MOSIM Usage and Installation

Janis Sprenger, Klaus Fischer, André Antakli

Nov. 2021

End-to-end Digital Integration based on Modular Simulation of Natural Human Motions

An open modular framework for efficient and interactive simulation and analysis of realistic human motions for professional applications

Mission Statement:

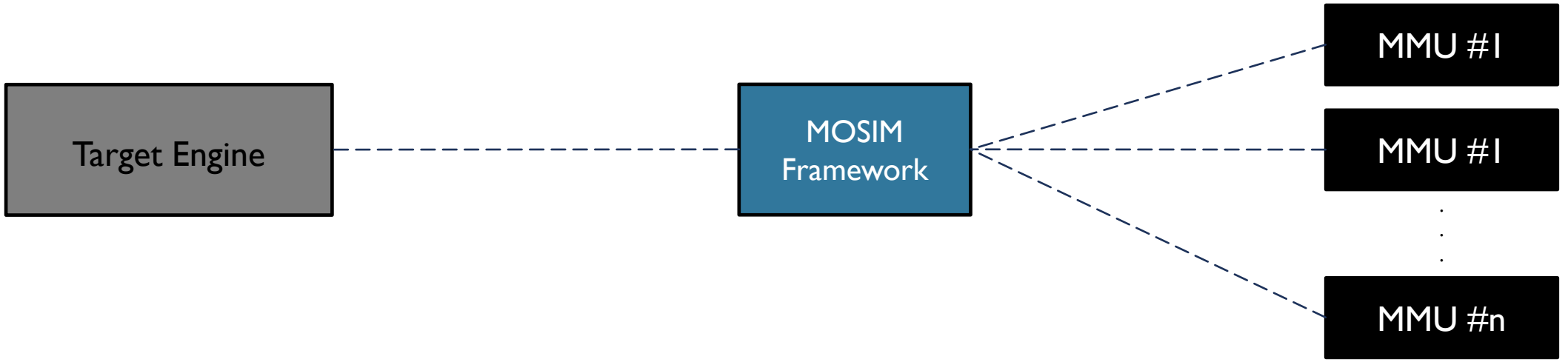
Agenda

- What is the MOSIM Framework?
- Where do I find the MOSIM Framework?
- How can I run the MOSIM Framework?
- How can I integrate MOSIM into Unity?

What is the MOSIM Framework?

Goal Description: Integration of different Motion Synthesis Approaches

- **MMU** (Modular Motion Unit): (black-box) motion synthesis approach
- **Target Engine**: (game-) engine, in which the visualization is displayed
- **Goal**: Combine multiple sequential and parallel MMUs to display a complex motion sequence in the target engine.



Cross-Language Compatibility

- MMUs: can utilize any language (C, C++, C#, Python, Java, Go, …)
- Communication over TCP/IP protocol
- Middleware: Thrift
- The MMIStandard is implemented in thrift
  - Defining standard data types (e.g. vectors, poses, constraints, …)
  - Defining service interfaces (e.g. for an MMU)
- Every software component implementing the thrift interfaces can be integrated in the MOSIM framework

What is Thrift?

- Defines data types and function interfaces in a programming language independent format
- Auto-Generates code for the data types and function interfaces for (almost) all programming languages
- Enables remote-function calls over different languages
- Example: left and right software snippets are supposed to run on different machines / in different programming languages

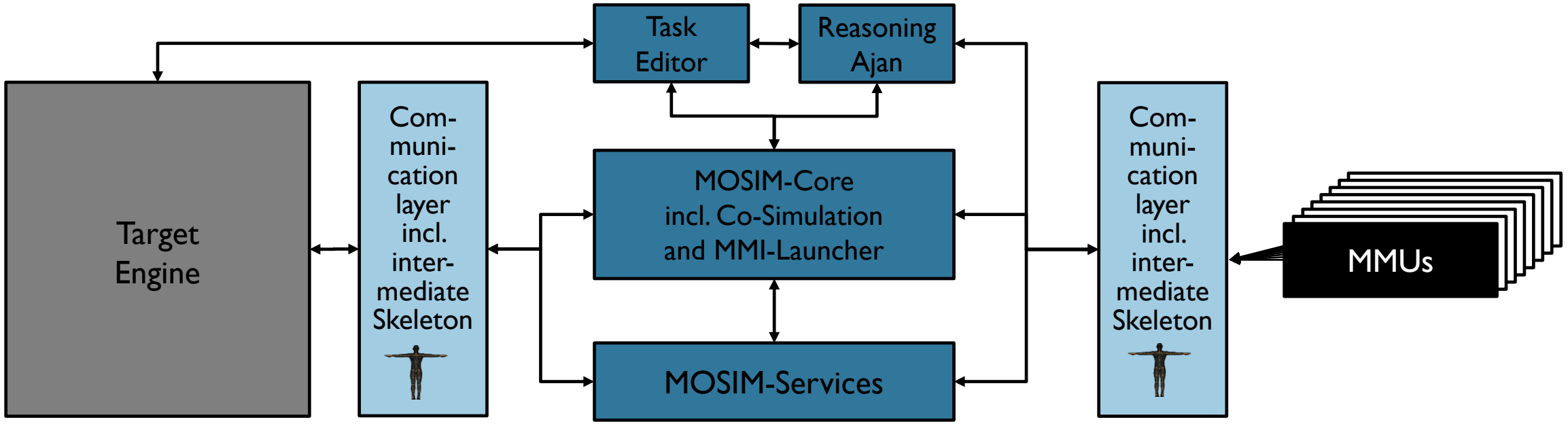
Example in MMU Side (e.g. Python)

