VR

*INTERACTIVITY PACK*

*Project S.D.F.*

*Andrea Carver*

*Loyola Marymount University*

*CMSI 402 – Individual Senior Thesis*

*Table of Contents ––––––––––––––––––––––*

To be completed with final draft of S.D.F.

*1.0 – Project Status Sheets ––––––––––––––*

*February 19, 2018*

Accomplishments since last report:

* Became familiar with Unreal Engine 4 (UE4) C++ programming through viewing and practicing with online tutorials.
* 3D-modeled geometry for visualizing grab points and geometry for visualizing axis symmetry. Created final materials in UE4.
* Designed detailed plan of fields and methods for all classes.
* Calculated 3D math / trigonometry equations for various methods.

Scheduled tasks to be done by next report:

* At the very minimum, stub all code and have all methods successfully passing dummy values to each other.
* Fix issue with geometry file pointer.
* Create all hand rig poses.
* Create the animation blueprint for the hand, which will control how the digital hands move in relation to the relevant fields and methods. Play-test it to ensure it is working effectively.

Noteworthy risks, concerns, or problems:

* Having issues getting a pointer to geometry files in C++. Any other code will be difficult to debug if there is no visual indicator of grab point transform.
* Cannot work or play-test on Mac laptop as I am working with Microsoft Visual Studio (Windows-only) and my laptop is not graphically capable of supporting the Oculus Rift. This limits my development time to when I am home.

*March 18, 2018*

Accomplishments since last report:

* Wrote files for all custom data types (mostly enums).
* Started implementing methods.
* Configured each class to be used properly by the game engine (using UE4 C++ macros).

Scheduled tasks to be done by next report:

* Fix geometry pointer issue and create visual display system.
* Create test objects to play-test system with.

Noteworthy risks, concerns, or problems:

* Oculus software had some major downtime during this work period. (Enough for them to distribute in-store credit!) Couldn’t play-test.
* Severe lack of time to dedicate to this endeavor due to job-hunting.

*April 2, 2018*

Accomplishments since last report:

* C++ proved to be too difficult to get a handle on in the allotted time, so I am porting my work over to the quicker UE4 Blueprints system. This is basically C++ code, but with visual coding (using nodes, connections, and a UI).
* Figured out a major flaw with my system: making all of my classes be components. Making them be actors (that contain other components themselves) has alleviated several confusing problems.
* Re-stubbed all the classes in Blueprints, implemented a few methods.

Scheduled tasks to be done by next report:

* Get as far as possible with the Pickup actor. I have already prototyped a version of this before, so it should be faster than the other interaction types.
* Create test objects to play-test system with.

Noteworthy risks, concerns, or problems:

* Could not resolve issues with loading geometry in C++. Debugged for hours, but could not ascertain a way to dynamically load geometry without crashing the editor. However, was able to accomplish this with Blueprints in around 10 minutes!

*April 16, 2018*

Accomplishments since last report:

* Successfully implemented a MVP for the pickup mechanic.
* Created test objects to play-test system with.

Scheduled tasks to be done by next report:

* Match digital hand orientation more closely to real hand orientation.
* Fix orientation issues when grabbing with right hand.
* Figure out a more permanent solution for correctly positioning objects relative to hand when grabbed.

Noteworthy risks, concerns, or problems:

* None- smooth sailing this time.

*2.0 – Preliminary Project Proposal ––––––*

*Project Proposal*

The arrival of consumer virtual reality (VR) is accelerating the desire for enterprise solutions that utilize VR. Yet, many game engines used to create VR experiences lag behind the progress in hardware. Critical functionalities are still missing, forcing content creators to start from scratch with each project. Content quality and development speed suffer as a consequence.

Unreal Engine 4, a leader among these game engines, needs a set of modular, reusable components that can expedite the creation of VR content. Specifically, Unreal Engine 4 needs a pre-built system for grabbing objects, locomotion, and managing inventory in virtual reality. Nearly every VR experience will use these mechanics; hence, designing them well once will be extremely profitable.

In my individual senior project, I will finish my work from summer 2017 on fulfilling these needs. I will be programming primarily in C++ with occasional support through Unreal Engine 4’s visual scripting language. If possible, I will package my work as a downloadable plugin for public use. In summary, I will create reusable mechanics that will make VR experience creation in Unreal Engine 4 more accessible to non-programmers.

*3.0 – Project Proposal ––––––––––––––––––*

*3.1 Project Description*

The VR Interactivity Pack is a plugin for Unreal Engine 4. It aims to close the gap in available software for virtual reality developers who may have a more artistic than technical background. Such developers may not be able to program fundamental virtual reality mechanic themselves, forcing them to abandon their creative visions for virtual reality experiences.

Given the state of virtual reality hardware development, the most important, non-trivial ability in virtual reality is the ability to pick up objects. Disappointingly, major game engines like Unreal Engine 4 do not offer templated starter solutions for picking up and interacting with objects. The VR Interactivity Pack, using C++ programming and Unreal Engine scripting macros, will provide these necessary templates.

After installing the plugin, users will use the templated interactions by adding one of the following components to an Actor (an object in Unreal Engine 4): a Simple Grab Component, a Twist Grab Component, a Piston Grab Component, or a Lever Grab Component. Each grab component allows the player to grab and interact with the component as specified by the grab type. For instance, a Twist Grab Component can be grabbed, twisted a certain amount, and released.

Each of the components can be configured more specifically to meet the user’s interaction needs. For instance, a Twist Grab Component could only be twisted a certain number of degrees, a Piston Grab Component could only be pulled out a certain length, and a Lever Grab Component could only be rotated a certain number of degrees.

On their own, the grab components are invisible, only offering the interaction functionality. Each different type of grab component will have one or more attributes where users can input geometry to ‘skin’ the grab components. This way, the grab components can have any customized appearance.

Project maintenance will be performed by myself, Andrea Carver. Project updates are likely to occur since I am one of the end users for this project. As I use the plugin and move on to more advanced interaction types, I will be expanding the types of grab components. Possible future grab component types include omnidirectional lever grab, elastic grab, and string grab.

*3.2 Justification*

Completing this project will require me to synthesize knowledge gained from both the animation and computer science department. I must have familiarity with the target platform, Unreal Engine 4, as well as the programming language, C++. The largest challenge of the project will likely be the programming itself, as the computer science major has focused on more modern programming languages. I have a high level of familiarity with the game engine, but not with C++. However, I wish to become more proficient in C++ so that I may apply lessons from computer science classes to future virtual reality projects. Hence, this project is a perfect way for me to gain experience.

*5.0 – Software Requirements Specification*

*5.1 Introduction*

The VR Interactivity Pack provides two core components: a specialized motion controller actor and a new component for grabbing functionality. The Motion Controller Actor is intended to be used as a parent class, from which users can create a child class to meet their more specific needs. The Interactivity Component is intended to be added as a component to actors that need to be interacted with; it adds the capability to interact with the player. As seen in the diagram below, the Motion Controller Actor and Interactivity Component do not exist only on their own, but are either directly connected to an actor/pawn as a component or pointed to with a pointer.

