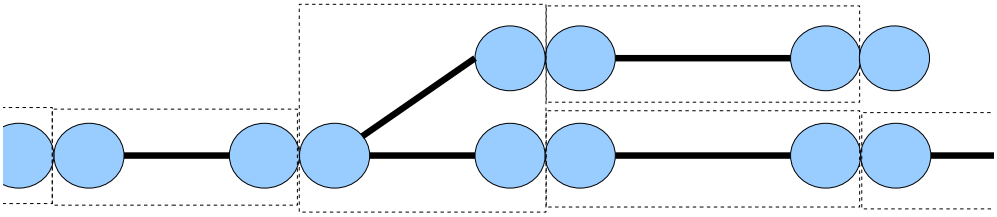


Ich entwickelte die "ABI" Gleisgeometrie, in der jeder Abschnitt des Gleissystems aus einem (räumlichen) Kreisabschnitt besteht.

Dieser Kreisabschnitt wird aus der Angabe eines Startpunktes (Punkt "A"), eines Endpunktes (Punkt "B") und eines Zwischenpunktes (Punkt "I") berechnet:



Die Gleisabschnitte werden in einem Doppelpunktgraphen miteinander verknüpft, sodass sich komplexe Gleisgeometrien mit Kreuzungen, Weichen, Abstellgleisen usw. ergeben.



3. Ziele sind nötig – das eierlegende-Wollmilchsau-Problem

- Dieses Kapitel beschreibt die Ziele des SrrTrains v0.01 Projektes

Am 6.1.2015 habe ich mir jetzt Gedanken zu meinen persönlichen Motivationen gemacht, und tatsächlich sind folgende drei Motive aufgetaucht:

1. Ich würde gerne eine Community gründen, die sich mit der "Do-it-yourself-virtual-multiplayer-model-railroad" beschäftigt
2. Ich würde gerne im Web3D Consortium bzw. in der IETF Beiträge leisten, um ein "fully standardized 3D multiplayer system based on the Web3D Network Sensor" zu unterstützen
3. Ich habe auch eine "hidden agenda", die sich natürlich mit GSM-R beschäftigt

Aber man muss gar nicht psychologisch argumentieren, es reicht, aus den "Concepts' Descriptions" die "Master Concepts" zu zitieren, die ja längst veröffentlicht sind:

Zitat:

[...]The concept SrrTrains is built by only a few "master concepts", everything else is derived from these master concepts:

1. build multiplayer virtual model railroads based on VRML/X3D
2. make it an open concept where many people can contribute their ideas and sweat
3. use open source software as far as useful and possible
4. players can assume so-called "roles" (engineer, shunter, ...)
5. players can communicate during the game (chat, voice-chat, ...)
6. a layout and its models can be influenced by a command line interface
7. authors can (easily) build models of rail vehicles
8. authors can (easily) build models of houses and other static objects
9. layouts are built by modules (from different authors)
10. mechanisms exist to restrict usage of your models and modules[...]

4. Der Simple Scene Controller (SSC)

- Dieses Kapitel beschreibt
  - den Netzwerksensor
  - Client Based Server Software (CBSS)
  - den "BIMPF Approach"
  - den Simple Scene Controller

Als ich im Jahr 2009 auf der X3D-public Mailing Liste fragte, welche standardisierten Möglichkeiten es gäbe, um 3D Multiuser Szenen zu bauen, bekam ich zwei Möglichkeiten vorgeschlagen:

1. Distributed Interactive Simulation (DIS)
2. den Netzwerksensor

Nach einem kurzen Studium der DIS Komponente im X3D Standard entschied ich mich, auf den Netzwerksensor zu setzen, da dieser nach meiner Meinung das generellere Interface darstellt und somit mehr Möglichkeiten offen läßt.

4.1. Der Netzwerksensor

Der Netzwerksensor ist ein X3D-Knoten mit frei definierbaren Feldern.

Diese Felder sind entweder Eingabe- oder Ausgabefelder, um Werte über einen Collaboration Server zu allen anderen Szeneninstanzen zu senden.

- Mit einem evt_<name> Feld kann man einen Event zum Collaboration Server senden, der diese Events dann an alle Szeneninstanzen verteilt, wo sie in einem <name>_evt Feld empfangen werden. Events werden einfach nur verteilt, ohne dass sie gespeichert werden
- Mit einem set_<name> Feld kann man einen State am Collaboration Server setzen. Diese States werden am Collaboration Server gespeichert und an alle Szeneninstanzen verteilt, wo sie in einem <name>_changed Feld empfangen werden.
- States können nicht nur mit einem set_<name> Feld gesetzt sondern auch modifiziert werden, z.b. mit einem add_<name> oder mit einem sub_<name> Feld
- Wenn ein Netzwerksensor initialisiert wird, feuern alle <name>_changed Felder den aktuellen Wert des States (falls bereits vorhanden).

Am problematischesten ist die Verwendung der set_<name> Felder, da dadurch ein State einen komplett neuen Wert bekommt. Dieser neue Wert sollte sich immer auf den alten Wert beziehen, es sei denn der State wird zurückgesetzt. Nun kann es aber durch Verzögerungen im Netzwerk dazu kommen, dass in unterschiedlichen Szeneninstanzen zur selben Zeit unterschiedliche Werte des States vorliegen, weshalb es nie eindeutig geklärt ist, auf welchen alten Wert sich ein set_<name> Kommando bezieht.

Diese Nicht-Kausalitäten lassen sich eigentlich nur beheben, indem man immer eine Szeneninstanz definiert, die das alleinige Recht hat, set_<name> Kommandos zu geben.

Die folgenden Abbildungen zeigen einige Beispiele von Verläufen mit States und Events

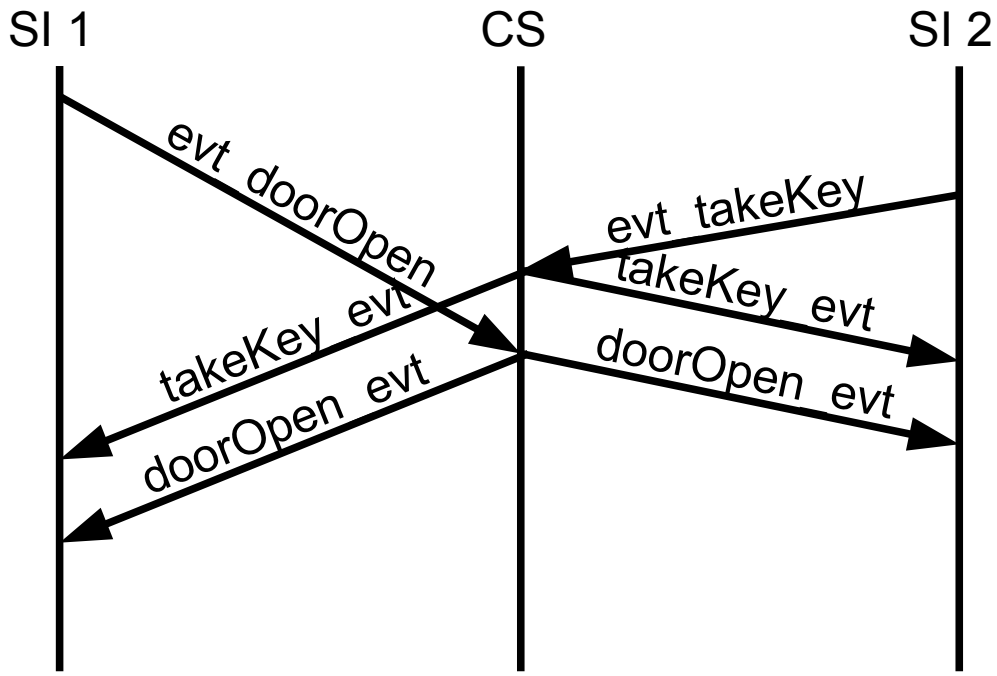


Abbildung 1: Verteilung von Events

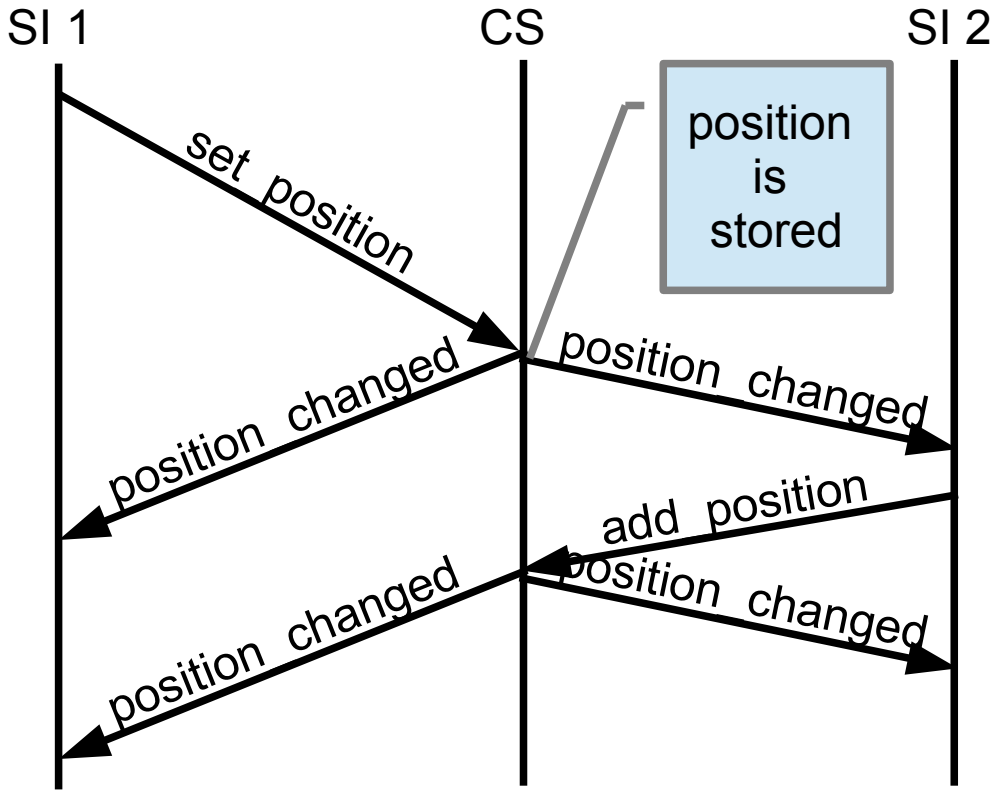


Abbildung 2: Behandlung von States

4.2. Client Based Server Software (CBSS)

Im letzten Abschnitt haben wir gesehen, wie States am Collaboration Server gespeichert werden und wie man diese States in weiterer Folge modifizieren kann, indem man sich auf den vorherigen Wert bezieht.

Nun ist es so, dass wir im SrrTrains v0.01 Projekt nach Möglichkeiten gesucht haben, wie man diese Modifizierung der States nicht am Server durchführt, sondern am Client.

Wir wollten möglichst flexible Server-Software schreiben – zum Beispiel in Form von X3D <Script>-Knoten –, ohne den Server ändern zu müssen.