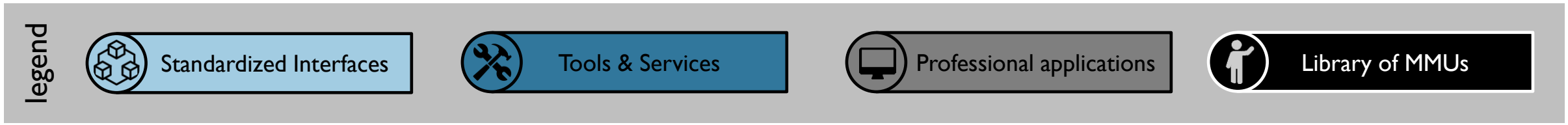
```
class MyMMU(MotionModelUnit):
        def DoStep():
            …
myMMUInstance = MyMMU (…)
myMMUInstance.DoStep()
```

Example in remote side (e.g. C#)

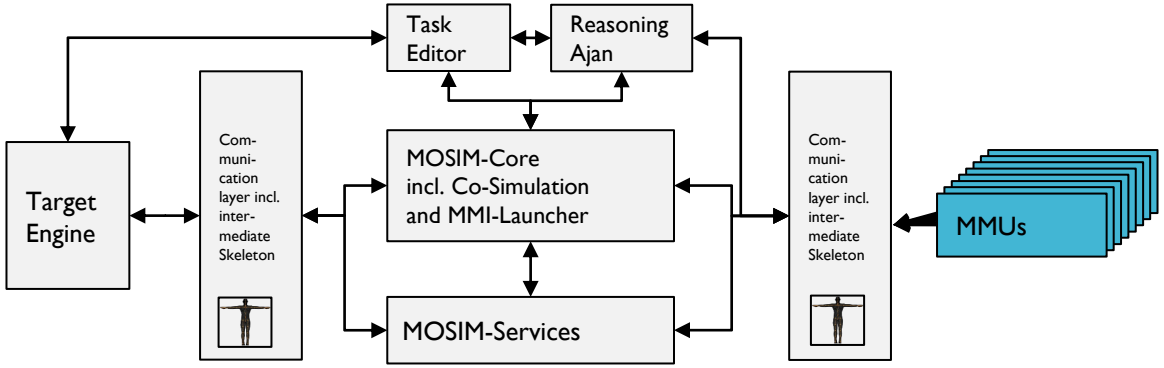
```
MotionModelUnit myMMUInstance = MotionModelUnit.Client(…);
myMMUInstance.DoStep();
```