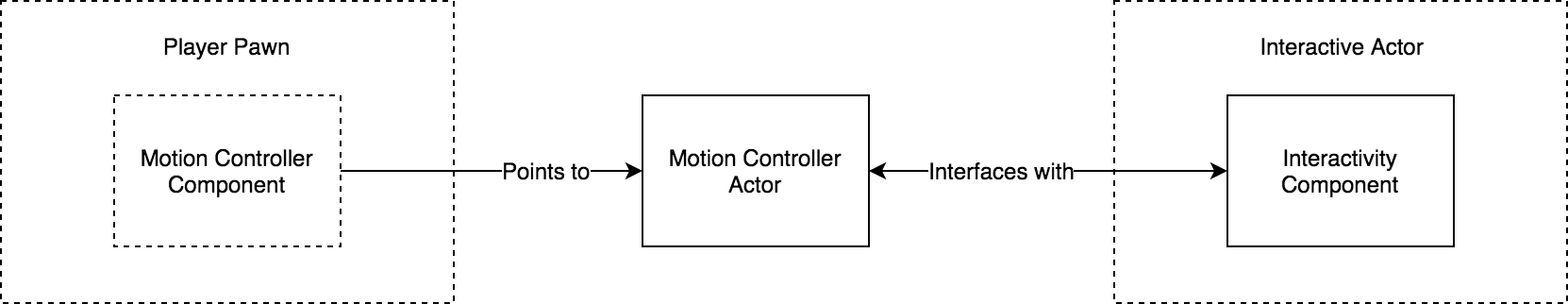


Fig 5A: System components diagram. Entities created by the plugin are rendered with solid lines while entities created by the user are rendered with dotted lines.

The remainder of section 5 is structured as follows. Section 5.2 contains functional requirements, section 5.3 contains performance requirements, and section 5.4 contains environment requirements. Functional requirements specify what the plugin should be able to do while performance requirements specify how the plugin should meet those functional requirements. Environment requirements specify what platforms, hardware, and software are required to run or develop the plugin.

*5.2 Functional Requirements*

Section 5.2 describes various abilities the plugin must offer the user. Each type of component is listed, as well as their respective functional requirements.

5.2.1 Motion Controller Actor

The Motion Controller Actor provides base functionality necessary to properly interact with the various types of interactivity components. It detects the movement of motion controller hardware, handles button inputs, computes logic associated with hand functionality, animates hand geometry, and vibrates the motion controller hardware to simulate tactile feedback.

5.2.1.1 The Motion Controller Actor shall allow the user to select between several existing hand geometry options / visual looks.

5.2.1.2 The Motion Controller Actor shall allow the user to enable only certain buttons on the motion controller hardware.

5.2.1.3 The Motion Controller Actor shall allow the user to switch a hand between three states: active, disabled, and hidden. An active hand is rendered and interactive, a disabled hand is rendered but not interactive, and a hidden hand is not rendered and not interactive.

5.2.2 Pickup Interactivity Component

A Pickup Interactivity Component allows an object to be picked up, appearing to attach itself to the hand that grabbed it. The user can position and rotate the grab point on the object, increase or decrease movement force to simulate weight, select the approximated shape of the surface, choose the axis of symmetry of the grab, and select whether or not to send tactile feedback to the Motion Controller Actor.

5.2.2.1 The Pickup Interactivity Component shall be able to be added to an actor as an actor component.

5.2.2.2 The Pickup Interactivity Component shall display a visual indicator of the component’s position, rotation, and shape.

5.2.2.3 The visual indicator mentioned in the previous requirement shall have an option for being visible either in editor, editor and in game, or nowhere. It shall default to visible in editor.

5.2.2.4 The Pickup Interactivity Component shall have an option to disable interactions with the Motion Controller Actor. This temporarily prohibits the Motion Controller Actor from picking up the object.

5.2.2.5 The Pickup Interactivity Component shall allow the user to specify the weight of an object. This value determines whether or not an object can be moved by a hand, and if so, how immediately or slowly the object moves.

5.2.2.6 The Pickup Interactivity Component shall allow the user to select whether the grab point symmetry axis is on the X, Y, or Z axis. Grab point symmetry determines how each hand grabs the handle in relation to one another. Handles with X-axis symmetry are like pistol handles, handles with Y-axis symmetry are like coffee cup handles, and handles with Z-axis symmetry are like drawer handles.

5.2.2.7 The Pickup Interactivity Component shall allow the user to choose the approximated shape of the grabbed surface. The possible shapes shall include flat, large sphere, medium sphere, small sphere, tiny sphere, large cylinder, medium cylinder, small cylinder, tiny cylinder, gun, and drawstring. The shape of the grabbed surface determines how the hand will position around the surface.

5.2.2.8 The Pickup Interactivity Component shall allow the user to choose the intensity of tactile feedback to send to the Motion Controller Actor.

5.2.3 Button Interactivity Component

A Button Interactivity Component allows an object to contain a button which toggles between enabled and disabled states. The user can select button geometry, select button housing geometry, specify an enabled state material and a disabled state material for the button, configure how far the button must be pressed before it toggles state, and select which tactile feedback to send to the Motion Controller Actor.

5.2.3.1 The Button Interactivity Component shall be able to be added to an actor as an actor component.

5.2.3.2 The Button Interactivity Component shall display a visual indicator of the component’s position, rotation, and shape.

5.2.3.3 The visual indicator mentioned in the previous requirement shall have an option for being visible either in editor, editor and in game, or nowhere. It shall default to visible in editor.

5.2.3.4 The Button Interactivity Component shall have an option to disable interactions with the Motion Controller Actor. This freezes the state of the component.

5.2.3.5 The Button Interactivity Component shall allow the user to select the button geometry.

5.2.3.6 The Button Interactivity Component shall allow the user to select the button housing geometry.

5.2.3.7 The Button Interactivity Component shall allow the user to choose the material applied to the button when the state is enabled and the material applied to the button when the state is disabled.

5.2.3.8 The Button Interactivity Component shall allow the user to specify how far the button must be pressed before it registers a press.

5.2.3.9 The Button Interactivity Component shall allow the user to specify the button’s logic type. Either the button toggles between two states on a press (example: on or off button) or the button cues an event on a press (example: fire missile button).

5.2.3.10 The Button Interactivity Component shall allow the user to choose the intensity of tactile feedback to send to the Motion Controller Actor.

5.2.4 Switch Interactivity Component

A Switch Interactivity Component allows an object to contain a switch which toggles between enabled and disabled states. The switch component is intended to simulate a tiny lever, such as a light switch, that would be too small to implement as a full lever component. The user can select switch geometry, select switch housing geometry, specify the angle range the switch toggles within, specify to what angle the switch must be rotated before it toggles state, and select which tactile feedback to send to the Motion Controller Actor.