Die Lösung fand sich in dem, was wir als "Client Based Server Software" (CBSS) bezeichnen, das heisst:

1. Zu jedem Netzwerksensor wird eine Szeneninstanz definiert, der es erlaubt ist, den set_<name> Befehl zu verwenden, um States basierend auf ihrem vorherigen Wert zu modifizieren. Wir nennen diese Szeneninstanz den "Controller"
2. Es kann zu jedem Zeitpunkt höchstens einen Controller zu jedem Netzwerksensor geben
3. Zeiträume, in denen ein Netzwerksensor keinen Controller hat, sind erlaubt

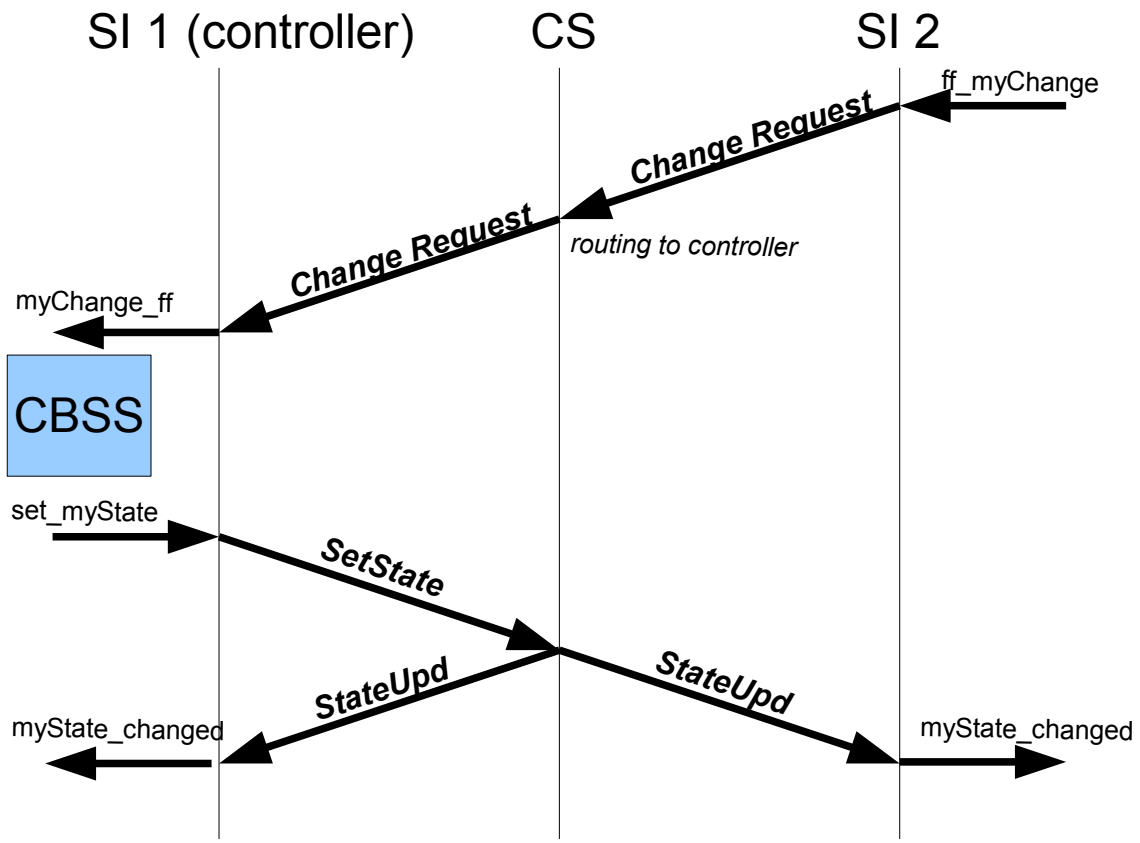


Figure 2: Principle of CBSS with enhanced network sensor

4.3. Der "BIMPF Approach"

Der Ansatz für CBSS, der im letzten Abschnitt beschrieben ist, wäre nur gangbar, wenn man den Netzwerksensor um die Funktionalität "Support of Controller Roles" ergänzte.

Im SrrTrains v0.01 Projekt mussten wir einen anderen Weg gehen, da wir den Netzwerksensor unverändert benützen.

Wir machten Versuche mit

- BS Contact und BS Collaborate und mit
- Octaga Collaboration Server.

Da beide Produkte ihre Netzwerksensoren leicht unterschiedlich implementiert hatten, mussten wir Netzwerksensor-Wrapper einführen (das waren X3D Prototypen, die die Unterschiede der beiden Netzwerksensor-Produkte versteckten).

Dies war sogar ein glücklicher Zufall, denn daraus ergab sich letzten Endes

- Die Keimzelle für die MIDAS Objekte (siehe Kapitel "11. Die MIDAS Base (MIB)")
- Die Namensgebung "BIMPF" (Browser Independent Multiplayer Framework)

Im "BIMPF Approach" konnten wir die "Change Request" Events nicht vom Server zum Controller routen lassen, sondern wir mussten die Events zu allen Szeneninstanzen routen lassen und dort erst entscheiden, ob wir sie verarbeiten oder ignorieren (je nachdem, welche Szeneninstanz die Controller-Rolle innehatte).

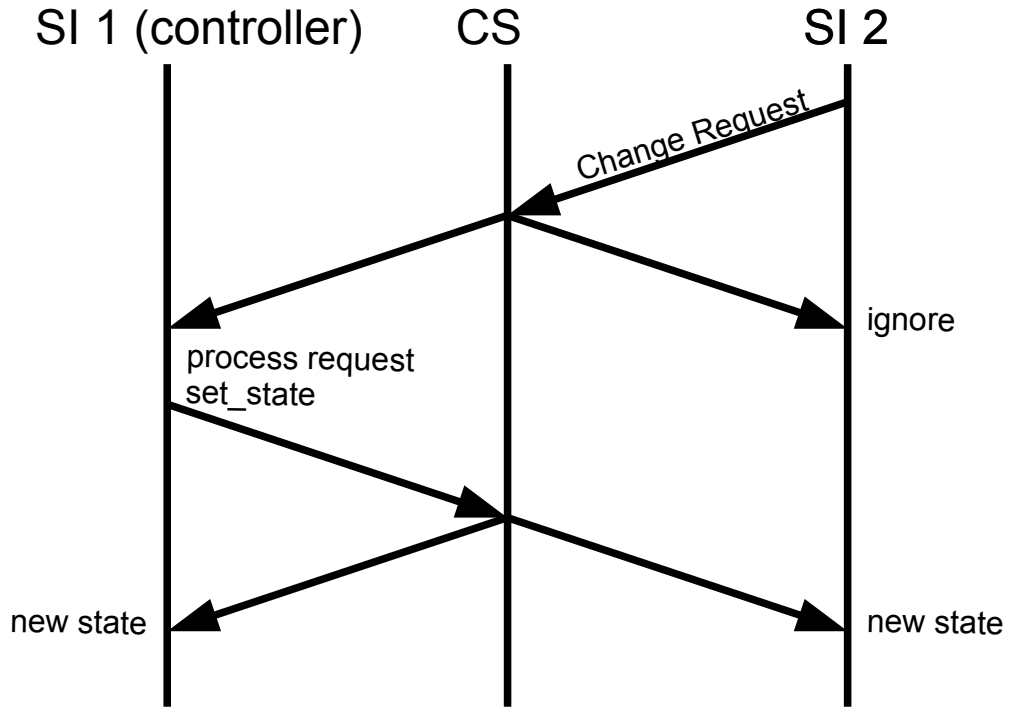


Abbildung 3: State Änderungen beim BIMPF Approach

4.4. Der Simple Scene Controller

Der Simple Scene Controller war nun der erste X3D Prototyp, an dem ich den "BIMPF Approach" ausprobieren konnte.

Der Simple Scene Controller wird in jeder Szeneninstanz genau einmal instanziiert und initialisiert, nachdem man den <BSCollaborate> - Knoten initialisiert hat.

Schauen wir in den "Concepts' Descriptions" nach, wozu man den Simple Scene Controller – sozusagen das zentrale Interface des SRR Frameworks – verwenden kann:

1. initialize the SRR/SMUOS Framework in multi-user-mode or in single-user-mode
2. add/remove avatars
3. report avatar position/orientation to the frame in relative coordinates
4. request the controller role
5. register/deregister modules explicitely
6. activate/deactivate modules
7. request MOC roles
8. set trace levels and output tracer output
9. display carried keys **)
10. put carried keys into key containers or locks **)
11. reset all keys **)
12. send/receive console commands/responses
13. display all beamer destinations *)
14. bind one beamer destination *)
15. join a remote user
16. output short status messages

*) In step 0033.05, the former SRR Controller was split into a base module and two extension modules. These functions (marked with an *)) are provided by the first SSC extension (the SSC "Beamer Manager" SscBeamerManager.x3d).

**) In step 0033.05, the former SRR Controller was split into a base module and two extension modules. These functions (marked with an **)) are provided by the second SSC extension (the SSC "Key Manager" SscKeyManager.x3d).

5. Module und Objekte

- Dieses Kapitel beschreibt
  Module und Objekte, MIDAS Objekte
  das MMF-Paradigma
  die Module Activity Matrix (MAM) und den Module Controller (MOC)
  den Object Controller (OBCO)

5.1. Einleitung

Da es möglich ist, Modelleisenbahnen aus ***Modulen*** zusammenzusetzen, war das Ziel naheliegend, SrrTrains-Anlagen ebenfalls aus Modulen zusammenzusetzen.

Dabei sollte es möglich sein, Module von unterschiedlichen Autoren in ein und derselben Anlage zu verwenden.

Weiters sollte es möglich sein, ***Modelle*** von unterschiedlichen Autoren (also Lokomotiven und Waggons) auf allen SrrTrains-Anlagen gleichermaßen einzusetzen.

Wir hatten also wieder einmal das altbekannte ***Standardisierungsproblem***.

Bei SrrTrains war dieses Problem aber nicht so tragisch, da wir ja bereits auf einen tragfähigen ISO-Standard aufsetzten, eben auf X3D.

5.2. Das MMF-Paradigma

X3D läßt bei der Definition von 3D-Szenen viele Freiheitsgrade.

Bei ***Multiuser-Szenen*** (MU-Szenen) ist es nun so, dass in der Szene verstreut sogenannte ***Netzwerksensoren*** liegen, die es ermöglichen, ***States*** und ***Events*** zwischen den Szeneninstanzen auszutauschen.

Wenn zum Beispiel eine Türe in Kurts Szeneninstanz geschlossen ist, dann soll sie auch in Michaels Szeneninstanz geschlossen sein. Und wenn sich die Türe öffnet, dann soll sie sich in allen Szeneninstanzen öffnen.

Durch die Netzwerksensoren werden die Szeneninstanzen ***synchronisiert*** und alle Mitspieler erleben, sie ***bewohnen dieselbe virtuelle Realität***.

Es ist nun so, dass jeder Netzwerksensor einen eindeutigen ***streamName*** benötigt, um den Netzwerkverkehr der verschiedenen Sensoren auseinanderhalten zu können.

Da die Module und Modelle, die ja die Sensoren enthalten, von verschiedenen Autoren stammen, stellt es ein Problem dar, die streamNames derart zu koordinieren, dass keine Überschneidung stattfindet.

SrrTrains löst dieses Problem, indem definiert wird,

- dass die Szene aus einem ***F***rame, aus ***M***odulen und ***M***odellen besteht (MMF-Paradigma),
- dass der streamName eines Netzwerksensors sich immer aus ***moduleName + objId*** ergibt.

Durch diese Vorgehensweise kann der Frame garantieren, dass alle enthaltenen Module unterschiedliche Namen (moduleNames) haben, und jedes Modul kann garantieren, dass die enthaltenen Modelle unterschiedliche Objekt IDs (objIds) haben.

So ist gewährleistet, dass innerhalb einer Szene alle streamNames eindeutig bleiben.

5.3. Initialization und Attachment

Im letzten Abschnitt war zu lesen, dass eine SrrTrains-Anlage aus Modellen, Modulen und aus einem Frame besteht (MMF-Paradigma).

Wie zu erwarten, bietet das SRR Framework für jeden dieser Anlagenteile eigene X3D-Knoten (Prototypen), dazu folgende Tabelle.

Teil der SrrTrains-Anlage	Zugehöriger Prototyp des SRR Framework
Modelle	MIDAS[1] Objekte (MOBs)
Module	Modulkoordinator (MC)
Frame	Simple Scene Controller (SSC)

Tabelle 1: Anlagenteile und Teile des SRR Framework

Diese Prototypen müssen ***initialisiert*** werden und im Zuge der Initialisierung soll es auch möglich gemacht werden, dass MC und SSC sowie MOB und MC miteinander kommunizieren.

Dazu leitet man eine Referenz auf die ***Common Parameters*** (***commParam***) an den MC weiter und man leitet eine Referenz auf die ***Module Parameter*** (***modParam***) an die MOBs weiter.

Nun können die MCs auf die commParam und damit auf den SSC zugreifen und die MOBs können auf die modParam und damit auf den MC zugreifen.

