GyvrEngine – Technical Document

# ENGINE COMPOSITION

The GyvrEngine is composed of multiple main projects. These projects are:

* GyvrAPI: Contains interfaces definitions
* GyvrCore: Mediator of the engine. This is what the user is more likely to manipulate
* GyvrRendering: Handle models, texts, sprites rendering
* GyvrPhysics: Apply physics to game entities
* GyvrAudio: 3D audio system
* GyvrMaths: Contains implementation of mathematic objects

# DLL COMMUNICATION

## Needs

To have a modular game engine, it is necessary to split the project into smaller projects as seen before. Every project is a unique DLL that is included in another project, except for the GyvrCore that will be included in the user project.

## Problem

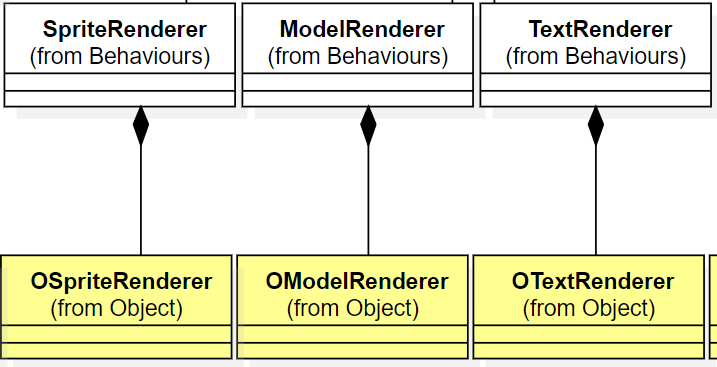
The problem here is to find a way to communicate with DLLs, knowing that if a DLL uses another DLL, the communication (Methods calls for instance) is one way.

## Solution

We will take the example of GyvrRendering that is used in GyvrCore.

In GyvrCore we have some GameObjects that can have components, like a ModelRenderer. We want to say to GyvrRendering: *“Can you render me this ModelRenderer?”*.

The problem here is that a ModelRenderer is a component, so it is specific to GyvrCore and can’t be sent to GyvrRendering. To get around this problem, we created some objects (Non-part of an entity-component architecture) that are wrapped in our components (OModelRenderer, OSpriteRenderer, …). This way, we have the ModelRenderer component (From GyvrCore) that wraps the OModelRenderer (From GyvrRendering), so we can say to GyvrRendering: *“Can you render me this OModelRenderer?”*



Rendering-based components wraps rendering-specific objects

# TODO …