5.2.4.1 The Switch Interactivity Component shall be able to be added to an actor as an actor component.

5.2.4.2 The Switch Interactivity Component shall display a visual indicator of the component’s position, rotation, and shape.

5.2.4.3 The visual indicator mentioned in the previous requirement shall have an option for being visible either in editor, editor and in game, or nowhere. It shall default to visible in editor.

5.2.4.4 The Switch Interactivity Component shall have an option to disable interactions with the Motion Controller Actor. This freezes the state of the component.

5.2.4.5 The Switch Interactivity Component shall allow the user to select the switch geometry.

5.2.4.6 The Switch Interactivity Component shall allow the user to select the switch housing geometry.

5.2.4.7 The Switch Interactivity Component shall allow the user to specify the angle range that the switch may rotate between.

5.2.4.8 The Switch Interactivity Component shall allow the user to specify the default toggled state that the switch will start at.

5.2.4.9 The Switch Interactivity Component shall allow the user to specify the change in angle that the switch must be rotated to toggle the state.

5.2.4.10 The Switch Interactivity Component shall allow the user to choose the intensity of tactile feedback to send to the Motion Controller Actor.

5.2.5 Twist Interactivity Component

The Twist Interactivity Component allows an object to contain a twistable knob which turns to control a float value. The twist component is intended to simulate objects such as dials or doorknobs. The user can select twist component geometry, select twist component housing geometry, specify the angle range that the twist component may be twisted, specify a default value, specify minimum and maximum output values, and select which tactile feedback to send to the Motion Controller Actor.

5.2.5.1 The Twist Interactivity Component shall be able to be added to an actor as an actor component.

5.2.5.2 The Twist Interactivity Component shall display a visual indicator of the component’s position, rotation, and shape.

5.2.5.3 The visual indicator mentioned in the previous requirement shall have an option for being visible either in editor, editor and in game, or nowhere. It shall default to visible in editor.

5.2.5.4 The Twist Interactivity Component shall have an option to disable interactions with the Motion Controller Actor. This freezes the state of the component.

5.2.5.5 The Twist Interactivity Component shall allow the user to select the twist knob geometry.

5.2.5.6 The Twist Interactivity Component shall allow the user to select the twist knob housing geometry.

5.2.5.7 The Twist Interactivity Component shall allow the user to specify the angle range that the twist knob may be twisted.

5.2.5.8 The Twist Interactivity Component shall allow the user to specify a default angle value that the twist knob will start at.

5.2.5.9 The Twist Interactivity Component shall allow the user to specify what value the component will output at the minimum angle and what value the component will output at the maximum angle.

5.2.5.10 The Twist Interactivity Component shall allow the user to choose the intensity of tactile feedback to send to the Motion Controller Actor.

5.2.6 Lever Interactivity Component

The Lever Interactivity Component allows an object to contain a rotatable knob which rotates to control a float value. The user can select lever component geometry, select lever component housing geometry, specify the angle range that the lever component may be rotated, specify a default value, specify minimum and maximum output float values, and select which tactile feedback to send to the Motion Controller Actor.

5.2.6.1 The Lever Interactivity Component shall be able to be added to an actor as an actor component.

5.2.6.2 The Lever Interactivity Component shall display a visual indicator of the grab area’s position, rotation, and shape, as well as the origin of rotation of the lever.

5.2.6.3 The visual indicator mentioned in the previous requirement shall have an option for being visible either in editor, editor and in game, or nowhere. It shall default to visible in editor.

5.2.6.4 The Lever Interactivity Component shall have an option to disable interactions with the Motion Controller Actor. This freezes the state of the component.

5.2.6.5 The Lever Interactivity Component shall allow the user to select the lever geometry.

5.2.6.6 The Lever Interactivity Component shall allow the user to select the lever housing geometry.

5.2.6.7 The Lever Interactivity Component shall allow the user to specify the angle range that the lever component may be rotated.

5.2.6.8 The Lever Interactivity Component shall allow the user to specify a default angle value that the lever will start at.

5.2.6.9 The Lever Interactivity Component shall allow the user to specify what value the component will output at the minimum lever angle and what value the component will output at the maximum lever angle.

5.2.6.10 The Lever Interactivity Component shall allow the user to choose the intensity of tactile feedback to send to the Motion Controller Actor.

5.2.6.11 The Lever Interactivity Component shall allow the user to enable or disable ‘simulate physics’ on the lever. As an example, this would accomplish the swinging of a doorway after a force is applied on the handle and then released.

*5.3 Performance Requirements*

Section 5.3 details how the plugin must accomplish the functional requirements from Section 5.2. Performance requirements aim to optimize the speed and storage of the relevant software, among other things.

Since the scale of this project is minimal and is already running in a fast, real-time engine, most of the requirements of Section 5 are functional, rather than performance-related.

5.3.1 Motion Controller Actor

5.3.1.1 The file size of the hand geometries shall not exceed 2 MB.

5.3.1.2 The Motion Controller Actor shall match the motion of the motion controller hardware with no perceptible delay.

5.3.2 Pickup Interactivity Component

5.3.2.1 The file size of the shape indicator geometries shall not exceed 1.5 MB.

5.3.2.2 When the user presses grip button on the motion controller, the Pickup Interactivity Component shall respond immediately.

5.3.3 Button Interactivity Component

5.3.3.1 The file size of the placement indicator geometries shall not exceed 100KB.

5.3.3.2 When the user pushes into a button with a motion controller, the Button Interactivity Component shall respond immediately.

5.3.4 Switch Interactivity Component

5.3.4.1 The file size of the placement indicator geometries shall not exceed 100KB.

5.3.4.2 When the user pushes into a switch with a motion controller, the Switch Interactivity Component shall respond immediately.

5.3.5 Twist Interactivity Component

5.3.5.1 The file size of the placement indicator geometries shall not exceed 100KB.

5.3.5.2 When the user pushes into a switch with a motion controller, the Twist Interactivity Component shall respond immediately.

5.3.6 Lever Interactivity Component

5.3.6.1 The file size of the placement indicator geometries shall not exceed 200KB.

5.3.6.2 When the user grabs a lever with a motion controller, the Lever Interactivity Component shall respond immediately.

*5.4 Environment Requirements*

The following two lists detail the necessary hardware and software necessary to either develop or use the VR Interactivity Pack.

5.4.1 Development Environment Requirements

The following resources are necessary to develop the VR Interactivity Pack plugin.

Processor 2.5 GHz or more

Hard Drive Space 200MB

RAM 8GB

Display 1920x1080

Sound Card Optional

Operating System Windows 7 or later

Compiler Microsoft Visual Studio Community Edition 2015

3D Modeling Blender 3D

Game Engine Unreal Engine 4.18

VR HMD Hardware Oculus Rift Consumer Version 1

Motion Controllers Oculus Motion Controllers

5.4.2 Execution Environment Requirements

The following resources are necessary to install and use the VR Interactivity Pack plugin.