Eine genaue Beschreibung aller Möglichkeiten im Zusammenhang mit Initialization, Attachment und Disabling findet sich in "Anhang C – Die Modes of Operation (MOOs)"

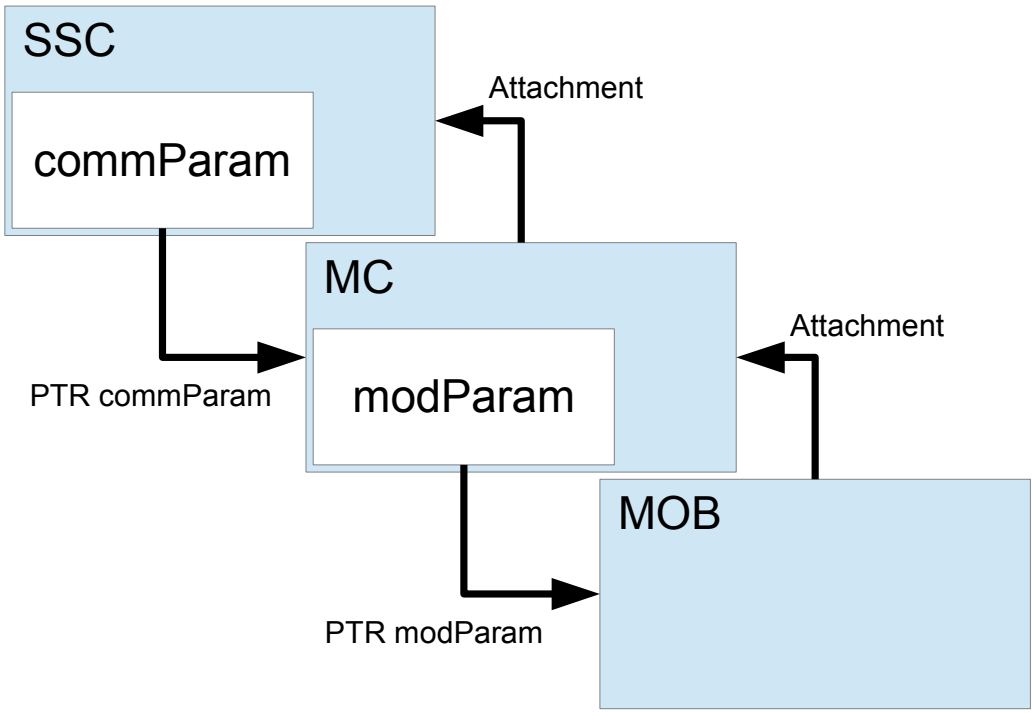


Abbildung 4: Initialization und Attachment

1 MIDAS = Multiuser Interactivity Driven Animation and Simulation

5.4. Die MAM und der MOC

Wir haben bereits in Kapitel "4. Der Simple Scene Controller (SSC)" (???) angedeutet, dass wir mit dem SRR Framework das Ziel verfolgen, sogenannte "Client Based Server Software" (CBSS) zu ermöglichen.

Wir verfolgen also das Ziel, die globalen States der Objekte (sei es nun Türe auf/zu, sei es die Position eines Fahrzeuges oder sei es zum Beispiel der Inhalt eines Schlüsselbretts) nicht am Server zu berechnen, sondern die Software dazu flexibel am Client auszuführen.

Bei Avataren ist das noch relativ simpel, da logischerweise die Szeneninstanz des Users, der durch diesen Avatar repräsentiert wird, die Szeneninstanz ist, die den Avatar kontrolliert.

Aber wie ist das zum Beispiel bei einer Dampflokomotive, die von zwei Usern bedient wird?

Ist die Szeneninstanz des Heizers zuständig für die Kontrolle der Dampflok oder die Szeneninstanz des Lokführers?

Wir könnten ganz einfach sagen: "Die Szeneninstanz, die den 'central controller' beherbergt, soll auch alle anderen Controller (Modul-Controller, Objekt-Controller) beherbergen".

Das wäre jedoch ein sehr unschöner und unsymmetrischer Ansatz.

5.4.1. Der Ansatz des SRR Framework

Das SRR Framework geht den Weg, dass der SSC im globalen "Communication State" unter anderem auch die sogenannte ***Module Activity Matrix (MAM)*** führt.

Ein Modul und alle seine Objekte können in einer Szeneninstanz aktiv sein oder inaktiv.

Das heisst, dass man in Szeneninstanzen, in denen ein Modul und seine Objekte inaktiv sind, rechenzeitintensive Kalkulationen unterläßt, da dieses Modul zur Zeit ohnehin nicht im Interesse des Betrachters liegt.

Es obliegt natürlich dem Modulautor, geeignete Mechanismen einzubauen, um das Aktivieren bzw. Deaktivieren eines Moduls anzustoßen. Im einfachsten Fall könnte dies zum Beispiel ein <ProximitySensor> sein, der prüft, ob sich der User in der Nähe des Moduls befindet.

Unter den Szeneninstanzen, in denen das Modul aktiv ist, wird nun immer genau eine Szeneninstanz definiert, die die Rolle des Modul-Controllers (MOC) spielt.

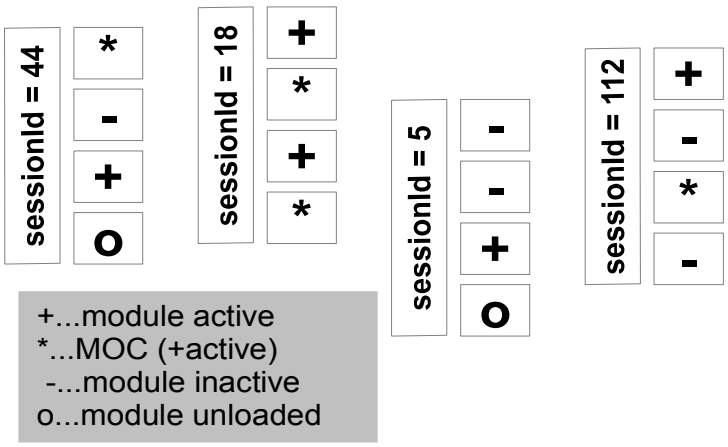


Abbildung 5: Module Activity Matrix (MAM) - 4 Module und 4 Szeneninstanzen

5.5. Die OBCO Rolle

Im Prinzip folgen die ***Objekt-Controller (OBCO)*** aller Objekte (also aller Modelle und aller MIDAS Objekte) den Modul-Controllern (MOC) der Module, in denen diese Objekte attached sind.

Die folgende Abbildung zeigt, wie die OBCO Rolle im global state des Objektes gespeichert und mit übertragen wird. Deswegen ist die OBCO Rolle immer durch die ersten beiden Elemente des global state eindeutig definiert.

Hier hat eine der beiden Szeneninstanzen (SI 1) die OBCO Rolle inne, solange bis es eine "switch over decision" (SOD) gibt, wodurch die OBCO Rolle in SI 1 sofort erlischt und im global state eine Indikation übertragen wird, dass nun SI 2 die OBCO Rolle innehat. Wir sehen, dass Request (A) vom OBCO in SI 1 verarbeitet (P) und vom nicht-OBCO in SI 2 ignoriert (D) wird.

Request (B) geht überhaupt verloren, da es zu diesem Zeitpunkt keinen OBCO gibt und Request (C) wird vom OBCO in SI 2 verarbeitet (P).

Mehr und detailliertere Informationen über die OBCO Rolle findet man im "Anhang D – Die OBCO State Machine".

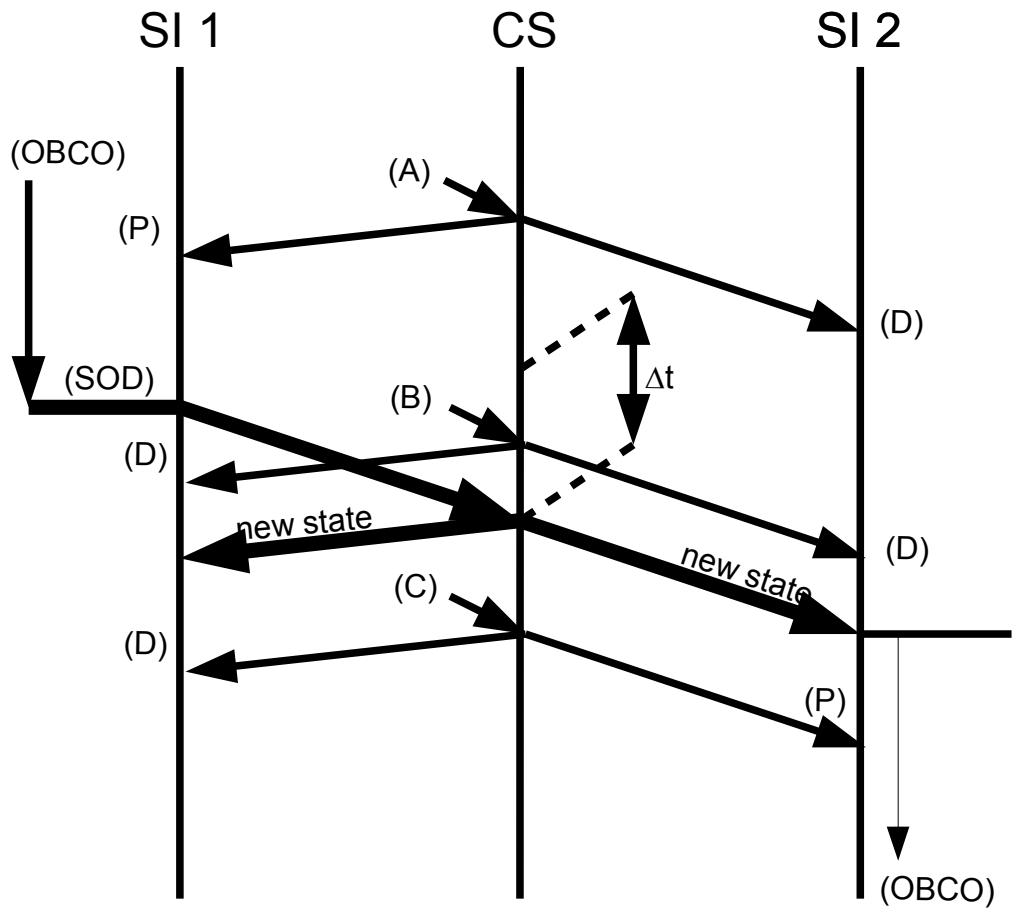


Abbildung 6: Übergabe der OBCO Rolle

Weitere Stichworte

- bereits implementiert

- MIDAS Objekte sind die "invisible engines", sie
    - verstecken die multiuser Problematik
    - rendern keine Graphik und keine Sounds
    - berechnen Parameter, die für die interaktive Animation und Simulation benötigt werden

6. Der Tracer

Da die Teile des SRR/SMUOS Framework einander innerhalb einer Szeneninstanz Events zusenden bzw., da sich Netzwerksensoren über die Grenzen von Szeneninstanzen Events und States zusenden (bzw. diese miteinander abgleichen), ist es zum Zweck des Debugging nötig, alle diese Events und Messages in geordneter Art und Weise auf der Konsole auszugeben, wenn nötig.

Dazu bietet der SMS Tracer (welcher ein eigener Prototyp ist), umfangreiche Möglichkeiten, um diese Events und Messages zu filtern, um einen information overflow zu verhindern.

7. Das Konsoleninterface

Eine SrrTrains Anlage besteht aus Modulen, Modellen und Objekten.

All diese Entitäten kann man über das Konsoleninterface zeilenorientiert ansprechen.

8. Dynamische Modelle

Dynamische Modelle sind Modelle, die man zur Laufzeit der Szene je nach Bedarf nachladen oder auch wieder entladen kann.

Die Probleme, die auftreten, wenn man dynamische Modelle aus verschiedenen Szeneninstanzen miteinader synchronisiert, sollen in einer der nächsten Versionen des SRR Framework von ebendiesem gemeistert werden.

Die Unterstützung dynamischer Modelle ist erst in Entwicklung, wenngleich wir auf der ersten LAN Party bereits mit dynamischen Modellen hantiert haben (siehe nächstes Kapitel).

Die ersten dynamischen Modelle, die es in SrrTrains geben wird, werden Lokomotiven und Waggons sein.

Anbei eine Idee von dynamischen Modellen anhand eines Flugzeugs.