Updated architecture based on demonstrators
Complete MOSIM Framework

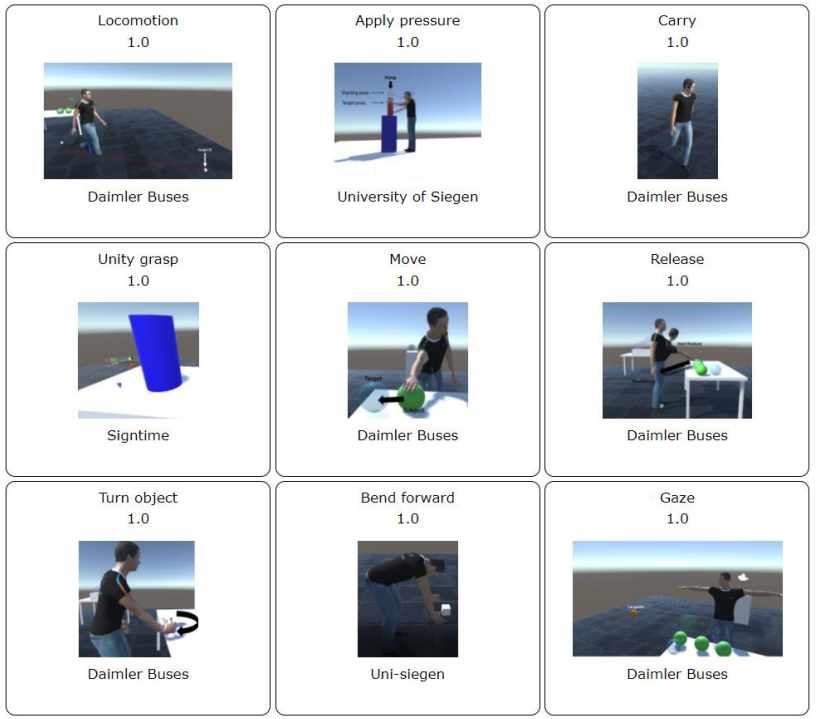




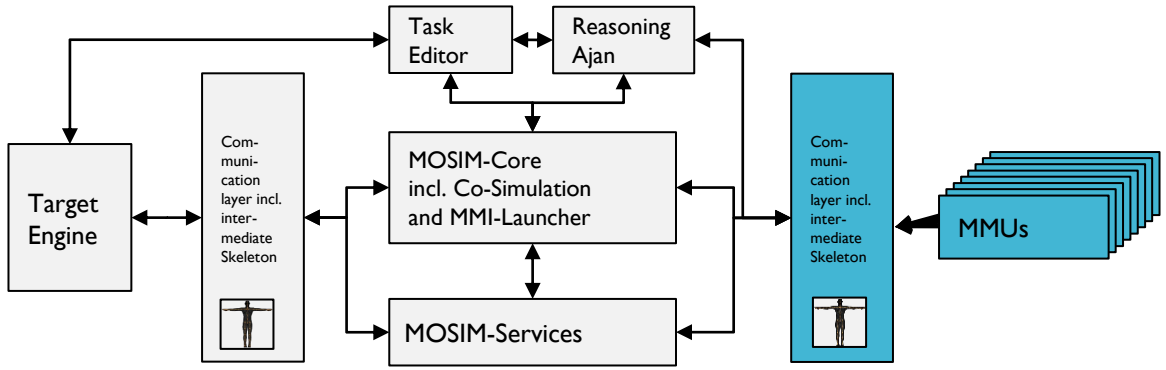
Complete MOSIM Framework: MMUs



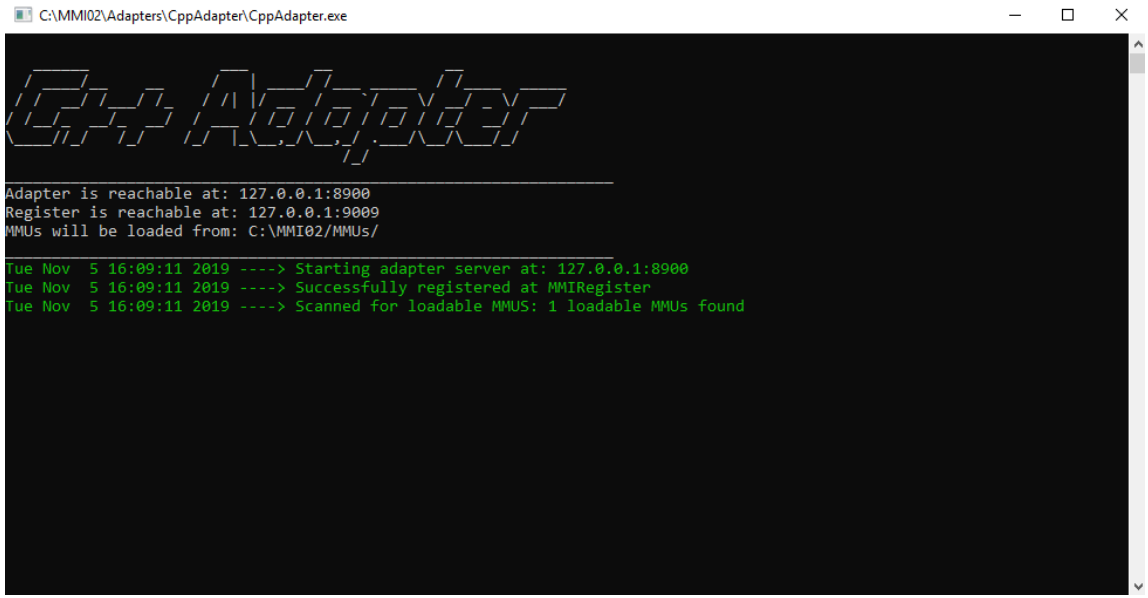
- Modular Motion Units
- Black box execution models
- Communication via thrift



Complete MOSIM Framework: Adapters

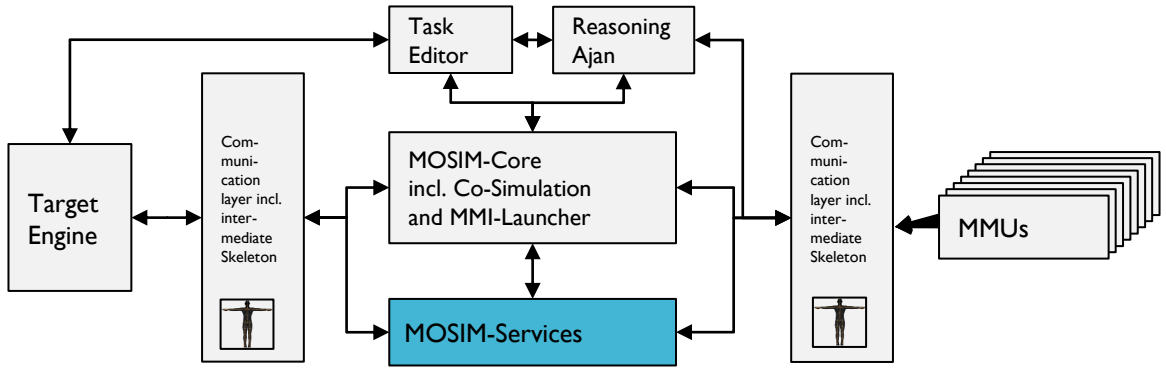


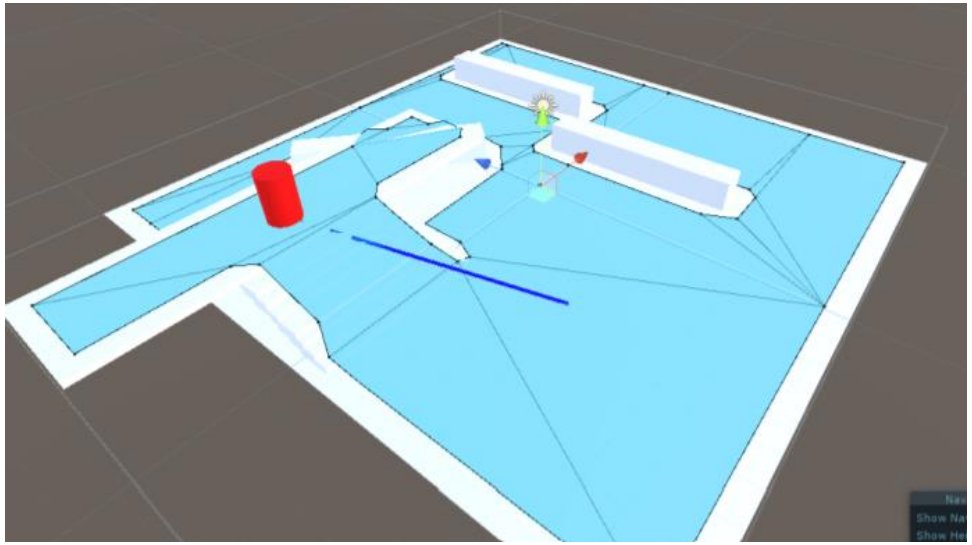
- Adapter: standalone executable
- MMU management: loading & instancing
- Session management
- Scene buffering