Processor 2.5 GHz or more

Hard Drive Space 200MB

RAM 8GB

Display 1920x1080

Sound Card Optional

Game Engine Unreal Engine 4.18

VR HMD Hardware Oculus Rift Consumer Version 1

Motion Controllers Oculus Motion Controllers

*6.0 – Software Design Description –––––––*

*6.1 Introduction*

This document presents the architecture and detailed design for the VR Interactivity Pack’s software. The VR Interactivity Pack is a small virtual reality plugin for Unreal Engine 4 that adds a handful of basic interaction presets.

6.1.1 System Objectives

The VR Interactivity Pack fills a certain software gap in Unreal Engine 4, a game engine that is missing crucial virtual reality interaction presets. Artists and beginning programmers thus are limited in their game development capabilities. The VR Interactivity Pack aims to equip them with the tools to quickly prototype VR games and experiences. The pack provides a handful of basic interaction styles, such as buttons, switches, and levers. Developers can add these interaction styles as components to their actors.

6.1.2 Hardware, Software, and Human Interfaces

6.1.2.1 A VR-ready GPU is highly recommended for the end user. According to the company Oculus, the minimum spec GPU is a GTX 980.

6.1.2.2 Either the DK2 or the CV1 version of the Oculus Rift headset is absolutely necessary to develop and use the VR Interactivity Pack.

6.1.2.3 Two Oculus Touch motion controllers are necessary to develop and use the pack. A Vive headset and motion controllers will not work properly, as the pack takes advantage of the Oculus Touch’s unique capability to approximate finger pose.

6.1.2.4 One Oculus motion tracking sensor is required. Two are preferred, however, for the increased tracking accuracy and coverage.

6.1.2.5 A mouse, keyboard, and high-definition monitor are necessary to develop and use the pack.

6.1.2.6 The Windows 7 operating system is necessary to develop the pack.

6.1.2.7 The Unreal Engine 4.18 game engine software is necessary to develop and use the pack.

6.1.2.8 Microsoft Visual Studio is necessary to develop the pack.

*6.2 Architectural Design*

This section provides an overview of the VR Interactivity Pack’s technical design.

6.2.1 Major Software Components

The VR Interactivity Pack consists of two major software components, consisting of the Hand Component and the Interactivity Components.

The end user spawns two Hand Components and specifies whether each is the right or left hand. The Hand Components track to the Oculus Touch motion controllers and pay attention to the button inputs. They issue grab commands to ‘grabbable’ interactivity components and forward button input data to held actors with pickup interactivity components.

The Interactivity Components house all of the interaction functionality except for calculating which actor is grabbed when the grab button is pushed. Depending on their subclass type, they will behave differently and generate different events upon interaction with a Hand Component.

6.2.2 Major Software Interactions

The most crucial aspect of the VR Interactivity Pack is how the Hand Component interacts with the Interactivity Components. Depending on the Hand Component’s transform (or more specifically, its position and rotation), the Interactivity Components will change states. The different state changes will be triggered in a manner that matches real-life, physical interactions. For instance, when a Hand Component pushes the button surface in a Button Interactivity Component past a certain point, the Button Interactivity Component will trigger a “press” event.

6.2.3 Architectural Design Diagrams

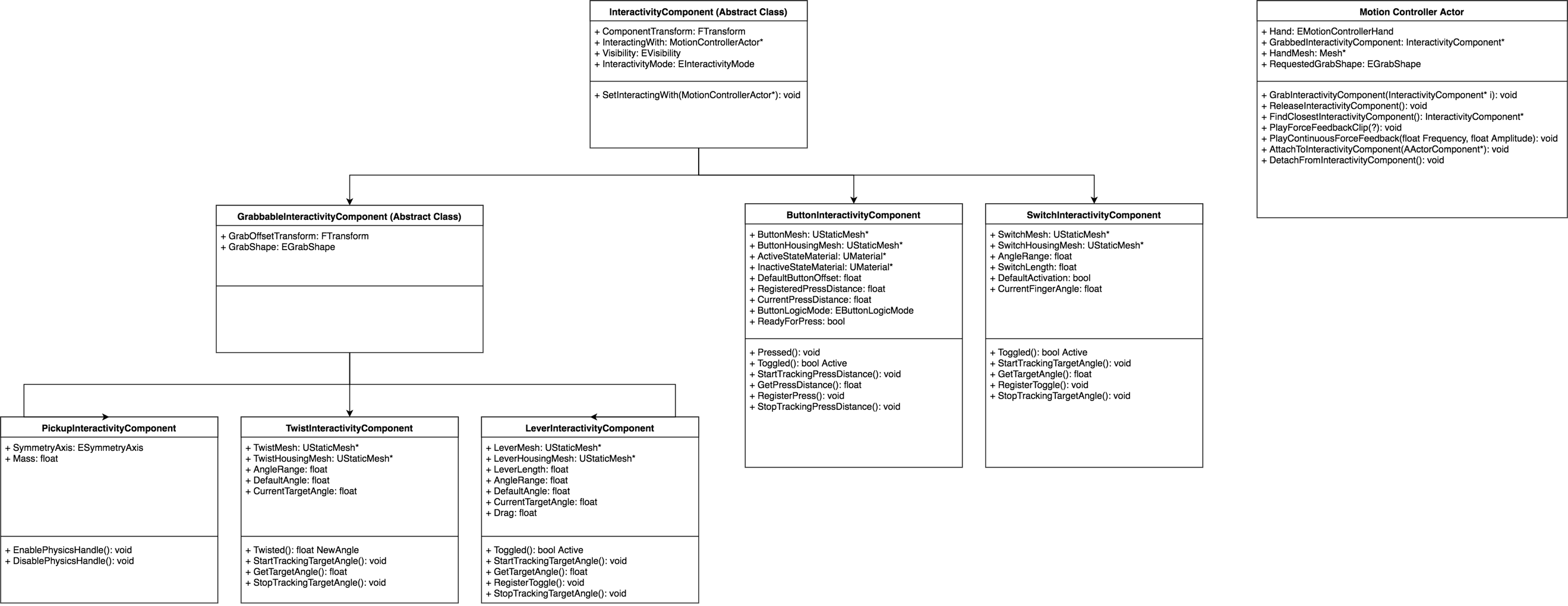


Fig 6A: Hand component class diagram.