In diesem Beispiel werden dynamische Module (hier als "Tiles" bezeichnet) nachgeladen, zum Beispiel mit Hilfe von Proximity Sensors, und das Flugzeug kann von einem Tile zum nächsten bewegt werden:

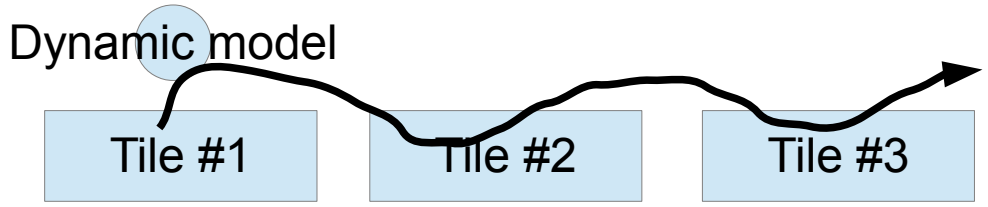


Figure 3: Dynamic model, moving from one dynamic module to another

9. Die erste LAN Party und der "BIMPF Approach"

Die erste, sehr experimentelle, Implementierung des SRR Framework erfolgte im Zeitraum von den Semesterferien 2009 bis zu Ostern 2010, wo dann die sogenannte "1. LAN Party" stattfand (sogenannter "Step 0032").

Einige Freunde trafen sich, um die erste experimentelle Version von SrrTrains zu testen.

Insbesondere die Implementierung der dynamischen Modelle war noch nicht befriedigend, weshalb ich ein Sourceforge Projekt startete und hoffte, es würde sich jemand finden, der hilft.

Dem war nicht so, jedoch war das Thema so faszinierend, dass es noch bis zum heutigen Tag "weiterköchelt".

Der "BIMPF" Approach bedeutet, dass man

- den Netzwerksensor verwendet so, wie er ist
- zusätzliche Funktionen in einer höheren Schicht implementiert
- diese höhere Schicht aus X3D Prototypen mit embedded javascript besteht

10. Dynamische Module

Das "Redesign nach der 1. LAN Party" (auch bekannt als "Step 0033") dauert noch heute an, zur Zeit sind wir im "Hibernation Mode".

Jedoch handelt es sich nicht nur um ein reines Re-Design, sondern es werden auch einige Neuigkeiten im "Step 0033" enthalten sein.

Das sind unter anderem die "dynamischen Module". Das sind Module, die zur Laufzeit der Szeneninstanz nach Bedarf nachgeladen und wieder entladen werden können.

11. Die MIDAS Base (MIB)

MIDAS Objekte wurden bereits in Kapitel 5 vorgestellt.

Die MIDAS Base (MIB) ist ein Teil des SRR/SMUOS Framework, der die grundlegen Funktionen aller MIDAS Objekte enthält, sodass diese nicht mehr in jedem einzelnen MIDAS Objekt implementiert werden müssen.

Ein zentraler Teil in der MIDAS Base wird der Prototyp MibCore (MIB Core), dessen Implementierung jedoch noch nicht abgeschlossen ist.

12. Der "SMUOS/C3P Approach" – Ausblick in die Zukunft

- Dieses Kapitel beschreibt
  die Idee der SMUOS Komponente (Simple Multiuser Online Scenes)
  die Idee des Collaborative 3D Profile (C3P)

12.1. Die SMUOS Komponente

Da der Netzwerksensor ein sehr allgemeines Interface ist – und somit viel Arbeit vom Szenenautor verlangt wird, wenn er den Netzwerksensor verwenden möchte –, ist mein Vorschlag, die grundlegenden Teile des SRR Frameworks, also die X3D Prototypen des sogenannten Base Modules, zu nehmen, um mit ihrer Hilfe neue X3D-Knoten zu definieren, die in der neuen Komponente "SMUOS – Simple Multiuser Online Scenes" zusammengefasst werden sollten.

Dabei könnte man gleich die Knoten von BS Contact, allen voran den <BSCollaborate> Knoten, mit berücksichtigen.

Das macht aber eigentlich nur dann einen Sinn, wenn man auch das Protokoll zwischen Client und Server standardisiert, damit Client und Server nicht vom selben Hersteller kommen müssen. Dazu mehr im nächsten Abschnitt "12.2. Das Collaborative 3D Profile (C3P)".

12.2. Das Collaborative 3D Profile (C3P)

Um das Protokoll zwischen Client und Server zu standardisieren, gab es auf der X3D-public Mailing List den Vorstoß, zu diesem Zweck das XMPP zu verwenden.

Dazu folgende Überlegungen meinerseits:

12.2.1. Auf- und Abbau der Session

Ob zum Auf- und Abbau der Session z.B. das SIP oder das XMPP besser geeignet sind, möchte ich hier nicht erörtern.

12.2.2. States, die sich selten ändern

Für States, die sich selten ändern, deren Änderung aber zuverlässig übertragen werden muss, scheint XMPP über TCP ein geeigneter Ansatz zu sein, oder auch XMPP über SCTP.

Solche States könnten z.B. verwendet werden, um Türen auf- und zuzumachen.

12.2.3. (Quasi-)Kontinuierliche States

Wenn States sich öfters ändern, z.B. die Position eines Avatars oder die Position eines Autos, dann erscheint die Übertragung eines Streams zweckmäßig. Dazu könnte man zum Beispiel ein neues RTP-Profil definieren, nennen wir es C3P analog zu AVP.

12.2.4. Grundlegende Gedankengänge für das C3P (Englisch)

12.2.4.1. Glossary

simple multiuser scene (SMS)
: an SMS enables users to collaborate. Users may exchange information as well as emotion and they may interact with the scene. Users may enrich the scene with information, based on their ideas and creativity or they may be enriched by the scene. Additionally, an SMS may enable users to interact with a subset of the reality

personal scene instance (PSI)
: a PSI is the first point of interactivity for a user. Technically spoken, it is one instance of a Web3D Browser, which interprets a concrete scene graph

server/controller scene instance (SCSI)
: the SCSI is a supporting entity, which enables the SMS to interact with reality. Technically spoken, it is a scene instance that runs client based server software (CBSS).

client based server software (CBSS)
: CBSS is implemented as a part of the SMS (e.g. in the form of X3D "Script" nodes). Each network sensor defines a concrete scene instance where the CBSS for this network sensor is active

user
: a user is a human person, who inhabits a PSI. A human person, who inhabits the reality, but who does not inhabit a PSI, is not a user in the sense of SMS. That person would be seen as "collateral entity"

connectivity platform (CP)
: an entity that connects the scene instances in a star environment

Collaborative 3D Profile (C3P)
: the communication protocol(s) between scene instances and CP

session
: a session is a relation between a scene instance and the CP. This relation is established, when the scene instance is initialized and it ceases to exist, when the scene instance is deleted

multiuser session
: the multiuser session is the current collection of all sessions that share their state via the CP. Sessions will leave and join the multiuser session dynamically

interface to reality (ITR)
: the ITR contains or dynamically creates the model of the reality, which is downloaded to the scene instances and the ITR interfaces the multiuser session with the reality via the POIs

WWW server
: the WWW server is seen as a part of the ITR. It provides the model of the subset of the reality, which can be downloaded via http to the scene instances. The model of the reality might be compiled from different parts in more or less sophisticated manners

Point of Interaction/Point of Interest (POI)
: a POI is an entity that exists in reality and that can influence reality as well as report information about the reality (e.g. a robot, a webcam etc.).
A point of interest delivers read-only data to the multiuser session, where a point of interaction can be read and/or written

12.2.4.2. C3P Systemarchitektur

Personal scene instances are inhabited by users.

Each user can inhabit more than one personal scene instance and he can additionally inhabit the reality.

The scene instances synchronize each other via the connectivity platform.

C3P connects the connectivity platform with the scene instances, in a star environment.

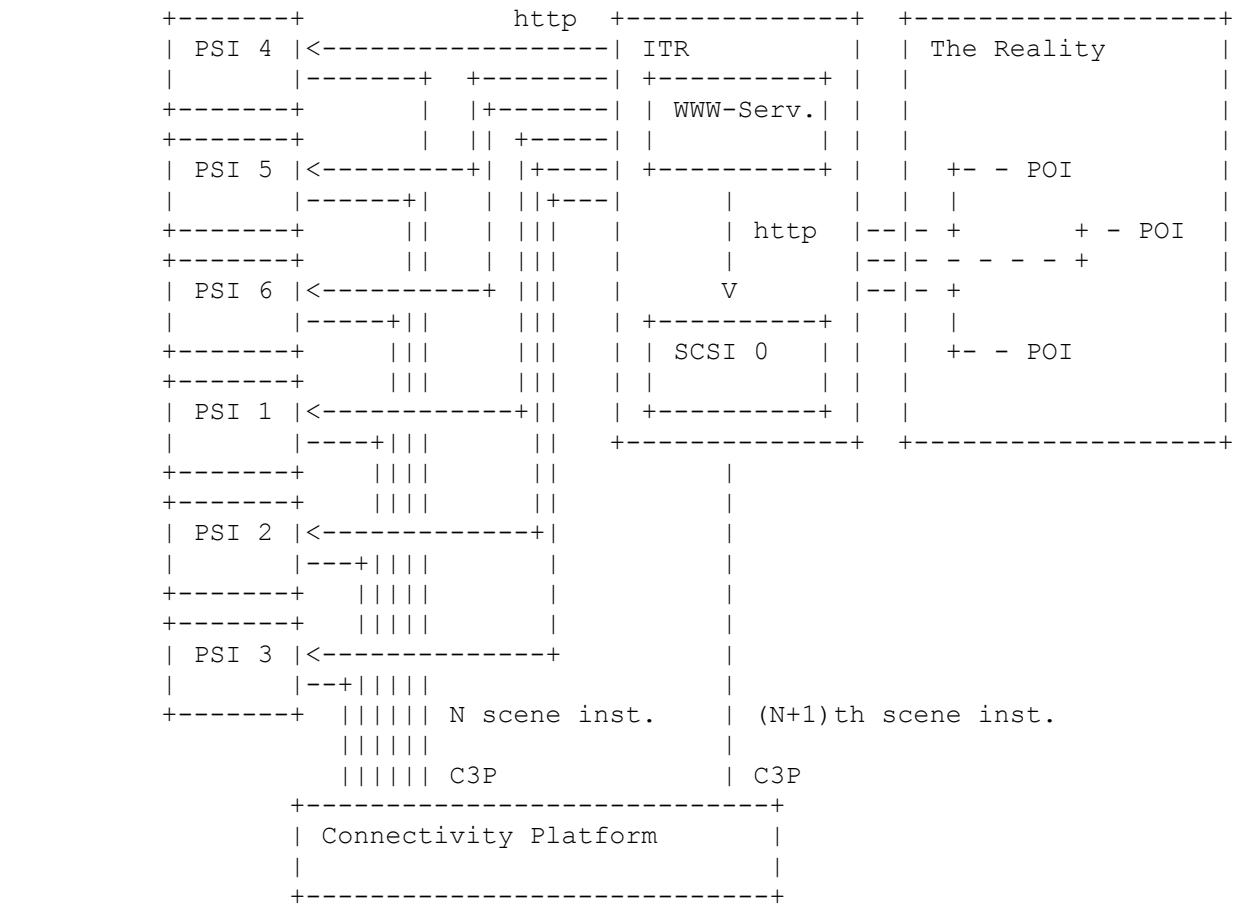


Fig. 1 Example of a Multiuser Scene

Legend:

PSI.....Personal Scene Instance

SCSI.....Server/Controller Scene Instance

ITR.....Interface To Reality

POI.....Point of Interest/Point of Interaction

C3P.....Collaborative 3D Profile

12.2.4.3. Modes of Operation of C3P

12.2.4.3.1. Singleuser VR Mode (w/wo interception of trajectory)

In SVR mode, a multiuser session consists of one and only one personal scene instance, which may have been downloaded from a WWW server.

The scene instance virtually represents an interactive, animated scene, which may be a snapshot representation of a subset of the reality.

If the user is not restricted to a pure virtual environment (e.g. some holo-deck like installation), i.e. if the user inhabits reality, then the scene instance may augment this reality.

12.2.4.3.2. Multiuser VR Mode (w/wo interception of trajectories)

In MVR mode, a multiuser session consists of several or many personal scene instances, allowing the users to virtually collaborate in a virtual scene, which may have been downloaded from a WWW server.

The scene instances virtually represent an interactive, animated scene, which may be a snapshot representation of a subset of the reality.

The users are able to communicate with each other, using e.g. chat, voice chat, avatar gestures etc.

If the users are not restricted to pure virtual environments (e.g. some holo-deck like installations), i.e. if the users inhabit reality, then the scene instances may augment these realities.