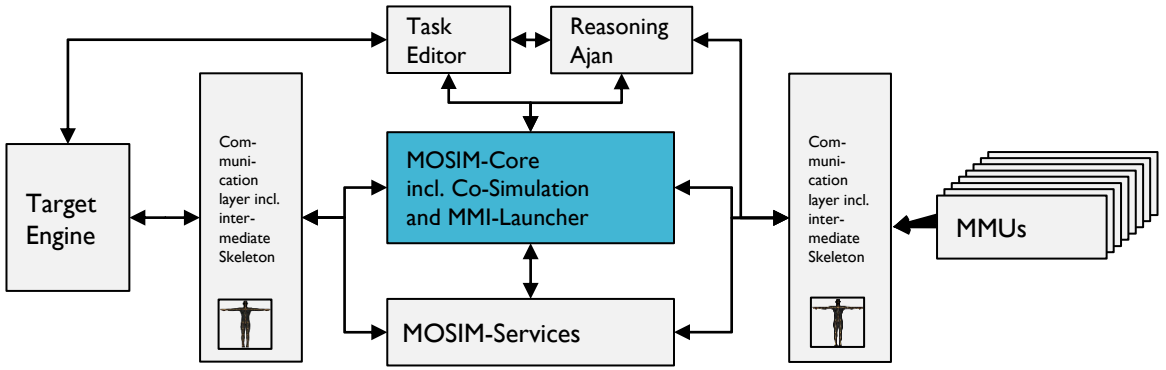
Complete MOSIM Framework: Services

- Provide different, specific functionality, E.g.
  - Path Planning
  - Inverse Kinematics
  - Retargeting
  - Posture Blending
  - …

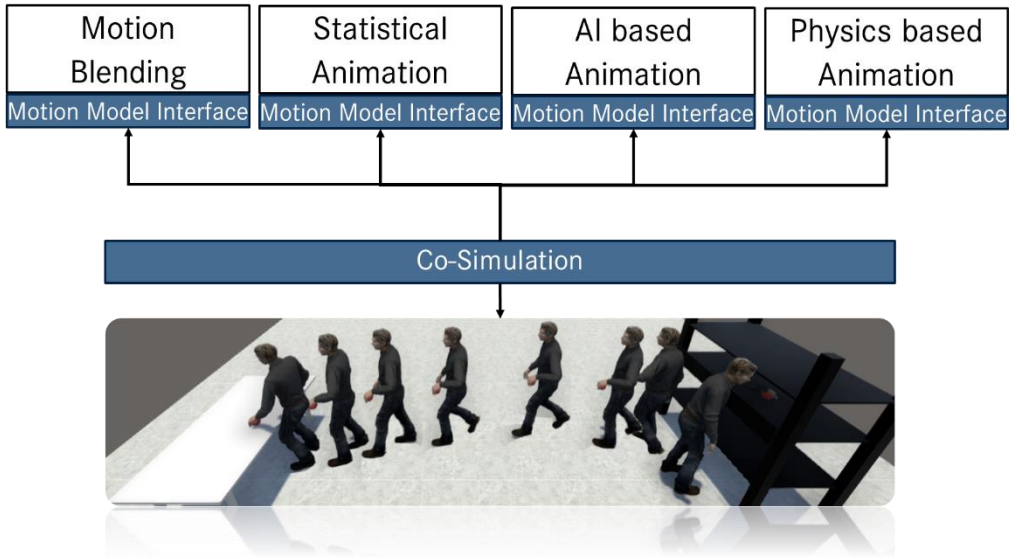




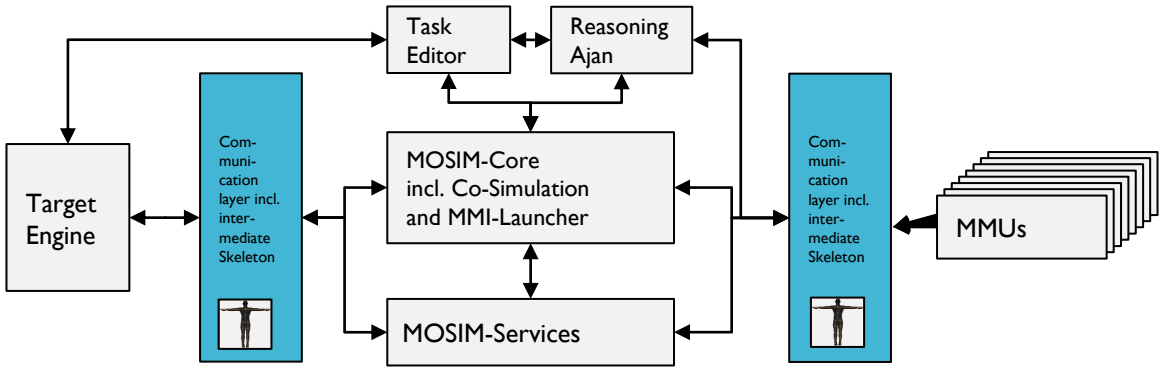
Complete MOSIM Framework: Co-Simulation



- A basic co-simulation approach is provided (C#)
- Integrated Co-Simulation in Unity
- Standalone Co-Simulation



Complete MOSIM Framework: Instructions



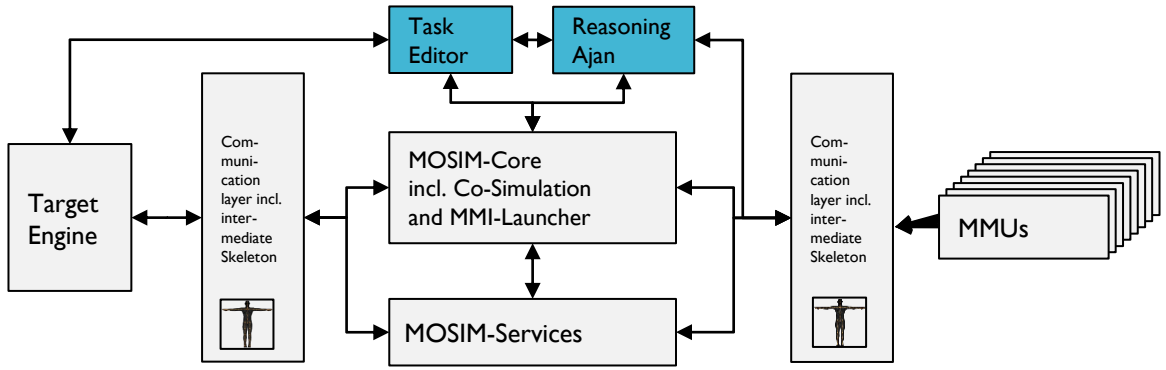
- Used to perform new actions
- Contain ID for identification
- human readable name
- Motion Type: which action to perform

