Proyecto Middleware / Prog. Gráfica

Arquitectura del proyecto

ESNE, CURSO 4.3 18/19

Santiago Arribas Maroto

Mayo 2019

1. Introducción a la herramienta

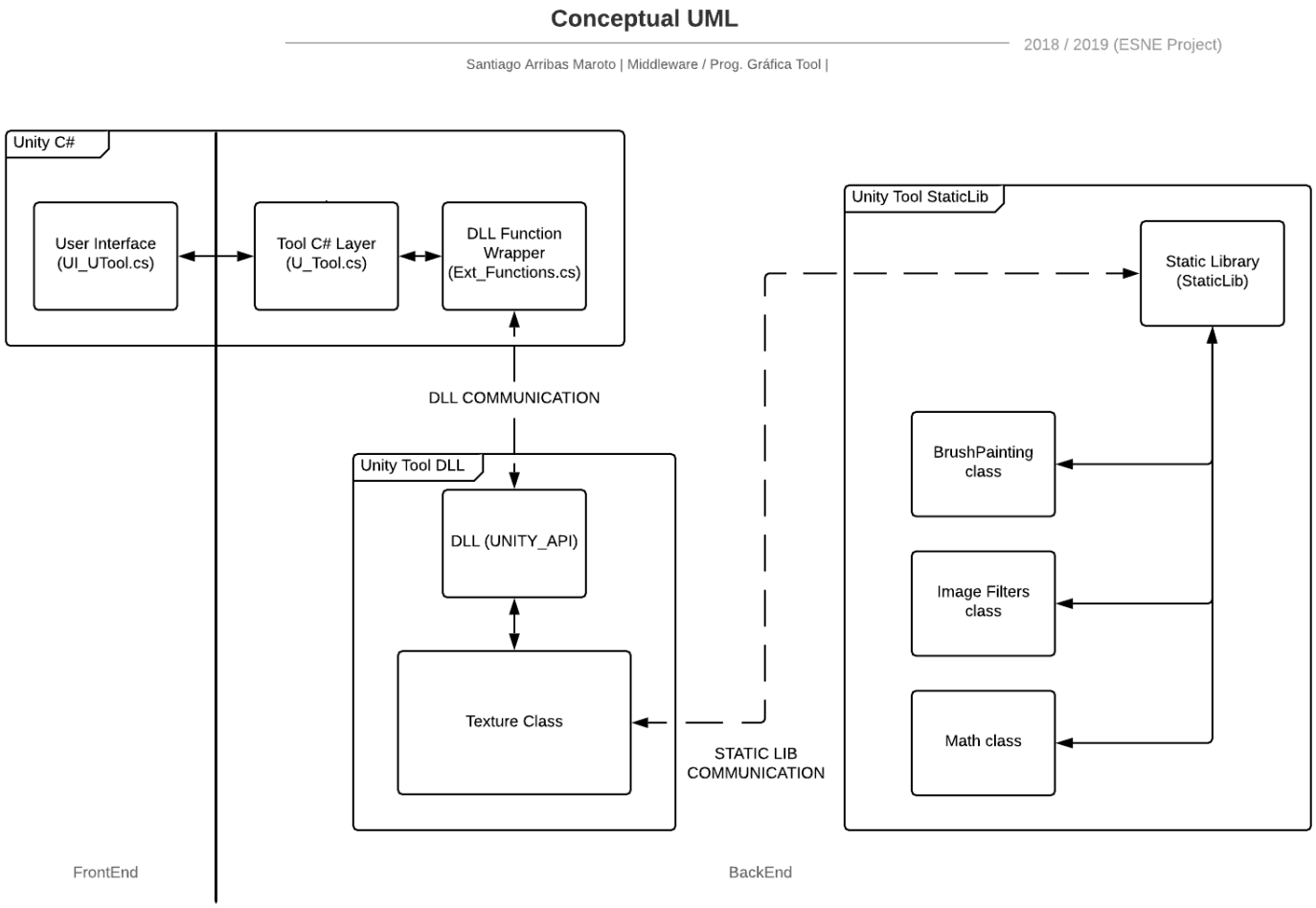

Imagen 1 –Conceptual Data Model - Véase documentación adicional para ver la ampliación

La herramienta creada dispone al usuario las capacidades para editar una textura directamente desde la herramienta Unity3D, sin necesidad de software externo, dicha edición permite, tanto **cambiar el nombre, el tamaño como los bytes de la misma**, la edición de bytes permite aplicar **filtros al conjunto** de bytes de la imagen (de tipo Sepia / Blanco y Negro), o edición específica de dibujado en la posición en la cual se sitúe el cursor (con la **herramienta pincel** (Botón Izquierdo del ratón)) así como borrado de los cambios (con la **herramienta borrador**(Botón derecho del ratón)).