12.2.4.3.3. Mixed Reality Mode (w/wo interception of trajectories)

In MR mode, a multiuser session consists of one, several or many personal scene instances and a server/controller scene instance, allowing the users to collaborate in a scene, which has been downloaded from the WWW server.

The personal scene instances virtually represent an interactive, animated scene, which is a representation of a subset of the reality.

The server/controller scene instance connects the multiuser session with the reality via points of interest/points of interaction. It must be clear that the subset of the reality and the representations of the subset of the reality in the scene instances might not be 100% aligned.
This is due to the fact that not everything (every state, every shape, every collateral entity of the subset of the reality, etc.) can be modeled in the representation 100% correctly, nor can it be kept synchronous with the reality in a 100% correct manner.

Hence the reality will often be equalized with a model of the reality. The model of the reality is downloaded via http from the WWW server.

Points of Interaction are entities, that allow bi-directional data exchange between personal scene instances and the reality.

Points of Interest are entities, that allow uni-directional data exchange from the reality to personal scene instances.

The users are able to communicate with each other, using e.g. chat, voice chat, gestures etc.

If the users are not restricted to pure virtual environments (e.g. some holo-deck like installations), i.e. if the users inhabit reality, then the scene instances may augment these realities.

Users, who inhabit reality, may influence the reality directly and hence indirectly impact the representation of the reality within the scene.

12.2.4.4. Operating Modes of POIs and Views of Users

Each user may have a different view to the scene. It's even possible, that a user has more than one view to the scene (this may e.g. happen, when he inhabits more than one personal scene instance).

- 3rd person view – such a user will not have the possibility to influence the scene, but he can be connected to the sensors of a POI.
- pilot view – such a user has the possibility to influence the reality via a POI (he is the pilot of the POI)
- real view – the user inhabits the subset of the reality and can interact with the scene without the help of the CP.

Each POI operates in one of the following operating modes at a given time

- AC/DC Modes
  - Autonomous Mode/Connected

    the POI may be connected to a pilot, but his influence to the POI is limited

    the sensors of the POI can additionally be connected to 3rd person views
  - Disconnected Mode

    the POI is not connected to the ITR
- Connected Mode

  the POI is controlled by a pilot *)

  the sensors of the POI can additionally be connected to 3rd person views

*) the connection between POI and pilot should be designed in a way, so that a possible outage of the ITR does not influence the controllability of the POI.

12.3. Mögliche Schritte, um SMUOS/C3P zu erreichen

12.3.1. Block A "Hobby Usage"

Es wurde das neue Sourceforge Projekt http://smuos.sourceforge.net gestartet.

Das Base Module des SRR Frameworks wurde in dieses Projekt ausgelagert, folgt aber zur Zeit immer noch dem "BIMPF Approach" (der später durch den "SMUOS/C3P Approach" ersetzt werden könnte).

Nunmehr gilt die "Gleichung"

SRR Framework = SMUOS Framework + Train Manager Extension

12.3.2. Block B "Try going to get serious"

Hier könnte man

- Den Netzwerksensor so verbessern, dass er die Controller Roles nativ unterstützt (siehe "Anhang F – Semaphoren und Controller Roles (Englisch)")
- die SMUOS Komponente und das C3P definieren

13. DIGITS, SMUOS and the Theory of Relativity[2]

SMUOS/SMS.....Simple Multiuser (Online) Scenes for the 3D Web

DIGITS.....Distributed Internet Geographic Information Transmission Service

WCS.....World Coordinate System

13.1. Use Cases, The Scope of a SMUOS/SMS

A SMUOS/SMS can be compared with what military people call a "theatre".

From technical perspective, each SMUOS/SMS needs a "world coordinate system" (WCS), which "hangs up" the SMUOS/SMS within the universe.

A WCS can be

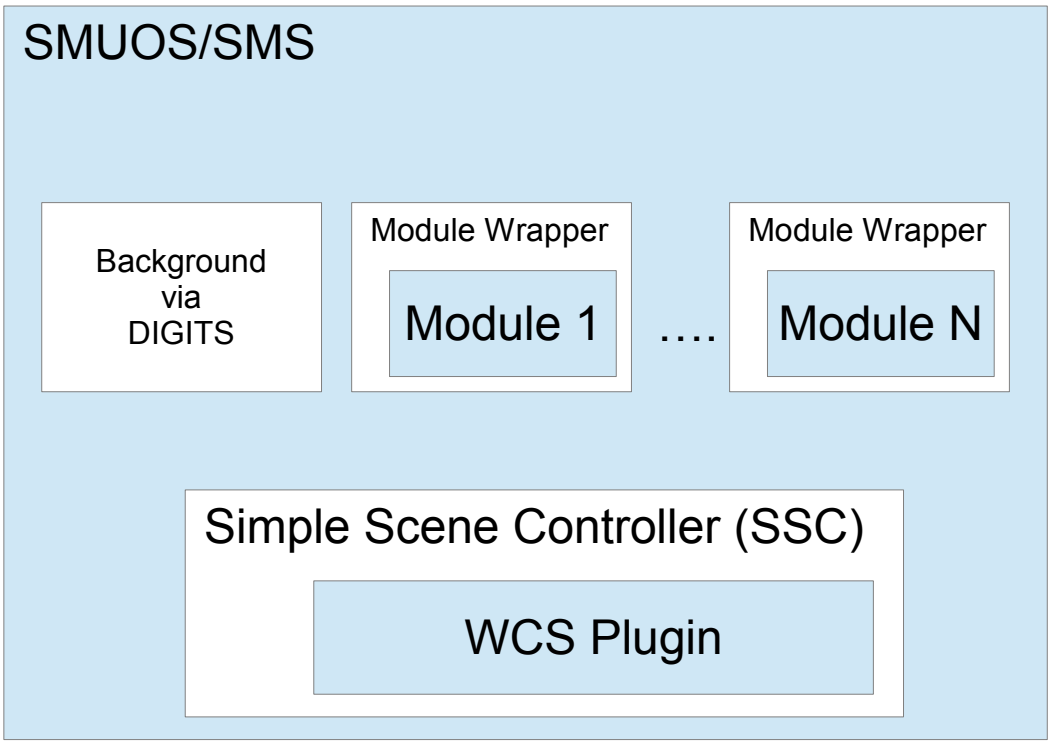
- a cartesian coordinate system or a non-cartesian coordinate system
- an inertial system or an accelerated system
- the WCS can include space warp or it can not include space warp
- and so on

We would like to distinguish following use cases

Type of SMUOS/SMS	Type of WCS	Comments
Classic Cartesian	Classic Cartesian	Gravity in -y direction
X3D-Earth	WGS84	Coriolis forces neglected
Relative SMUOS/SMS	Cartesian with inertial forces	e.g. a scene within one railway car
Relativistic SMUOS/SMS	Riemann Spaces	e.g. starship Enterprise within the subspace

2 Before starting this new generation of SMUOS, I have to seriously study the theory of relativity, which might take a few years

13.2. New Software Structure of a SMUOS/SMS



Use Case	Module Wrapper	WCS Plugin
Classic Cartesian	<Transform>	none
X3D-Earth	<GeoLocation> + <Transform>	none
Relative SMUOS/SMS	New Prototype "ModWrapperRelative" (for inertial forces)	New Prototype "SscWcsRelative" (for inertial forces)
Relativistic SMUOS/SMS	?Maybe need new renderer?	?Maybe need new renderer?

Anhang A – Weiterführende Konzepte

A.1. Handover

Stichworte:

- Zukunftsvision
- Dynamische Modelle können das Modul wechseln

A.2. Moving modules

Stichworte:

- Zukunftsvision
- Module können Teile eines Modells sein
- Teil eines dynamischen Modells -> Handover of Modules
- "Ich baue mir mein kleines Universum"

A.3. Real multibrowser capability

Stichworte:

- Zukunftsvision
- SRR Framework kann unterschiedliche Browser benützen (entweder oder)
- konkurrierende Browser in einer MU Session (gleichzeitig)
- Voraussetzung: Protokoll muss standardisiert sein

A.4. Authoring support

Stichworte:

- Zukunftsvision
- Z.B. Python Scripts for Blender
- aber auch andere Tools denkbar

A.5. Rebase SRR Framework to additional MU Systems

Stichworte:

- Blaxxun
- DeepMatrix
- ...

Anhang B – Ein möglicher Projektplan

SrrTrains v0.01	SMUOS	DIGITS
0033.01 – Initiate Project X		
0033.02 – Module Coordinator		
0033.03 – Dynamic Modules		
0033.04 – Basic SRR Objects		
First Official Pre-Alpha (DONE)		
0033.05 – Modularity and Train Manager Module		
0033.06 – Preps for Rail Vehicles and Trains		
Second Official Pre-Alpha (DONE)		
0033.07 – Web Spaces		
Third Official Pre-Alpha (DONE)		
	A.1: Simplify SIMUL-RR	
0033.08 – Rebase to SMUOS	A.2: Introduce Support for SrrTrains	
Fourth Official Pre-Alpha (DONE)		

Tabelle 2: Pre-Alpha, erster Teil

SrrTrains v0.01	**SMUOS**	**DIGITS**
0033.09 – Rail Vehicles and Trains	A.3: Enhance Support for SrrTrains	
console improvements	MIDAS Object: Trigger	
rename setup points -> replicators	concept for dynamic models and replicators	
MIDAS Object: Replicator	MIDAS Object: Creator	
Fifth Official Pre-Alpha (DONE)		
!!! PROJECT STOPPED HERE = NOW !!!		
Key Container improvements	MIB improvement (MIB Core)	
Inofficial Sixth Pre-Alpha (TO DO)		

Tabelle 3: Pre-Alpha, zweiter Teil

SrrTrains v0.01	SMUOS	DIGITS
creating a dynamic model	model prototypes for dynamic objects	
experimental vehicles (incl. MIDAS Objects)		
basic vehicles (incl. MIDAS Objects)		
SRR Framework improvements		
further models of houses		
Inofficial "LAN Party #3" (TO DO)		
		I. Relativation
		Modules bound via beamer destinations
		Works with Contact Geo and Contact
		Dynamic and Static Modules
		Dynamic/Static Modules and Tiles
		Lorentz-Transformation (SRT)
		II. Tiles vs. DIGITS vs. X3D-Earth
		III. Dynamic Models and DIGITS
0033 – "Very Official" Pre-Alpha		

Tabelle 4: Pre-Alpha, dritter Teil

SrrTrains v0.01	**SMUOS**	**DIGITS**
	A.4: Learn more about Avatars	IV. Avatars and Dynamic Models
	new self-made avatar(s)	DM-Avatars and conventional avatars
	avatar gestures	
	what is specific to BS Collaborate?	
	which editors?	
	graphical avatar selection	
	new background	
	LGPL for Hello World Scenes	
	A.5: Improve GUI	
	A.6: VRML Version	
		V. Handover of DM-Avatars

Tabelle 5: Alpha, erster Teil

SrrTrains v0.01	SMUOS	DIGITS
0034 - Handover and Bumpers		
derailment = explosion		
SRR Objects for bumpers bumper handling		
SRR Objects for HO track HO handling		
SRR Objects for crossing		
SRR Objects for Moving Modules		VI. Moving Modules and DIGITS
models of Y-turnouts, bumpers, crossings Turntables, handover-tracks		
Tracks and Turnouts on dynamic modules		
model of a wagon carrying a wagon		
Alpha		

Tabelle 6: Alpha, zweiter Teil

SrrTrains v0.01	SMUOS	DIGITS
0035 - Coupling and Collisions		
derailment on points, bursting open the points		
Coupling, rear end collision, head-on collision		
slanting collision on turnouts and crossings		
decoupling track		
0036 – Basic Interlocking (1900's)		
Steps to be defined		
Beta		
00xx - More Topics		VII. DIGITS
Real Multibrowser Capability		
Authoring Support (Blender Python Scripts)		
X3D-Earth		
0100 "Release" SrrTrains v0.01	**SMUOS "Block A"**	**DIGITS "Initial"**

Tabelle 7: Beta und Release SrrTrains v0.01 / DIGITS

Anhang C – Die Modes of Operation (MOOs)

C.1. Die MOOs des SSC

Bevor irgendein Modul oder Modell einer Simplen Multiuser Szene (SMS) initialisiert werden kann, muss man den Simple Scene Controller (SSC) initialisieren.