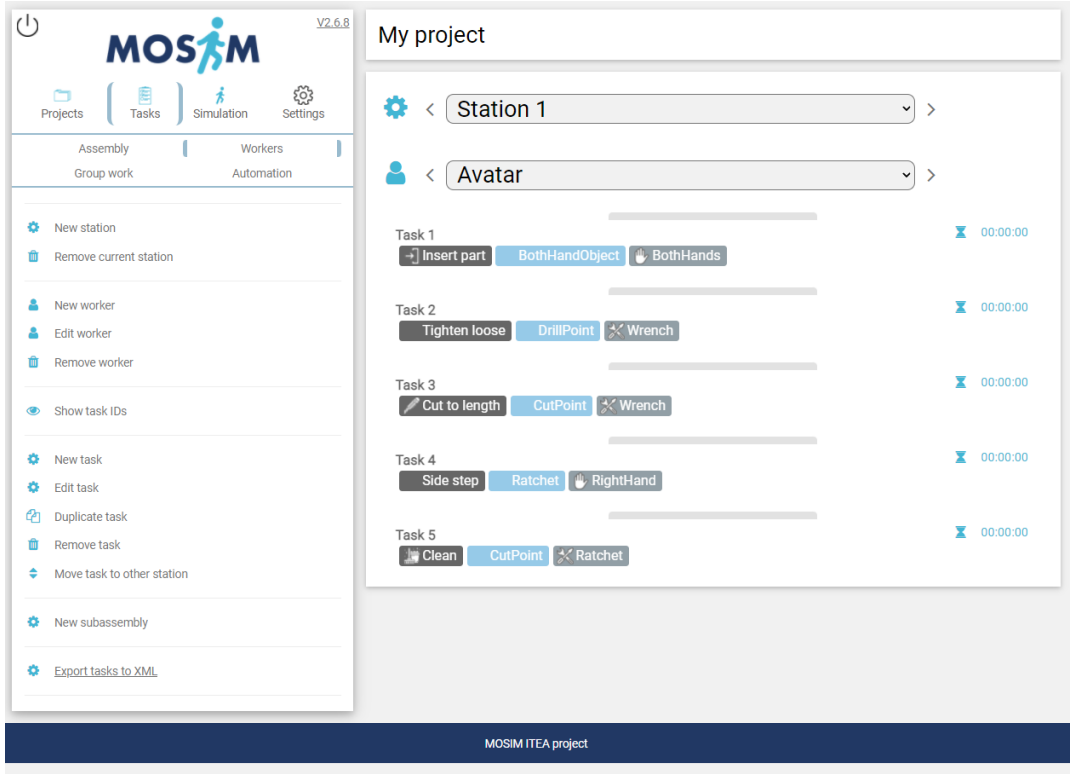
```
struct MInstruction
{
    1: required string ID;
    2: required string Name;
    3: required string MotionType;
    4: optional map<string,string> Properties;
    5: optional list<constraints.MConstraint> Constraints;
    6: optional string StartCondition;
    7: optional string EndCondition;
    8: optional string Action;
    9: optional list<MInstruction> Instructions;
}
```