Además, existen dos configuraciones finales, un botón que permite **revertir todos los cambios** a su estado original, así como un botón que permite **guardar la imagen**. Al guardar la imagen se establece una nueva imagen en formato PNG, con el tamaño especificado y el conjunto de bytes que el usuario haya modificado.

Esta herramienta dispone de 3 partes diferenciadas, la **parte gráfica**, programada en Unity3D, como clase de tipo Editor, la parte del ***BackEnd* situada en Unity (C#),** que posee la información de la textura y permite operar con la misma, y la parte del ***BackEnd* en proyectos externos (DLL y *StaticLib* (C++)).**

2. FrontEnd: Interfaz de usuario

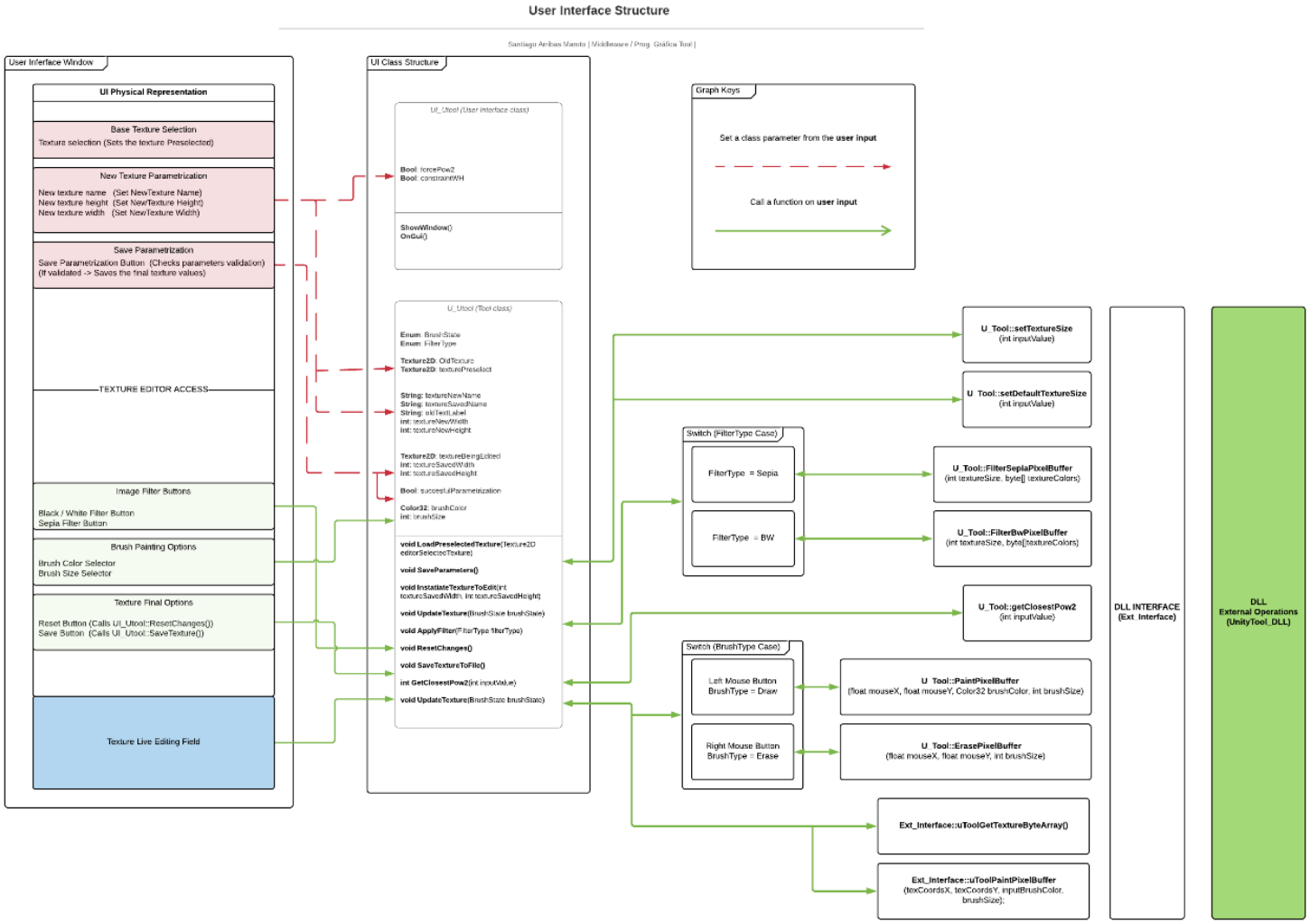

Imagen – User Interface Structure – Véase documentación adicional para ver la ampliación

La interfaz de usuario está implementada dentro de la herramienta Unity, se puede acceder a la misma a través del acceso directo en la barra de navegación ("*Window -> UTool*"), dicha interfaz hereda de la clase **Unity "Editor"**, lo cual permite que la misma se ejecute al tiempo que el editor está en activo, sin necesidad de iniciar la ejecución de la escena en el viewport.