Dieser befindet sich im Frame der SMS und kann somit niemals entladen werden, ausser die gesamte Szeneninstanz wird beendet und entladen.

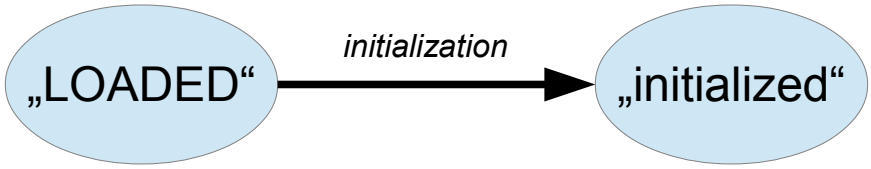


Abbildung 7: Die MOOs und MOO Changes des Simple Scene Controller

C.2. Die MOOs des MC

Wenn der SSC initialisiert worden ist, liefert er eine Referenz auf die "Common Parameters" (commParam).

Diese Referenz kann an ein Modul, insbesondere an den Modulkoordinator, weitergereicht werden, damit dieses Modul initialisiert wird und sich an den SSC attached.

Diesen Zustand nennt man "attached".

Im Falle eines dynamischen Moduls kann dieses auch wieder mit Hilfe des Feldes "disable" in den Zustand "disabled" versetzt werden. Dadurch wird das Modul vom SSC detached und alle Modelle auf dem Modul werden ebenfalls disabled (siehe Kapitel "C.3. Die MOOs eines Objektes").

Dadurch verhält sich das gesamte Modul im Zustand "disabled" komplett passiv.

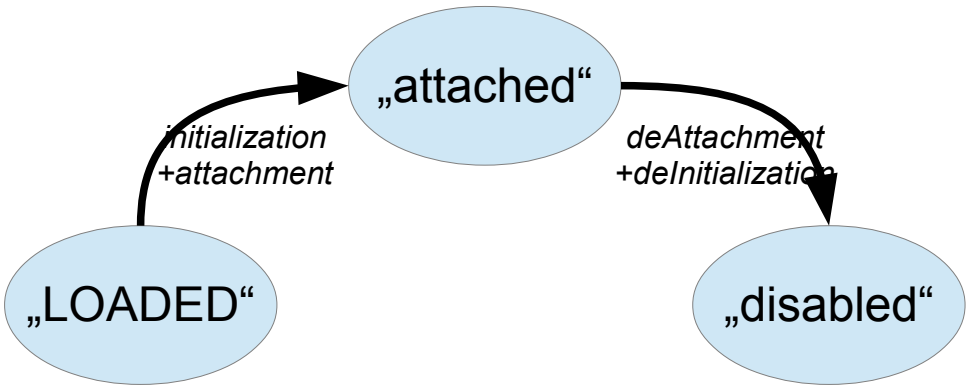


Abbildung 8: Die MOOs und MOO Changes des Module Coordinator

C.3. Die MOOs eines Objektes

In Kapitel "5.3. Initialization und Attachment" wurde bereits angedeutet, dass ein Objekt (also ein MOB oder ein Modell) an ein Modul "attached" werden kann, indem man ihm die "Modul Parameter" übergibt.

Die Regel ist eigentlich sehr einfach:

1. Wenn man einem Objekt die "Common Parameters" commParam übergibt, dann "kommt es ausserhalb aller Module zu liegen" (es ist dann "detached"). Diesen Zustand bezeichnet man als "initialized".
2. Wenn man einem Objekt die "Module Parameters" eines Moduls (modParam) übergibt, dann "kommt es innerhalb dieses Moduls zu liegen". Diesen Zustand bezeichnet man als "attached".
3. Wenn man das Feld "disable" eines Objektes triggert, dann verhält sich dieses Objekt ab sofort absolut passiv. Diesen Zustand bezeichnet man als "disabled". Das ist nötig, weil Web3D Browser Content nicht unbedingt aus dem Speicher entfernen, wenn man ihn aus einer Szene entfernt hat. Deshalb muss man garantieren, dass der Content "nicht mehr stört" und "sich passiv verhält".
4. Dynamische Objekte können – auch mehrmals – zwischen den Zuständen "attached" und "initialized" ("detached") wechseln, statische Objekte können dies nicht.

Diese Regeln sind in folgenden Abbildungen zusammengefasst.

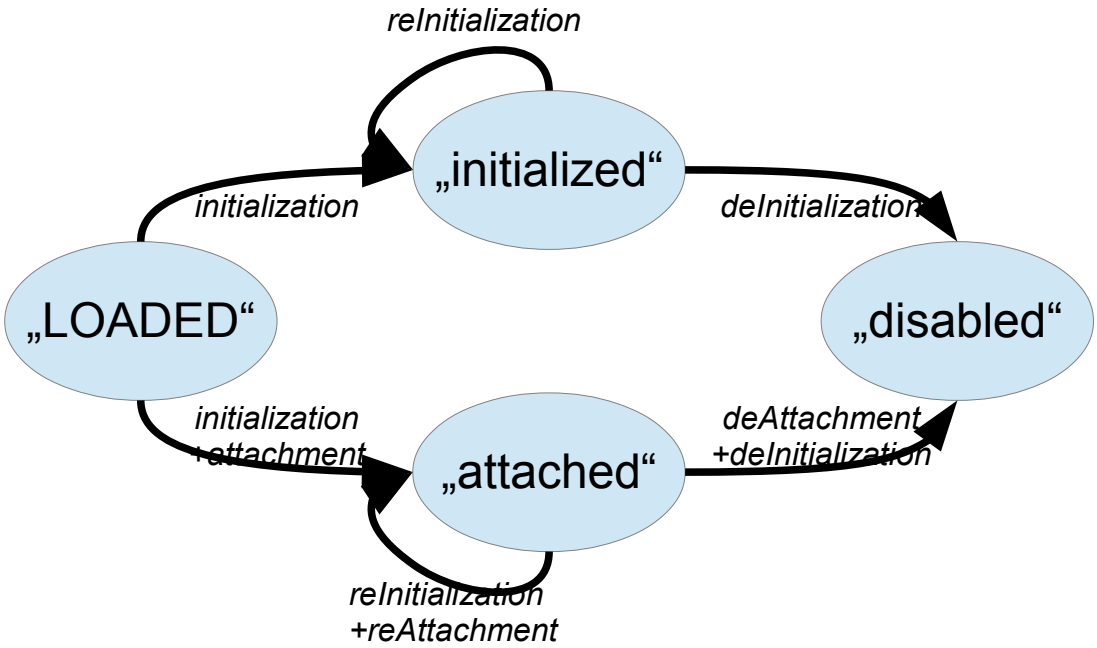


Abbildung 9: Die MOOs und MOO Changes eines statischen Objektes

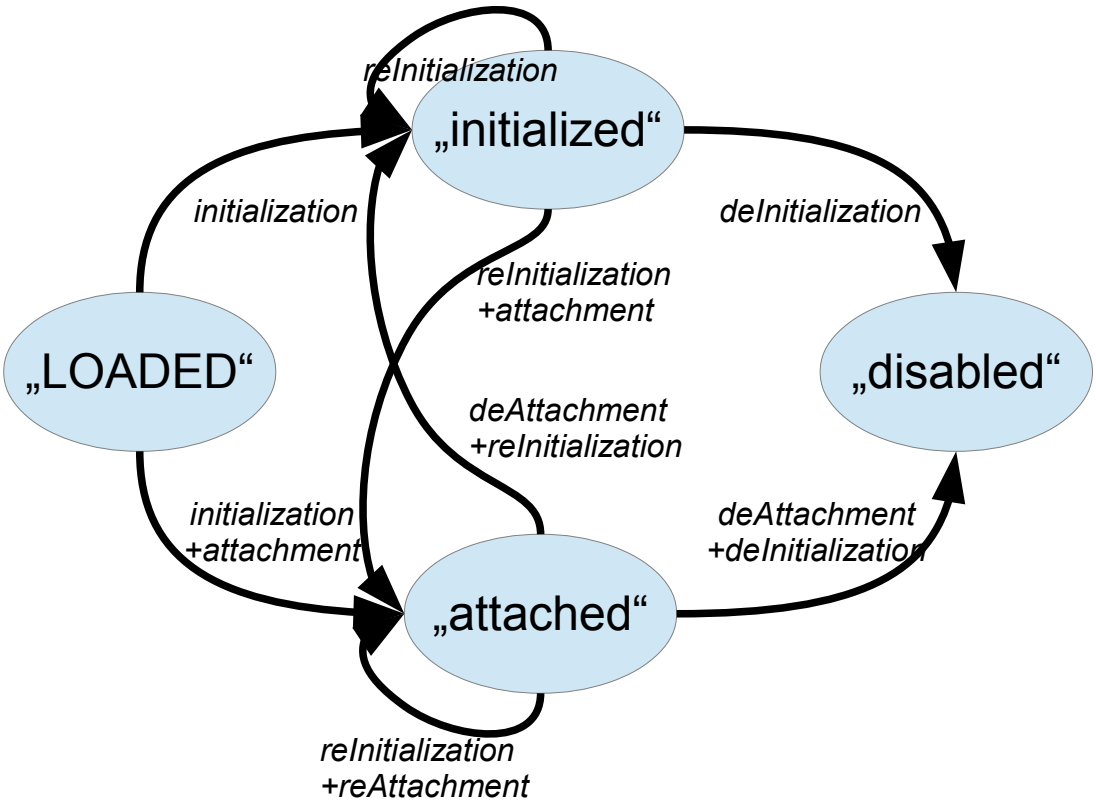


Abbildung 10: Die MOOs und MOO Changes eines dynamischen Objektes

Anhang D – Die OBCO State Machine

Die OBCO State Machine ist ein Teil der MIDAS Base und wird vom SRR Framework zur Verfügung gestellt.

Der User, also der Programmierer, der das MIDAS Objekt entwickelt, muss nur die beiden Flags "iAmActive" und "iAmObCo" beachten, die der Output der OBCO State Machine sind und am externen Interface der MIDAS Base zur Verfügung stehen.

Andererseits ist die OBCO State Machine vom MOO des Objektes abhängig (siehe dazu auch das Kapitel "C.3. Die MOOs eines Objektes") und sie verarbeitet die Inputs von der MAM.

Die folgende Abbildung gibt grob wieder, in welchen Zuständen sich die OBCO State Machine befinden kann. Dabei gelten folgende Regeln:

- Ein Objekt kann höchstens einen OBCO haben
- Es kann auch längere Zeiträume geben, in denen ein Objekt überhaupt keinen OBCO hat (zum Beispiel, wenn das parent module in keiner Szeneninstanz aktiv ist und somit auch keinen MOC hat)

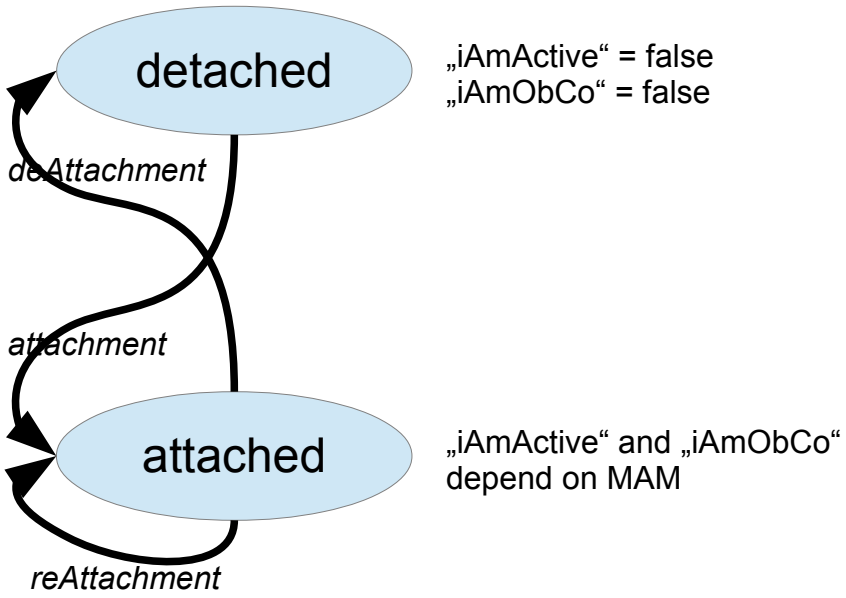


Abbildung 11: Mögliche Zustände der OBCO State Machine

Welche Szeneninstanz die OBCO Rolle innehat (das Flag "iAmObCo" kann nur in einer Szeneninstanz den Wert true haben), ist in den ersten beiden Elementen des global state dargestellt (der global state ist ein MFString Wert).