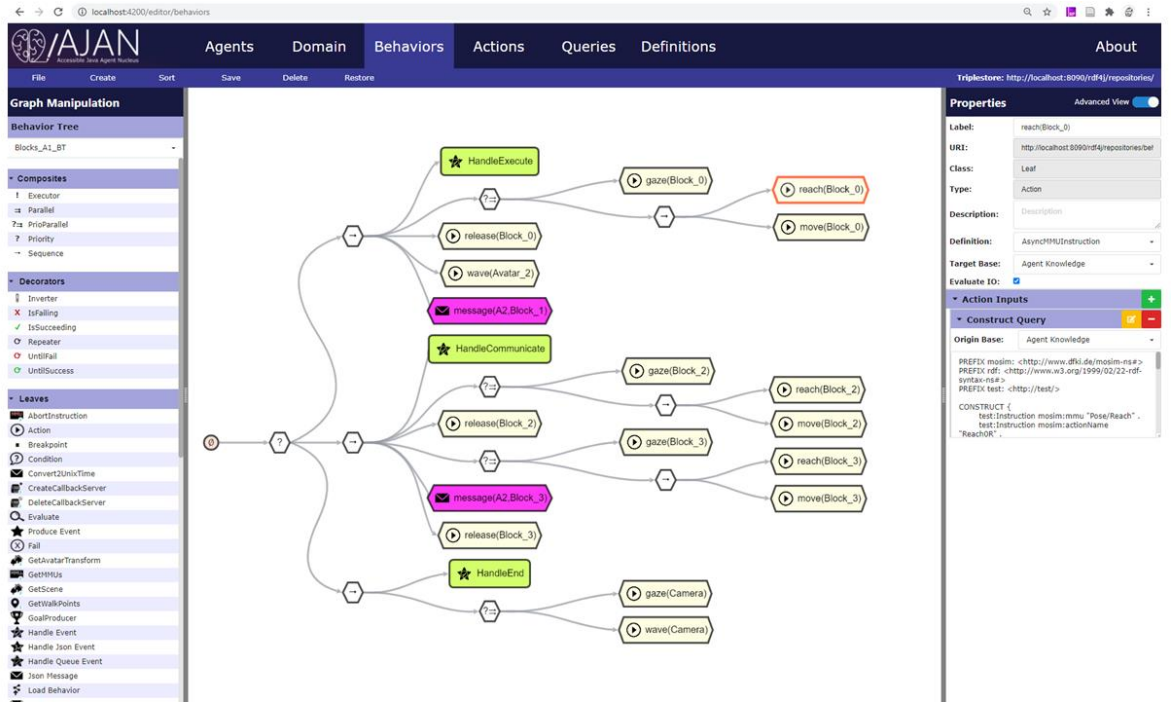
LOADABLE MMUS

NAME	LANGUAGE	MOTION TYPE
CoSimulation	C#	co-simulation
DefaultLocomotion	C#	Locomotion
UnityLocomotionMMU	UnityC#	Locomotion/Walk
CarryMMUConcurrent	C#	Object/Carry
MoveMMUConcurrent	C#	Object/Move
ReleaseMMU	C#	Object/Release
TurnMMU	C#	Object/Turn
AlignBodyMMU	C#	Pose/AlignBody
GazeMMU	C#	Pose/Gaze
GraspMMUSimple	C#	Pose/Grasp
UnityIdleMMU	UnityC#	Pose/Idle
SimpleLookAtMMU	C#	Pose/LookAt
MoveFingersMMU	C#	Pose/MoveFingers
ReachMMUConcurrent	C#	Pose/Reach
IKTestMMU	C#	Test/IK

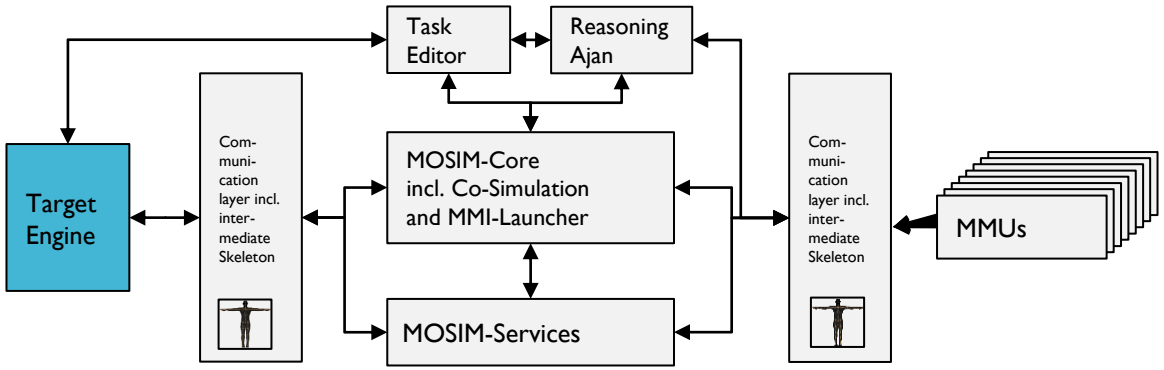
Complete MOSIM Framework: Behavior Modelling