Esta interfaz simplemente se encarga de **obtener los valores que el usuario introduce** para la configuración de su nueva textura, así como la **llamada de funciones** mediante la interacción con los botones o el editado de la textura mediante la ventana de edición de textura con pincel o borrador.

La interfaz posee un acceso a capacidades de forma secuencial, solamente se podrá acceder a los parámetros de edición sobre bytes (Filtros / Pintado) si los parámetros introducidos para la nueva textura son **válidos** (Véase Guía de Uso para ver parámetros válidos).

3. BackEnd: Capa C# (Unity 3D)

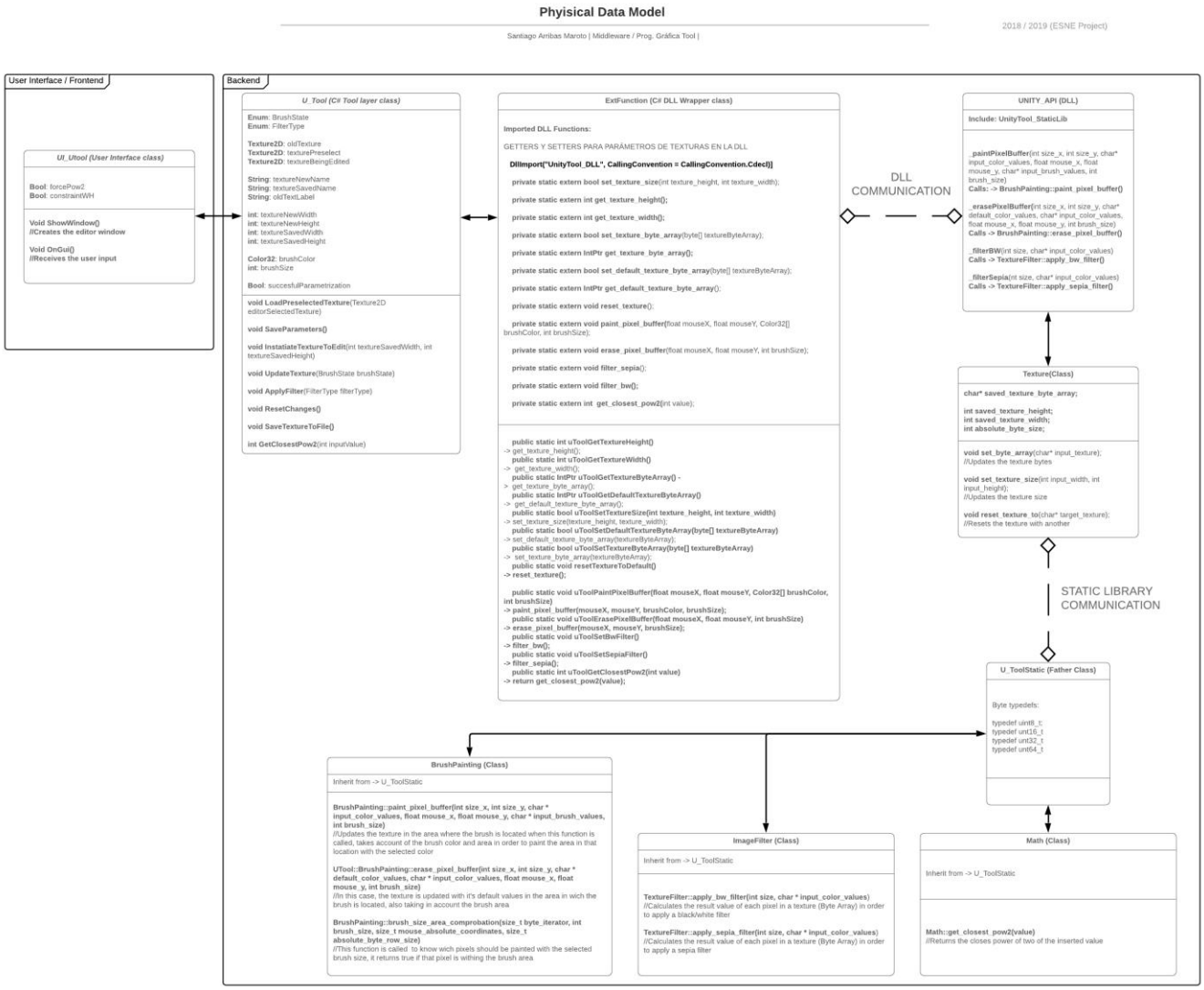

Imagen – Physical Data Model – Véase documentación adicional para ver la ampliación