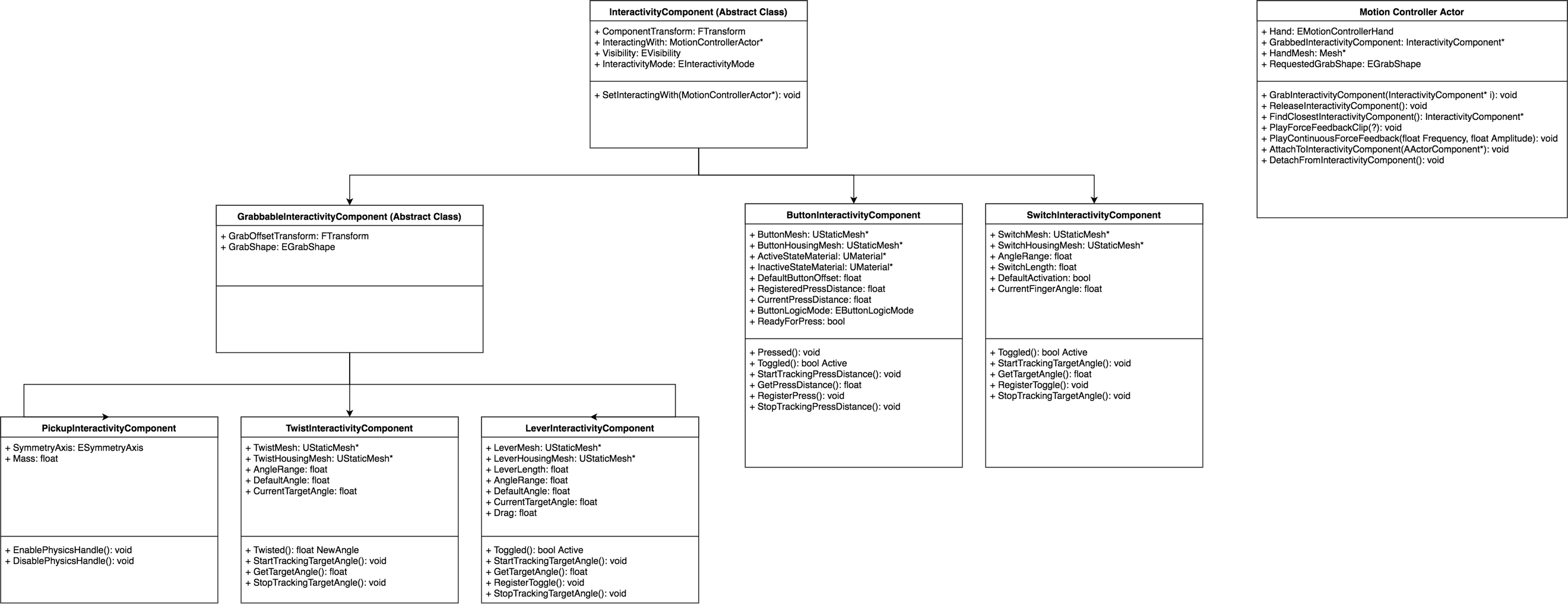


Fig 6B: Interactivity components class diagram.

*6.3 CSC and CSU Descriptions*

This section describes in precise detail the architectural design of the VR Interactivity Pack.

6.3.1 Class Descriptions

The following sections provide details about all classes used in the VR Interactivity Pack.

6.3.1.1 HandComponent Class: Holds all functionality for the motion controller hands.

Fields:

* Hand: EControllerHand. Specifies whether this is the right or left hand.
* GrabbedInteractivityComponent: InteractivityComponent\*. A pointer to the Interactivity Component currently in hand.
* HandMesh: UAsset\*. A pointer to the skeletal mesh for the hand. Allows the hand to look differently.
* RequestedGrabShape: EGrabShape. A field to store the grab shape the hand should animate to.

Methods:

* GrabInteractivityComponent(InteractivityComponent\* i): void. High level control over the grab routine.
* ReleaseInteractivityComponent(): void. Releases currently grabbed interactivity component.
* FindClosestInteractivityComponent(): InteractivityComponent\*. Used in the main grab method to calculate which interactivity component to grab.
* PlayForceFeedbackClip(AudioClip\*): void. Plays provided force feedback on the motion controllers.
* PlayContinuousForceFeedback(float Frequency, float Amplitude): void. Plays a constant vibration, rather than an audio clip.
* AttachToInteractivityComponent(AActorComponent\*): void. Attaches hand mesh to grabbed interactivity component.
* DetachFromInteractivityComponent(): void. Detaches hand mesh from grabbed interactivity component.

6.3.1.2 InteractivityComponent Abstract Class: Base class for all interactivity components.

Fields:

* InteractingWith: MotionControllerActor\*. Pointer to hand that is currently holding this interactivity component.
* Visibility: EVisibility. Visibility mode for this interactivity component.
* InteractivityMode: EInteractivityMode. Whether or not this interactivity component can be interacted with.

Methods:

* SetInteractingWith(MotionControllerActor\*): void.

6.3.1.3 GrabbableInteractivityComponent Abstract Class: Base class for all interactivity components that require grabbing.

Fields:

* GrabOffsetTransform: FTransform. Allows the grab point to be offset from the mesh origin to its correct position/rotation.
* SurfaceShape: ESurfaceShape. Approximate surface shape that the hand will animate to hold.

6.3.1.4 PickupInteractivityComponent Class: Class for interactive objects that can be pickup up.

Fields:

* SymmetryAxis: ESymmetryAxis. Defines how the left hand will grab in relation to the right hand’s orientation.
* Mass: float. Defines how much weight to simulate. If heavy, the object will move less and slowly. If light, the object will move quickly and immediately.

Methods:

* EnablePhysicsHandle(): void. Start simulating a grab.
* DisablePhysicsHandle(): void. Stop simulating a grab.

6.3.1.5 TwistInteractivityComponent Class: Class for interactive objects that can be twisted.

Fields:

* TwistMesh: UStaticMesh\*. Mesh for the twistable part.
* TwistHousingMesh: UStaticMesh\*. Mesh for the part that does not move.
* AngleRange: float. How much the component can be twisted.
* DefaultAngle: float. What angle the component starts at.
* CurrentTargetAngle: float. Stores the current twisted angle of the component.

Methods:

* Twisted(): float NewAngle. Dispatches angle of the twist component to any actor that may be listening.
* StartTrackingTargetAngle(): void. Begins a twist interaction.
* GetTargetAngle(): float. Calculate the change in angle for the current frame.
* StopTrackingTargetAngle(): void. Ends a twist interaction.

6.3.1.6 LeverInteractivityComponent Class: Class for interactive objects that can be pulled like a lever.

Fields:

* LeverMesh: UStaticMesh\*. Mesh for the moveable part.
* LeverHousingMesh: UStaticMesh\*. Mesh for the part that does not move.
* LeverLength: float. How far away the grip should be from the rotation origin.
* AngleRange: float. How much the lever can be rotated.
* DefaultAngle: float. The starting angle.
* CurrentTargetAngle: float. The current angle the component is trying to reach.
* Drag: float. How slowly or quickly the lever catches up to the target rotation.

Methods:

* Toggled(): bool Active. Dispatches to any listening actors whether or not the current angle is above or below zero degrees.
* StartTrackingTargetAngle(): void. Begins a lever interaction.
* GetTargetAngle(): float. Calculates what angle the lever should rotate to in order to match the position of the hand mesh.
* StopTrackingTargetAngle(): void. Ends a lever interaction.