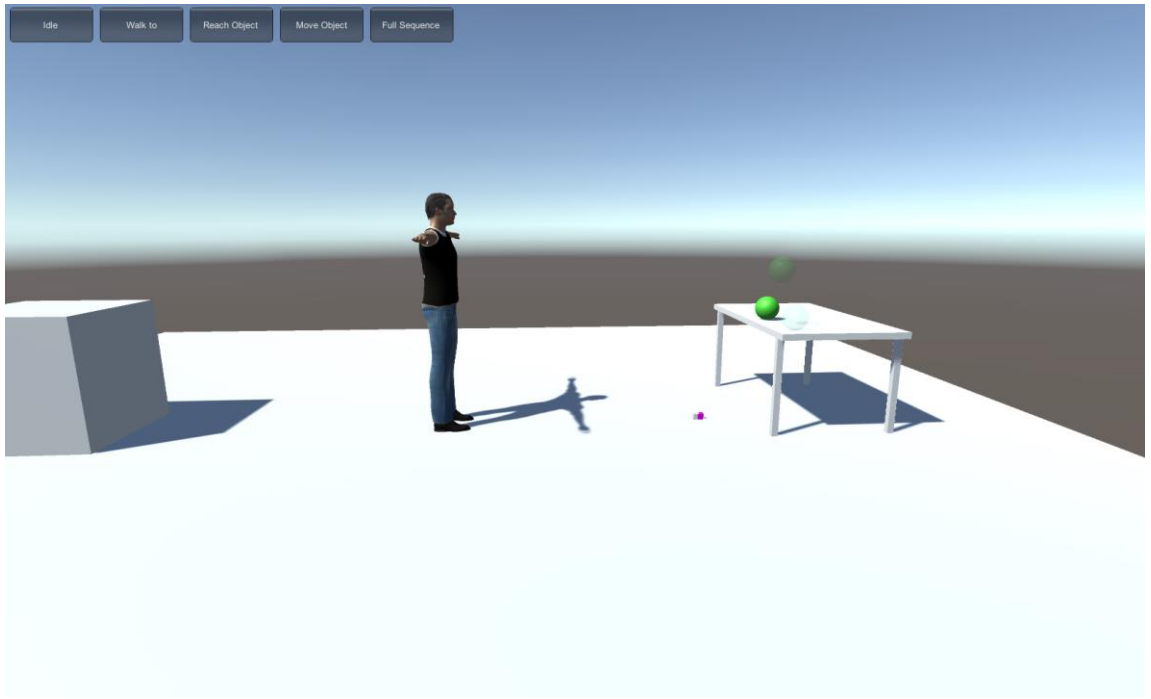




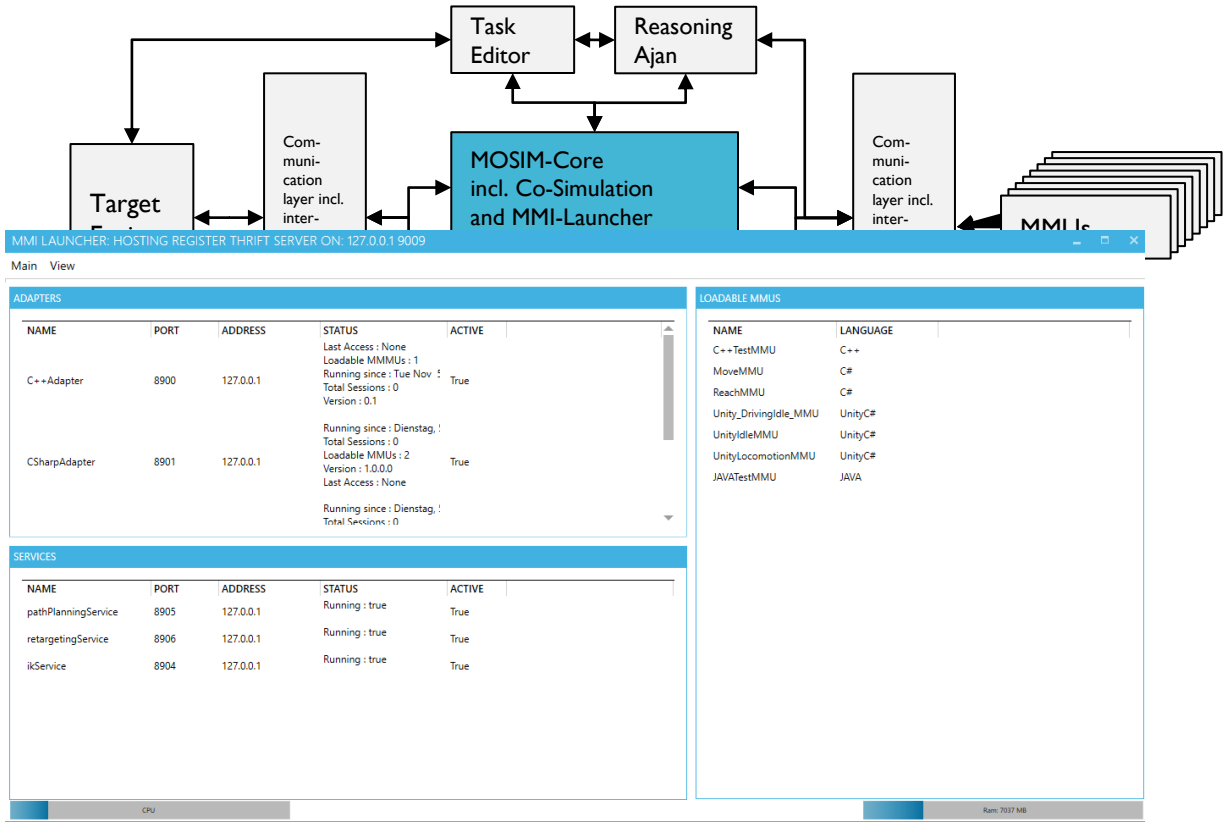
Complete MOSIM Framework: Target Environment



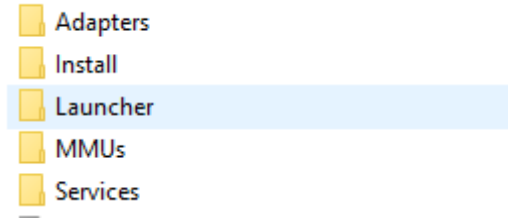
- Simulation Controller
- 3D engine providing the central "tick"