- OBCO ist definiert: state[0] = <moduleName>, state[1] = <sessionId>
- OBCO existiert nicht: state[0] = '', state[1] = '-2'
- OBCO existiert noch nicht, aber eine Szeneninstanz hat bereits angekündigt, dass sie nach einiger Zeit "die OBCO Rolle an sich reissen wird": state[0] = <sessionId>, state[1] = '-2'

Anhang E – Analogien zwischen 3D Szene und Realität (Engl.)

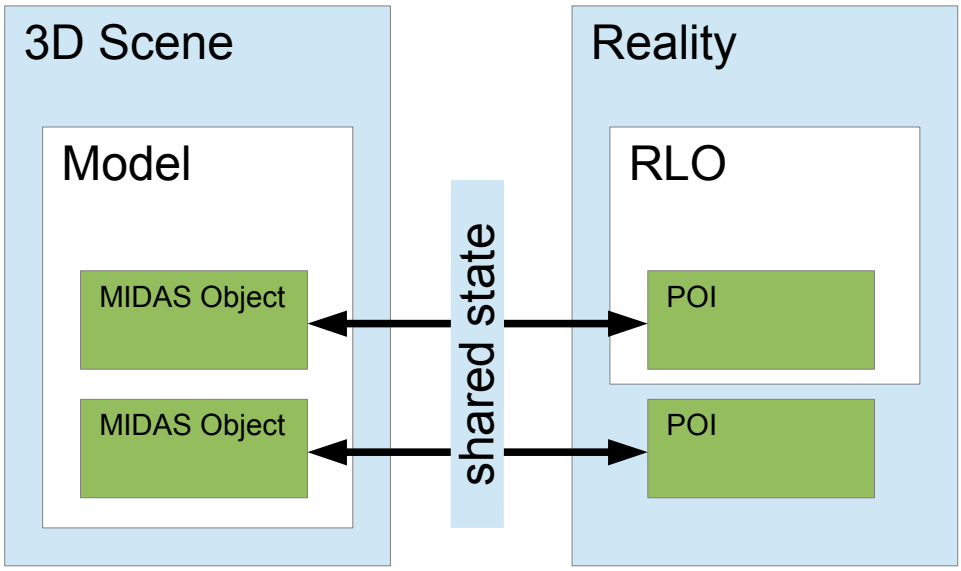


Figure 4: Analogies between 3D scene and reality

The SMUOS concept will try to define following analogies:

SMUOS Term	Equals in Reality	Example
Simple Multiuser (Online) Scene SMUOS / SMS	subset of the reality	a car race
Model	real-life object (RLO)	a car
MIDAS Object	point of interaction (POI)	steering of a car

Table 8: Analogies between 3D scene and reality

The SMUOS concept will define the usage of identifiers for 3D objects, i.e.for models and for MIDAS Objects.

However, it's up to the ITR to map those identifiers to real-life identifiers for RLOs and POIs. We will try hard to not define such mapping, nor to define real-life identifiers.

Mapping between MIDAS Objects and POIs need not be a 1:1 mapping, e.g. several MIDAS Objects defining the steering of a car could be mapped to one single POI (transport address) for the real-life steering.

However, we think mapping between models and RLOs should be kept a 1:1 mapping, whereever possible, at least as long as we talk about one single SMUOS.

This should not preclude a parallel mapping of one and the same RLO to different models in more than one SMUOS.

Anhang F – Semaphoren und Controller Roles (Englisch)

F.1. References

[1] SMUOS/C3P *: The Idea
http://smuos.wordpress.com/2011/03/01/smuos-and-the-ietf/ (meanwhile deleted)

[2] SMUOS *: The Project
http://smuos.sourceforge.net

[3] Description of BS Collaborate (an example network sensor implementation)
http://bitmanagement.de/download/BS_Collaborate/BS_Collaborate_documentation.pdf

[4] The SrrTrains Project
http://simulrr.wordpress.com/berichte (meanwhile deleted)

* SMUOS = Simple Multiuser Online Scenes

C3P = Collaborative 3D Profile

F.2. Summary

This hobby report deals with a possible enhancement of the X3D Network Sensor node.

We rely on the experiences with the example implementation BS Collaborate [3].

Those experiences were made during conduction of the SrrTrains project [4].

F.3. Motivation

The SMUOS Framework [2] uses network sensors for the **following use cases**

(a) in the Simple Scene Controller Base (SSC Base)

(b) in Simple Scene Controller Extensions (SSC Extensions)

(c) in the SSC Dispatchers

(d) in MIDAS Objects

In all cases **we rely on what we call "Client Based Server Software" (CBSS)**.

That means, we do not use prefixes like "add_", "inc_" or "dec_" to perform server side calculations, but we use the prefix "set_" to apply the result of client side calculations to the states, which are stored persistently on the collaboration server (CS) afterwards.

Client side calculations have the **advantage** they can be easily and flexibly programmed, e.g. using client side JavaScript.

On the other hand, those calculations have the **disadvantage** a clear distinction must be made to define, which scene instance is responsible to perform the calculations and to update the states. Otherwise (if two or more scene instances felt "responsible" to update the states at the same time) an unpredictable behavior would be the result.

For this reason, **we defined "controller roles"**.

Each network sensor is "controlled" by one distinct scene instance. This scene instance, and only this scene instance, is allowed to perform client side calculations for this network sensor and to

update states using the "set_" prefix.

Using the present mechanisms of the network sensor (events, states, ...), we described the BIMPF approach to achieve our goals in an experimental way. This is examplified in the SrrTrains project [4] and in the SMUOS project [2].

We think it would be of advantage not only for us, but for a broader variety of applications, if the network sensor would support the controller roles in a native way (this is similar to the fact that semaphores should be supported by the operating system).

F.4. Proposal

Thus we do following proposal:

We do this proposal without having tried it.

Some experiments with real software should be done, before incorporating the proposal to the X3D standard.

1. Enhance the Network Sensor by five additional fields

outputOnly	SFBool	controller	
inputOutput	SFBool	requestController	default: TRUE
outputOnly	SFBool	controllerRequest	
inputOnly	SFBool	controllerGrant	
initializeOnly	SFString	followStreamName	default: ""

2. Define a new prefix/suffix for network sensor events, which provides for routing of events to the scene instance owning the controller role
   "ff_", "_ff"

ad 1.

The field "controller" indicates, whether this scene instance holds the controller role for this network sensor. The network sensor together with the CS guarantee that only one scene instance receives "controller"=true at one time for one network sensor.

Sending an SFBool event to the field "requestController" a scene instance can request the controller role for this network sensor.

The setting of "requestController" indicates during initialization of the network sensor, whether the network sensor shall queue up at the CS for getting the controller role.

The fields "controllerRequest" and "controllerGrant" enable the CBSS to reject or to allow a controller request of another scene instance.

Setting the field "followStreamName" to a value not equal the empty string will cause the controller role of the network sensor to "follow" the controller role of another network sensor, i.e. the sensor that is identified by its "streamName" equal to the "followStreamName".

Hence an input to the "requestController" field will not have any effect, if the "followStreamName" field matches the "streamName" of another existing network sensor.

If not any scene instance feeds an event to the "requestController" field and if the "followStreamName" field is empty and if the "requestController" field is initialized to a value of "TRUE" in all scene instances, then the CS and the network sensor will care automatically that one and only one scene instance owns the controller role for this network sensor at any given time in the lifetime of the multiuser session.

ad 2.

Events at fields with the "ff_" prefix will not be distributed evenly to all scene instances, but they will be routed to the scene instance that owns the controller role for this network sensor.

The abbreviation "ff" is inspired by theory of linear differential equations, where the "forcing function" takes a similar role as the "ff" prefix/suffix does for the proposed network sensor enhancement.

F.5. Examples

Please find following figures just for demonstration of the concepts of the proposal.

The names of the messages between network sensors and collaboration server (CS) are exemplary and need not be identical to the actually used message identifiers.

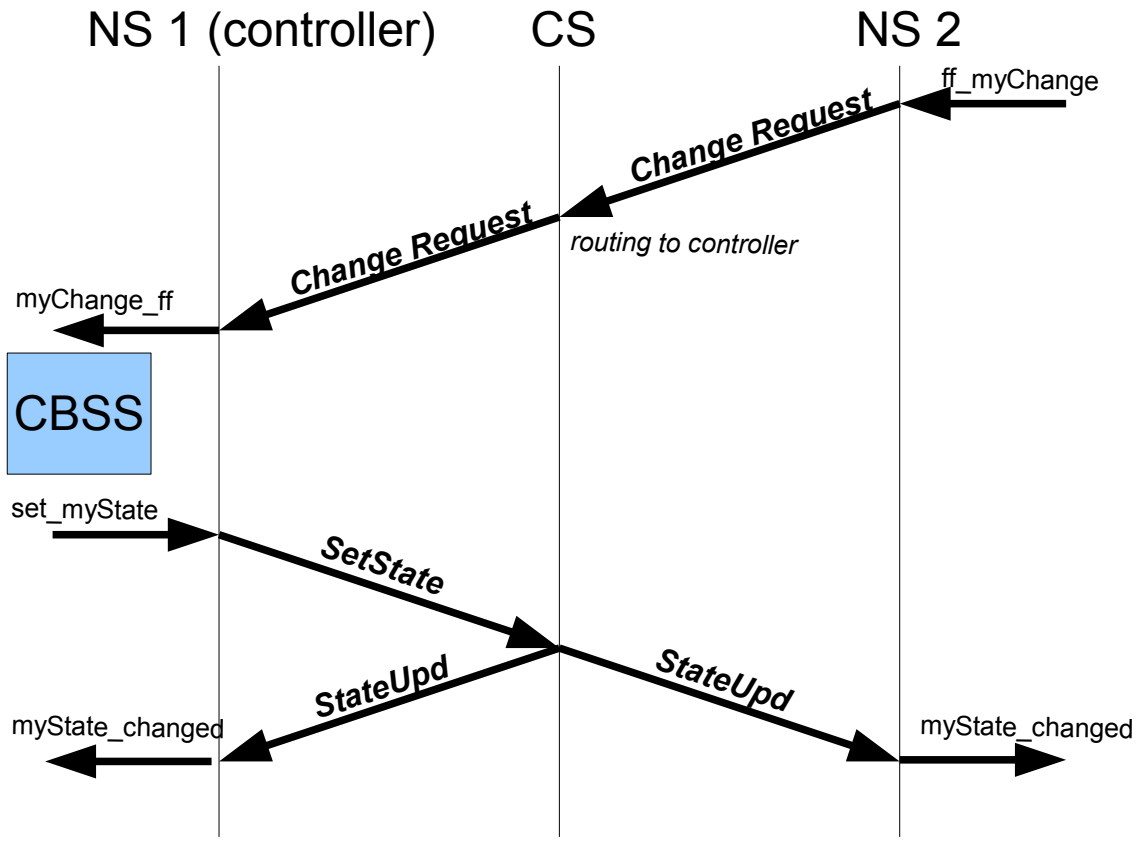


Figure 5: Principle of CBSS with enhanced network sensor

Figure 5 demostrates the basic idea of Client Based Server Software (CBSS). If we talk about one network sensor within the scene, then we have N instances of this network sensor, where N is the number of users logged in, i.e. the number of scene instances.

The "Change Request" (let's call it so) is similar to any other network sensor event in that it is not stored at the CS. On the other hand, it is not distributed to all scene instances, but it is routed to only one scene instance. This is the scene instance that owns the controller role for this network sensor.

Since this routing is done by the CS, we see the CS must keep track of all controller roles.

The CBSS is active in one and only once scene instance (but it is present as e.g. JavaScript in all

scene instances).

The CBSS calculates the new value of the state and sends the new value to the CS, who stores it persistently and distributes it to all scene instances, as usual.