6.3.1.7 ButtonInteractivityComponent Class: Class for interactive objects that can be pushed like a button.

Fields:

* ButtonMesh: UStaticMesh\*. The part that moves.
* ButtonHousingMesh: UStaticMesh\*. The part that stays stationary.
* ActiveStateMaterial: UMaterial\*. Material applied to the ButtonMesh if the state is active.
* InactiveStateMaterial: UMaterial\*. Material applied to the ButtonMesh if the state is inactive.
* DefaultButtonOffset: float. How far the button sticks out on its own.
* RegisteredPressDistance: float. How far the button must be pushed in to register a press.
* CurrentPressDistance: float. How far the button is currently pressed.
* LastPressDistance: float. How far the button was pressed in the last frame.
* ButtonLogicMode: EButtonLogicMode. Whether the button toggles states or fires and forgets upon being pressed.

Methods:

* Pressed(): void. Dispatches event to listening actors when the button is pressed, independent of the ButtonLogicMode.
* Toggled(): bool Active. Dispatches whether the button is active when it is pressed if the ButtonLogicMode is ‘toggle’.
* StartTrackingPressDistance(): void. Begin a button interaction.
* GetPressDistance(): float. Calculate the current press distance.
* StopTrackingPressDistance(): void. End a button interaction.

6.3.1.8 SwitchInteractivityComponent Class: Class for interactive objects that can be switched on and off.

Fields:

* SwitchMesh: UStaticMesh\*. Mesh for part that moves.
* SwitchHousingMesh: UStaticMesh\*. Mesh for part that stays stationary.
* AngleRange: float. How wide the switch can rotate.
* SwitchLength: float. How long the switch is.
* DefaultActivation: bool. Whether the switch starts facing up or down.
* CurrentFingerAngle: float. What angle the motion controller has currently flipped the switch interactivity component to.

Methods:

* Toggled(): bool Active. Dispatch whether or not the switch is active upon passing through zero degrees (the middle of the switch range).
* StartTrackingTargetAngle(): void. Begin a switch interaction.
* GetTargetAngle(): float. Calculate what angle the motion controller is flipping the switch to.
* StopTrackingTargetAngle(): void. End a switch interaction.

6.3.2 Detailed Interface Descriptions

The following sections provide details about all interfaces between classes occurring in the VR Interactivity Pack.

6.3.2.1 HandComponent to PickupInteractivityComponent Interface: The HandComponent gets the SurfaceShape and SymmetryAxis from the PickupInteractivityComponent in order to determine the correct hand pose. The HandComponent attaches its hand mesh to the PickupInteractivityComponent in order to visually appear attached even when the PickupInteractivityComponent is offset from the HandComponent.

6.3.2.2 HandComponent to TwistInteractivityComponent Interface: The HandComponent attaches its hand mesh to the TwistInteractivityComponent in order to visually appear attached even when the TwistInteractivityComponent is offset from the HandComponent. The HandComponent sends the TwistInteractivityComponent its Y-axis rotation in order to rotate the TwistInteractivityComponent.

6.3.2.3 HandComponent to LeverInteractivityComponent Interface: The HandComponent attaches its hand mesh to the LeverInteractivityComponent in order to visually appear attached even when the LeverInteractivityComponent grab region is offset from the HandComponent. The LeverInteractivityComponent uses transforms and trigonometry to determine what angle the HandComponent is moving the LeverInteractivityComponent to.

6.3.2.4 HandComponent to ButtonInteractivityComponent Interface: When the HandComponent enters the detection area of the ButtonInteractivityComponent, the ButtonInteractivityComponent will use a cylindrical raycast every frame to determine how far the HandComponent has pushed.

6.3.2.5 HandComponent to SwitchInteractivityComponent Interface: The SwitchInteractivityComponent uses transforms and trigonometry to determine what angle the HandComponent is flipping the SwitchInteractivityComponent to.

6.3.3 Detailed Data Structure Descriptions

The following sections provide details about all data structures used in the VR Interactivity Pack.

6.3.3.1 ESurfaceShape: Defines preset surface shapes. For each type of surface shape, the hand actor will animate to fit the approximated shape. This avoids having to make any complex calculations to grab realistically.

* SS\_Flat
* SS\_Sphere\_Large
* SS\_Sphere\_Medium
* SS\_Sphere\_Small
* SS\_Cylinder\_Large
* SS\_Cylinder\_Medium
* SS\_Cylinder\_Small
* SS\_Bow
* SS\_Gun
* SS\_Handle

6.3.3.2 ESymmetryAxis: Defines how the left hand will orient itself when grabbing in relation to how the right hand orients itself. Such can be captured in the axis of symmetry of a grab region.

* SA\_X
* SA\_Y
* SA\_Z

6.3.3.3 EButtonLogicMode: Defines whether a button will act with toggle logic or fire and forget logic.

* BLM\_Toggle
* BLM\_FireAndForget

6.3.3.4 EVisibility: Defines where the interactivity component is visible.

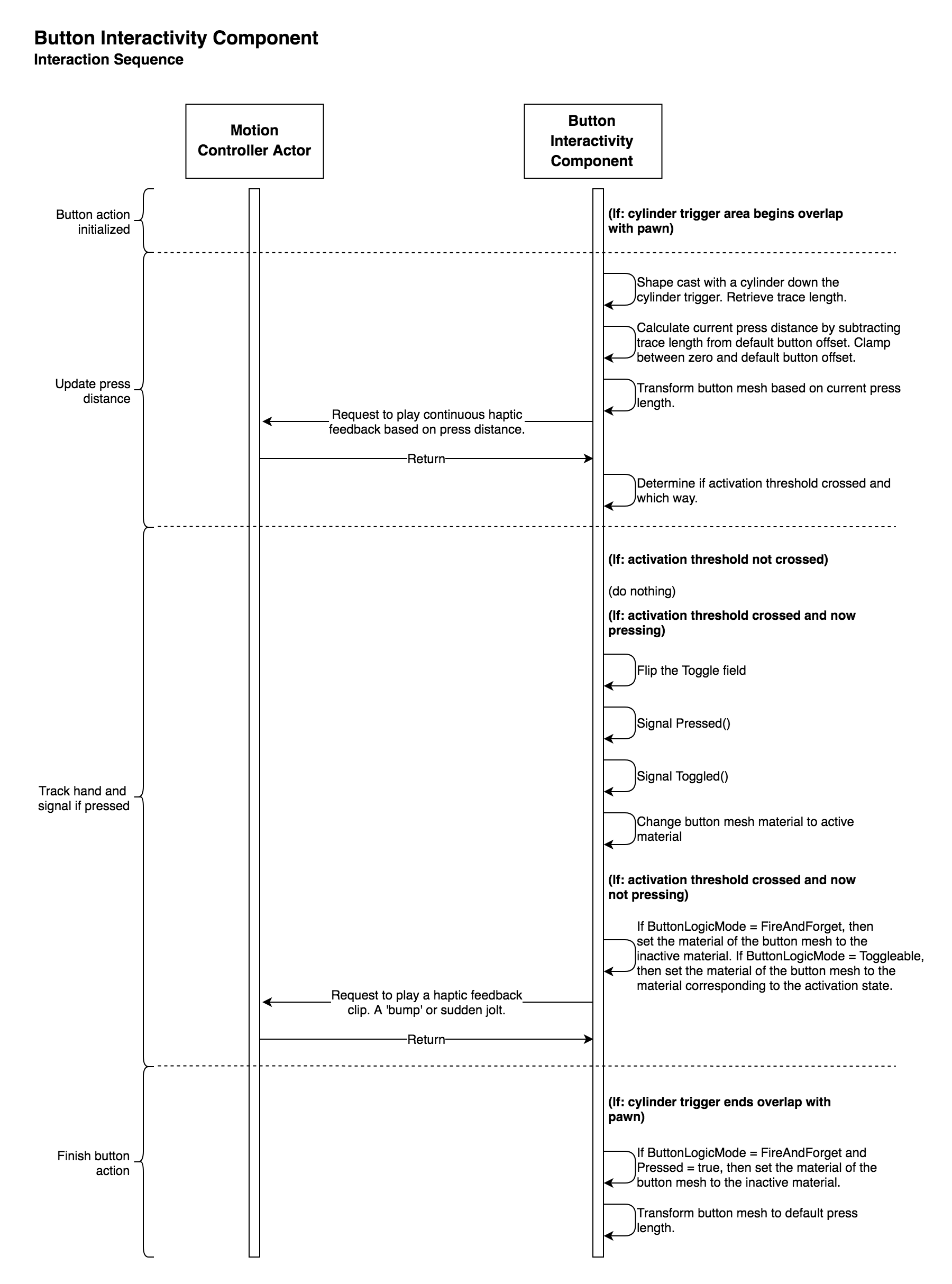
* V\_VisibleInGame
* V\_VisibleInEditor
* V\_Invisible