Complete MOSIM Framework: Launcher

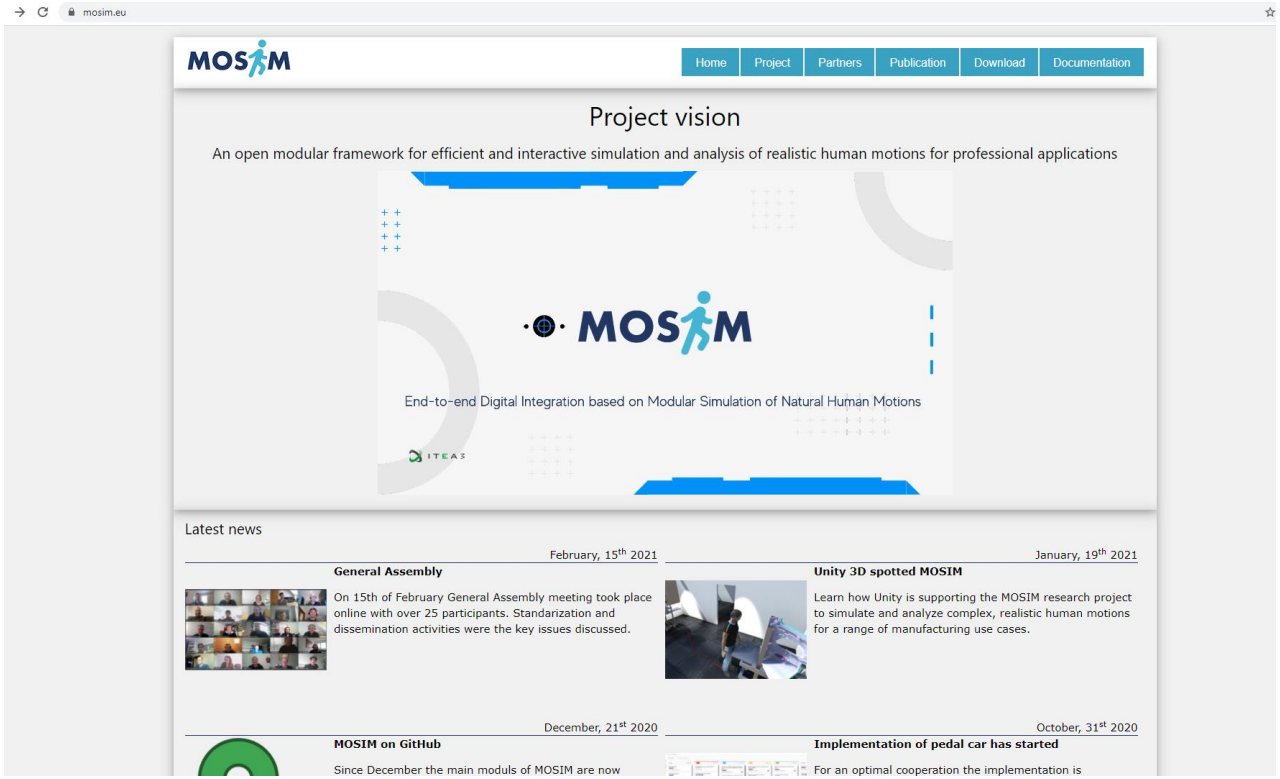


- Central component starting all entities according to folder structure
- Ability to load new Adapters, Services and MMUs by placing them in the respective folder
- Display initialized and running components



Where do I find the MOSIM Framework?

MOSIM Webpage: mosim.eu



Github

- https://github.com/Daimler/MOSIM: MOSIM meta repository
  - https://github.com/Daimler/MOSIM_Core: Core repository containing
    - Language Support (C#, c++, python, Java)
    - Engine Support (Unity, Unreal Engine)
    - Basic MMUs
    - Co-Simulator
    - Launcher
  - https://github.com/Daimler/MOSIM_Services: Service repository containing all Services
    IK, Retargeting, Path Planning, Posture Blending, etc.
  - https://github.com/Daimler/MOSIM_Demos: Demos (Unity, Unreal)
  - https://github.com/Daimler/MOSIM_Tools: Tools repository containing
    MMU Generator, Retargeting Configurator, etc.

Using the Documentation

- Some documentation is provided on https://github.com/Daimler/MOSIM_Core/wiki
- Not always up to date!
- Thrift interface definition can be found in the github core repository https://github.com/Daimler/MOSIM_Core/tree/master/Framework/LanguageSupport/thrift/mmi

How can we run the MMI Framework?

1. Ensure to have an admin account on the system.

Requirements

Running the Framework

- Windows 10 (ideally Pro)
- Admin privilege required at least once
- Decent hardware (e.g. > 6 GB RAM, > 4 Cores)
- Deployed framework files
  or access to framework (e.g. via Docker)
- Target engine project
  (e.g. Unity 3D Editor and Scene, compiled Unity game)

Developing the Framework

- All of the left
- Visual Studio
- Unity Engine

Downloading the latest Release of the Framework from Github

- For documentation visit https://github.com/Daimler/mosim_core/wiki/InstallPrecompiled
- Download the Framework from: https://github.com/Daimler/MOSIM_Core/wiki/Framework-Releases
- Run the "Enable Firewall" script with admin privileges
  - Enables MOSIM components inside the Windows Firewall
- Start the launcher

The Environment folder

- The MMIEnvironment gets released regularly and contains the following folder structure:

```
Environment/
    Adapters/
      CSharpAdapter/
      UnityAdapter
    Launcher/
    MMUs/
      Idle/
      …
    Services/
      IKService/
      …
```

Compile from source

- Documented in https://github.com/Daimler/mosim_core/wiki/CompileFramework
- Clone all repositories with

  meta git clone git@git.hb.dfki.de:mosim/mosim.git
- (Checkout the develop branch on all repositories except for the Tools repository)
- Adjust the values in the deploy_variables.bat script to your system settings
- Run the deploy.bat script
- The deployed framework will open when successfully deployed (location: build/)

How can I Integrate MOSIM into Unity?

Preparation

- Currently supported Unity version: 2019.4.25f1
- Documentation can be found at: https://github.com/Daimler/mosim_core/wiki/IntegratingFramework
- Download the UnityPackage from: https://github.com/Daimler/MOSIM_Core/wiki/Framework-Releases
- Import custom package from
- For different Unity versions, E.g. 2018 or 2020
  - Clone repositories
  - Deploy framework
  - Copy Unity Target engine libraries from Framework/EngineSupport/Unity/MMIUnity.TargetEngine/MMIUnity.TargetEngine/build to your Unity project

Practical Example

Integration of MOSIM Components in Scene (I): Basic Scene Setup

- Scene Setup:
  - Add empty game object (e.g. "Scene")
  - Add Simulation Controller component:
    - Connects to the MOSIM Framework
  - Add Unity Scene Access component:
    - Synchronizes game objects with the framework
    - Game objects require a Scene Object component
    - Only objects below the "Scene" in the hierarchy are synchronized
  - Adjust MMI Settings to your configuration

Integration of MOSIM Components in Scene (I): Scene Objects

- MMI Scene Object (Script Component)
  - Synchronization of objects
  - Must be added to interact with object
  - Place object below "Scene" in hierarchy!

Integration of MOSIM Components in Scene (I): Avatars / Agents

- MMI Avatar (Script Component)
  - Setup Root Transform & Pelvis as in the Skeleton Configurator
  - Provide path to retargeting configuration file
  - Place below "Scene" in hierarchy!

Integration of MOSIM Components in Scene (I): Behavior

- Implement Avatar Behavior class (or similar)
- Attach custom behavior to agent / avatar

Running the Simulation

- Start the Launcher / Framework
- Wait until it is loaded
- Start the target engine

Identifying Sources of Errors

- Check the different components of the MOSIM framework for errors
- If thrift throws an error in the target engine, it is probably due to some component failing
- Typical errors contain:
  - Wrong / Missing retargeting configuration during simulation start
  - Missing scene object on Assign Instruction
  - Missing MMU / motion type if there is no action performed
  - No path found for locomotion MMUs (the avatar will not walk to the target), e.g. due to target being inside a collision boundary

Q & A

Thank you for your attention!

End-to-end Digital Integration based on Modular Simulation of Natural Human Motions