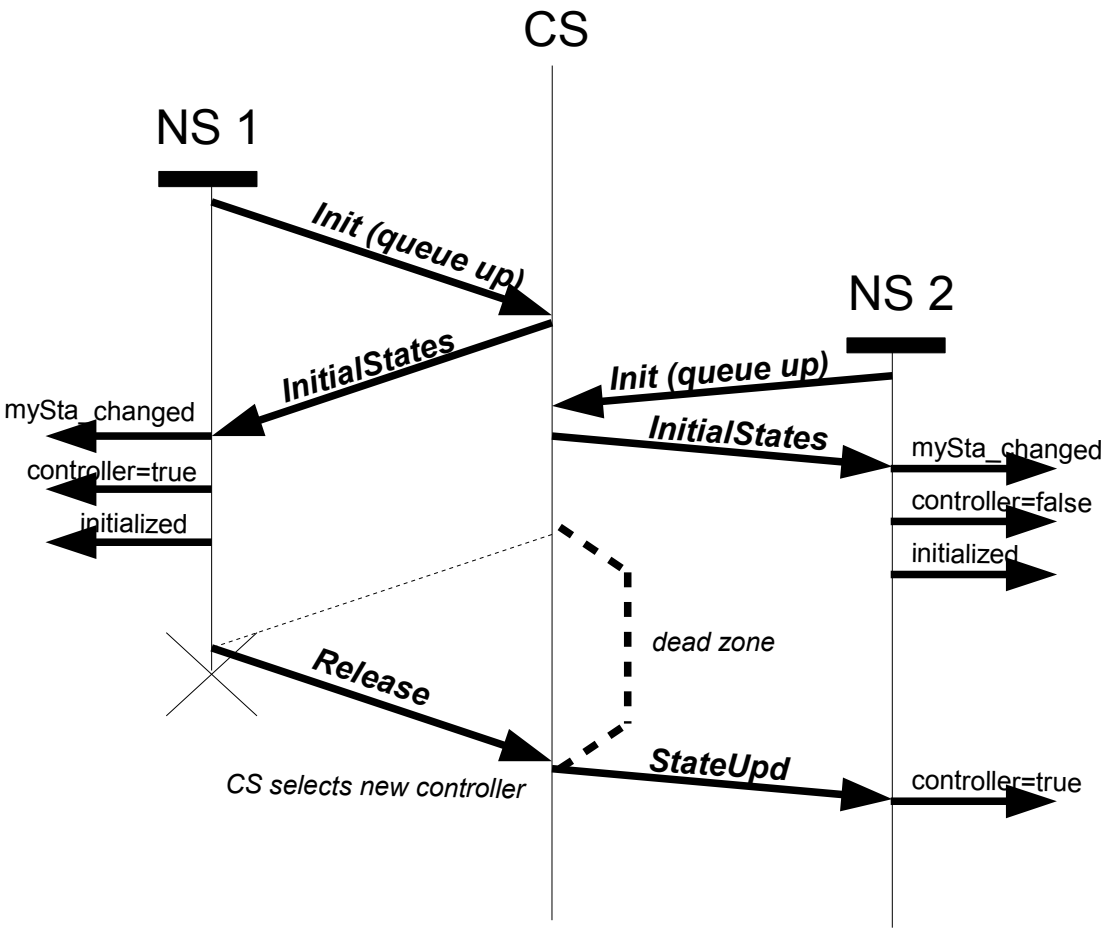


Figure 6: Change of Controller Role without event at "requestController"

Figure 6 Demonstrates, how the CS defines the controller role automatically, if not any scene instance sends the "requestController" event.

- The first scene instance that initializes the network sensor, gets the controller role (if all scene instances initialize "requestController" to "TRUE", otherwise none will get the controller role)
- If a scene instance having the controller role does leave the multiuser session, then the CS defines a new scene instance having the controller role (all scene instances queued up for the controller role, because they have initialized "requestController" to "TRUE")

We see a "dead zone". If a Change Request ("ff_") is received at the CS during this time interval, then the Change Request will not be processed by a controller. The Change Request will get lost.

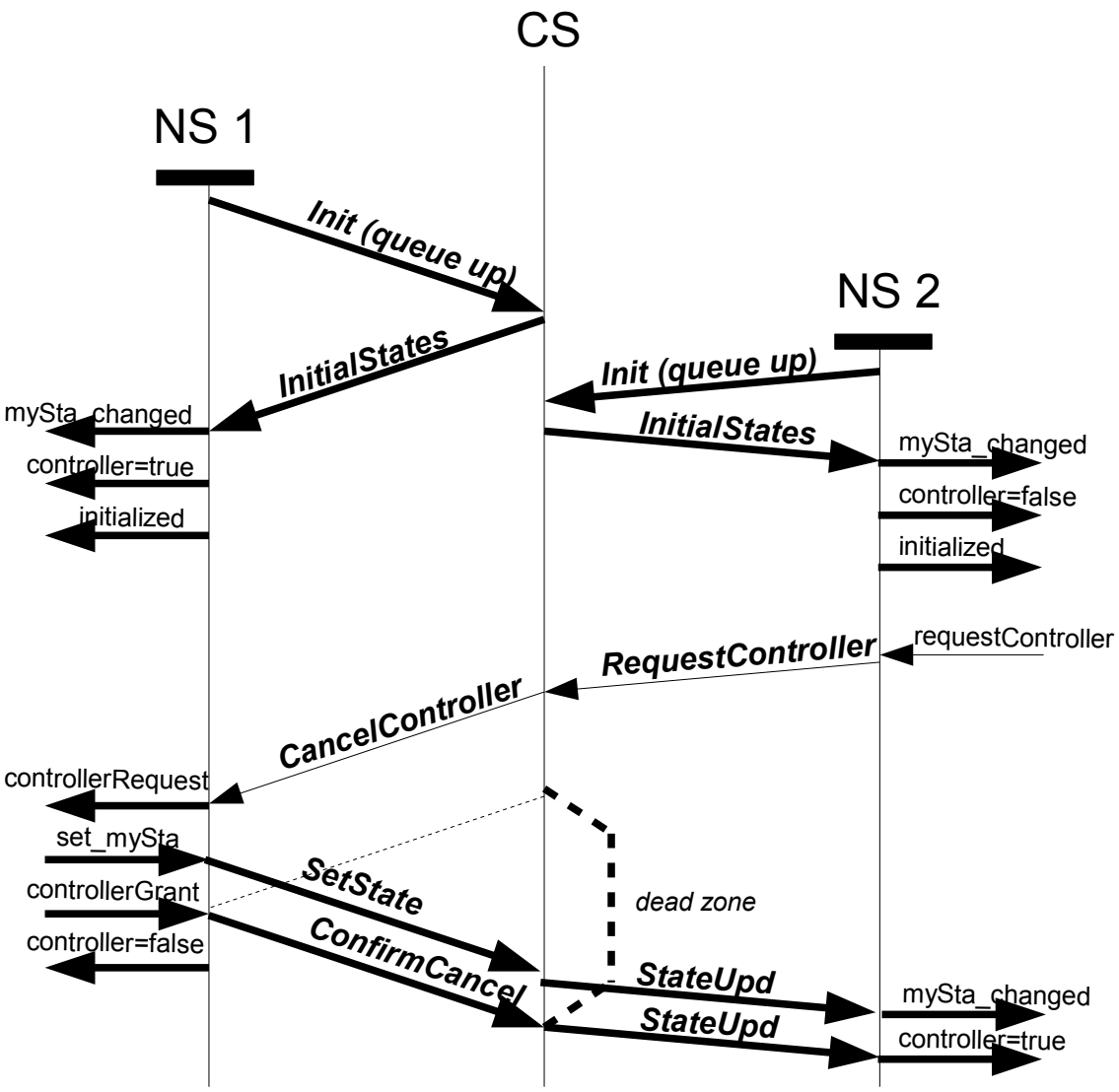


Figure 7: Change of Controller Role with "requestController"

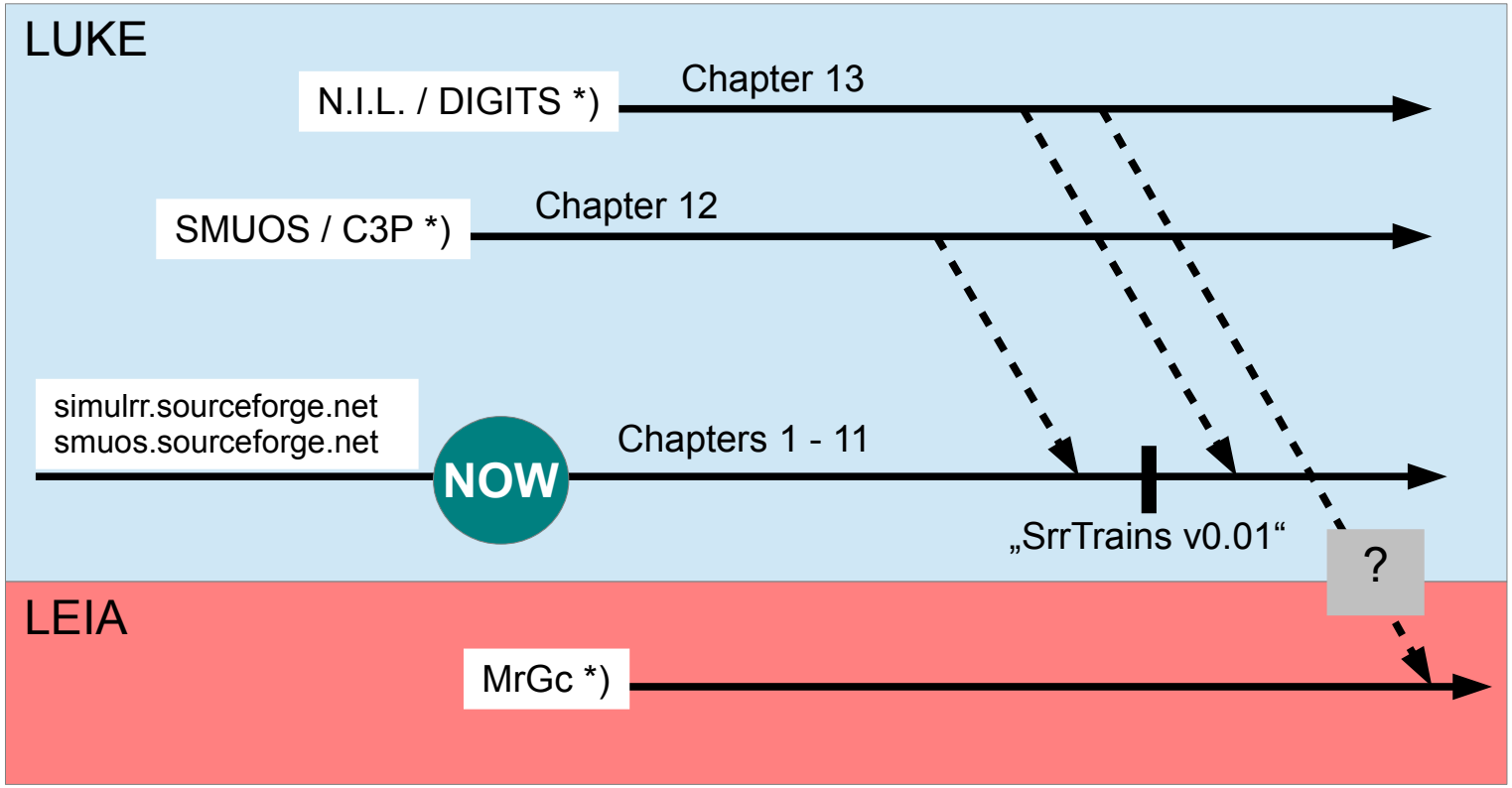
Figure 7 examplifies, how a scene instance (here NS2) can request the controller role and the CBSS (here NS1) must confirm the request. The CBSS **may**, but **needs not** send a last state update, before it confirms the controller request.

A similar function was introduced to the SRR Framework and to the SMUOS Framework, to be able to run the tracer on a selected scene instance and to be sure the CBSS will be traced by the tracer.

However, in conjunction with the SMUOS/C3P idea [1] this might be useful in another sense, too.

We see a "dead zone" again. If a Change Request ("ff_") is received at the CS during this time interval, then the Change Request will not be processed by a controller. The Change Request will get lost.

Anhang G – Trying a Prophecy about SrrTrains v0.01 – Principle Timeline



*) These names are placeholders for „Geographic Infrastructure“, „MU System“, „application-early-in-one-universal“

Figure 8: Ideas of a rough timeline