6.3.3.5 EInteractivityMode: Defines if an interactivity component is disabled and how.

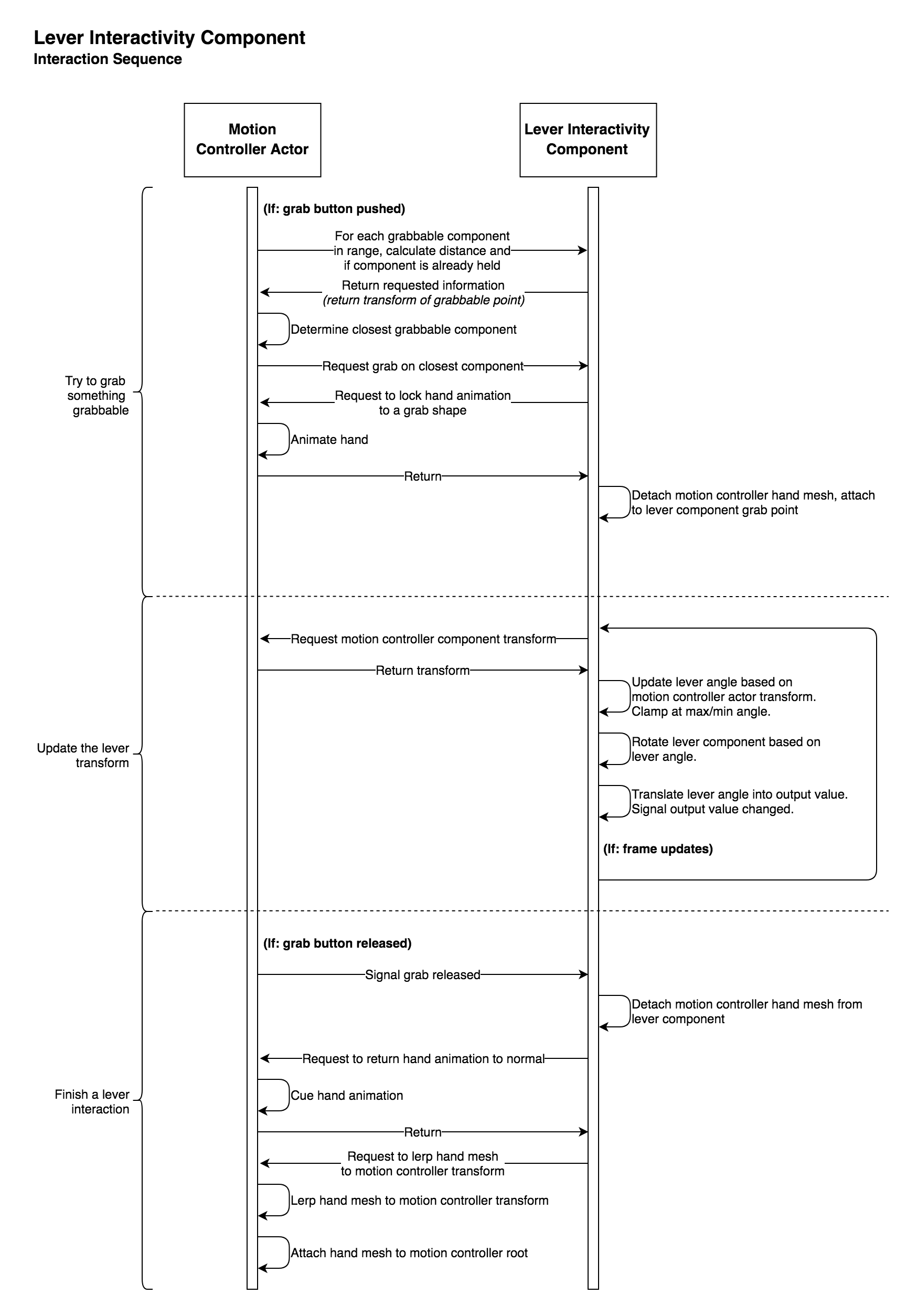
* IM\_Enabled
* IM\_NoEvents (Component will move, but will not dispatch any information to listening actors.)
* IM\_Static (Component will not move and will not dispatch any information.)