Como ya comenté, la herramienta posee una parte del *BackEnd* en c#, uno de los elementos principales de esta parte, es la clase "***UTool***" dicha clase permite **mostrar y actualizar la información que se ve en el editor**, pero principalmente está pensada para **albergar la lógica de funcionamiento** en la comunicación con la parte externa de la herramienta y las acciones del usuario con la interfaz. Otro elemento destacado de esta parte es el ***wrapper* para las funciones importadas de la DLL**, llamado "***ExtInterface***", la cual encapsula funciones C++ para que el uso de las mismas sea más simple y coherente dentro del uso de la herramienta.

Se ha tratado de limitar al máximo las operaciones realizadas en la parte de C#, aun así, una operación ha debido quedarse en esta parte, dicha operación es *SetTextureRawData()*, esta operación carga un array de bytes a una variable de tipo "*Unity::Texture2D*". Esta función se usa para actualizar la textura que se ve en la parte de edición de textura con pincel del editor. Podrían haberse importado capacidades en la DLL para operaciones con variables propias de Unity, pero se decidió mantenerlo separado para mantener las capacidades de la DLL desvinculada de Unity por **reusabilidad de la misma en otros proyectos** (con o sin Unity), ya que solo usa tipos básicos.

4. BackEnd: Capa C++ (Dynamic Link Library + Static Library)

La parte de la herramienta localizada en la **DLL** vinculada al proyecto, posee tanto las capacidades de realizar las **operaciones con texturas**, así como la info**rmación fundamental de las mismas** para permitir dichas operaciones. En esta DLL existe **la clase** "***Texture***" esta clase guarda tanto el tamaño de la misma, como los bytes que la conforman, además la propia clase "*Texture*" posee funciones para **actualizar, cargar y resetear la información que posee.**

Más allá de la DLL, existe una **Librería Estática** vinculada, esta librería estática posee **funciones estáticas de cálculo de bytes fundamentalmente**, posee tres clases diferentes, "***Filter***", "***BrushPainting***" **y** "***Math***".

"***Filter***" permite la **actualización completa de la textura aplicando un filtro de imagen** sobre la misma, todos los píxeles de la imagen reciben el mismo tratamiento, resultando en una transformación homogénea de la imagen.

"***BrushPainting***" permite una **transformación localizada, ya que recibe la posición de ratón del usuario**, calcula el offset de la posición del mismo sobre la textura y modifica esta solamente en la posición indicada con el radio indicado, además permite el borrado con borrador, el cual funciona similar al pincel, salvo que instancia los píxeles en la posición y área del borrador con los píxeles de la textura original.

Por último existe la clase "***Math***", la cual solamente posee un método para devolver la potencia de 2 más cercana al valor introducido, esta clase es fácilmente ampliable y está pensada para albergar algunas de las operaciones más pesadas en futuras implementaciones

5. Notas sobre la optimización

A pesar de esto, soy consciente de que mayor optimización seguro que existe, y no solo a nivel de velocidad de transferencia de información, si no a nivel de lógica de ejecución, tal como el completado de información entre posiciones de ratón, para evitar los "Puntos" que se generan al mover rápidamente el ratón, mediante un **vector que una ambos puntos**, **completando con color el vector director entre ambos.** Por otro lado también he investigado el funcionamiento de la herramienta Photoshop, y la misma no aplica los cambios directamente sobre la textura con cada modificación, si no que el **sistema de capas** que usa, le permite crear una **representación visual de la textura editada con los cambios del usuario en una capa virtual no accesible superior**, esta capa es muy ligera en resolución, lo que permite modificaciones instantáneas, y solamente al pulsar el botón de guardado o de aplicación de cambios es cuando se lee la información de dicha capa y se sobrescribe sobre la imagen original.

Tanto el cálculo del vector director entre puntos como la existencia de "Capas" en la herramienta son opciones que **podrían ser implementadas a futuro**, para mejorar la herramienta en sus capacidades, a pesar de eso, teniendo en cuenta el tiempo disponible actualmente, y a sabiendas de que son **"Ampliaciones Funcionales" sobre la aplicación y nada vital para el funcionamiento de la misma,** he decidido centrarme en otras áreas de la herramienta y tildar estas funcionalidades como interesantes pero secundarias en el contexto de esta entrega.