6.3.4 Detailed Design Diagrams

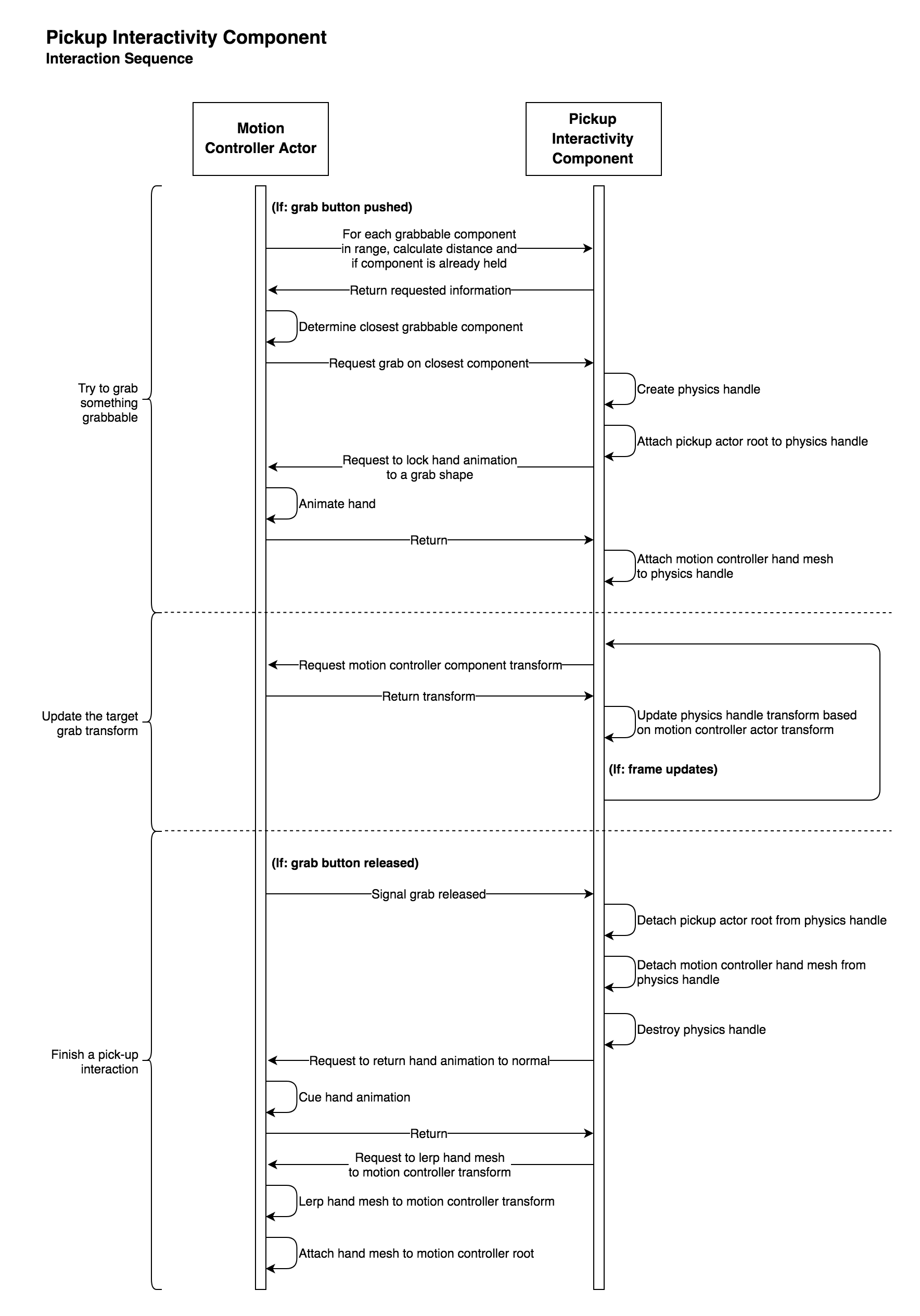
Button Sequence Diagram:



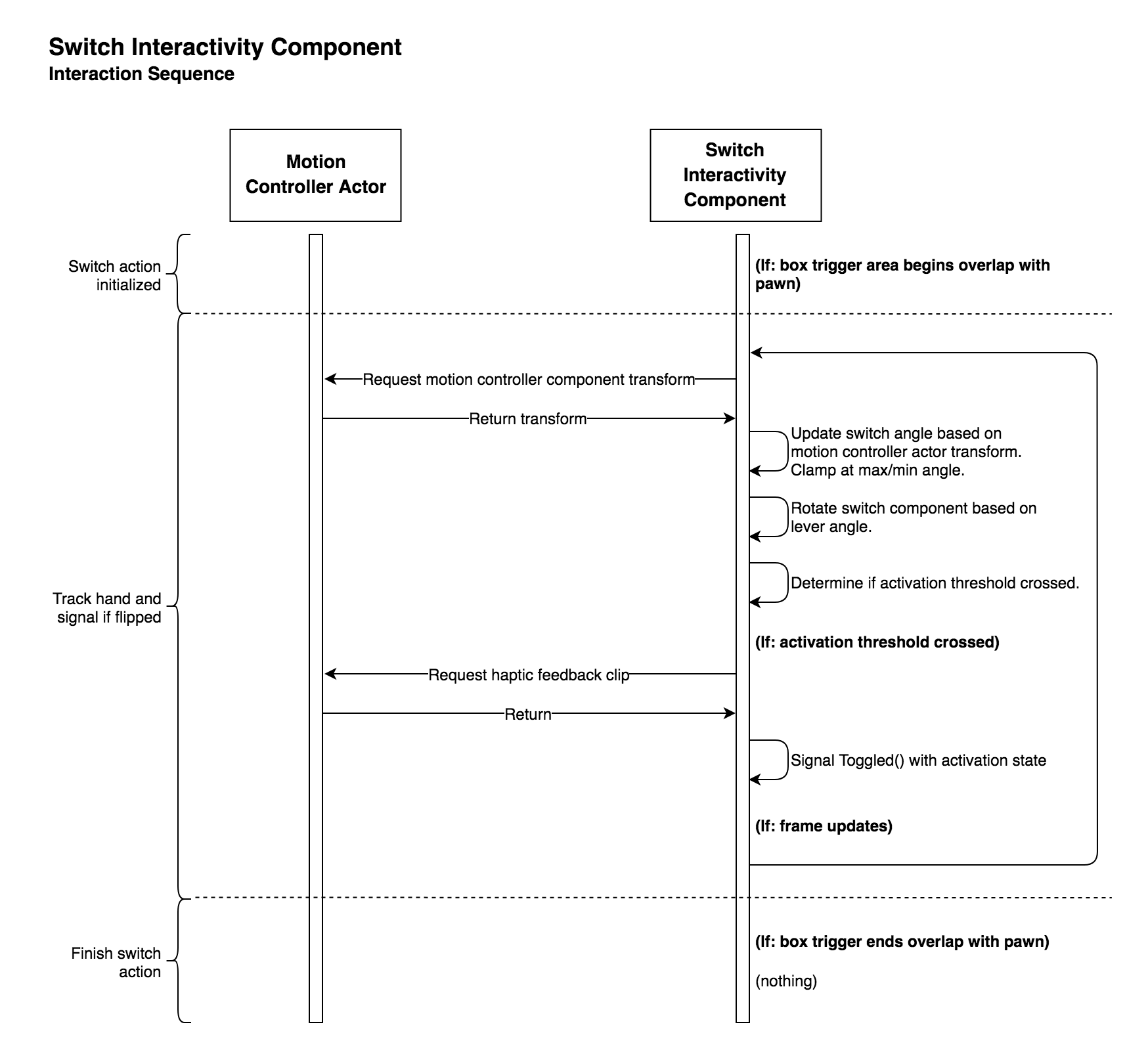
Lever Sequence Diagram:



Pickup Sequence Diagram:



Switch Sequence Diagram:



Twist Sequence Diagram:

