UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Documento de arquitectura y diseño

Informe presentado por

Dara Leslie Silvera Martínez y Nicolás Cámera López

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Federico Gómez Frois
Sylvia Rita da Rosa Zipitría

Montevideo, 30 de julio de 2023

Índice

1. Introducción

1.1. Propósito

El presente documento tiene la finalidad de definir la arquitectura de la aplicación a desarrollar y especificar el diseño de los casos de uso a implementar.

1.2. Alcance

Este documento presentará la arquitectura del sistema, así como un detalle de los módulos que la componen. Mediante diagramas de secuencia, se mostrarán los flujos de información entre los distintos componentes y actores.

1.3. Referencias

Se recomienda leer previamente el Documento de Requerimientos y Casos de Uso, para entender mejor los diagramas de secuencia, así como las decisiones tomadas para extender la arquitectura de MateFun.

2. Arquitectura de MateFun

La arquitectura base de la aplicación MateFun seguirá siendo la misma que tenía previo al inicio de este proyecto de grado, ya que el objetivo del mismo es incorporar nuevas funcionalidades, construidas sobre la arquitectura ya existente. La integración de los nuevos casos de uso se realizará modificando los distintos módulos de la arquitectura existente.

En las siguientes secciones se detallarán las distintas vistas de la arquitectura de MateFun.

2.1. Vista lógica

En la Figura 2.1.1.1 se puede apreciar el diagrama de capas de la aplicación MateFun junto con las tecnologías a utilizar para la implementación de cada una.

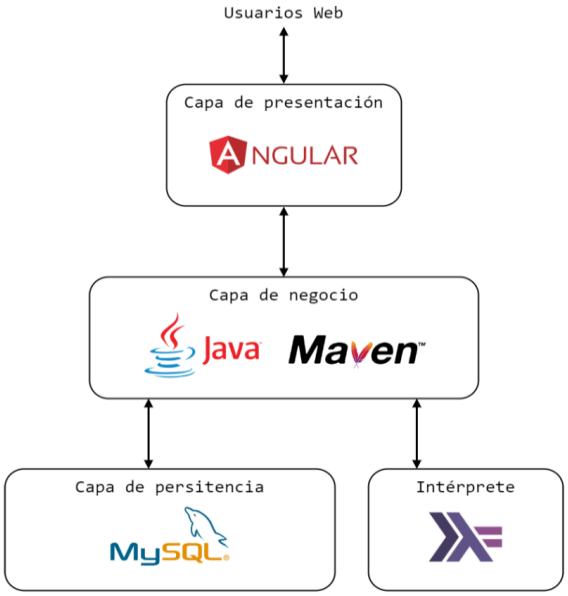


Figura 2.1.1.1: Diagrama de capas.

Presentación: La capa de presentación se refiere a la interfaz gráfica. Es la encargada de la interacción con el usuario y del despliegue de los elementos visuales del sistema.

Negocio: Refiere a la lógica de negocio del sistema. Aquí se encuentra la implementación de las funciones que resuelven las funcionalidades que el usuario ejecuta desde la interfaz. Además, esta capa se encarga de comunicar la capa de presentación con el intérprete de MateFun y con la capa de persistencia.

Persistencia: Es la capa encargada de la gestión de la base de datos del sistema. Provee una API que permite a la capa de negocios realizar operaciones sobre la base de datos.

Intérprete: Consiste en un ejecutable binario capaz de compilar y ejecutar en modo interactivo código Matefun. Almacena archivos temporales en el sistema de archivos del servidor. [1]

2.2. Vista de distribución

En la Figura 2.2.1.1 se puede apreciar el diagrama de distribución de la aplicación MateFun. En el mismo, desde el punto de vista físico se identifican tres nodos: Navegador (browser), Servidor de aplicaciones (Application Server) y Servidor de base de datos (DataBase System).

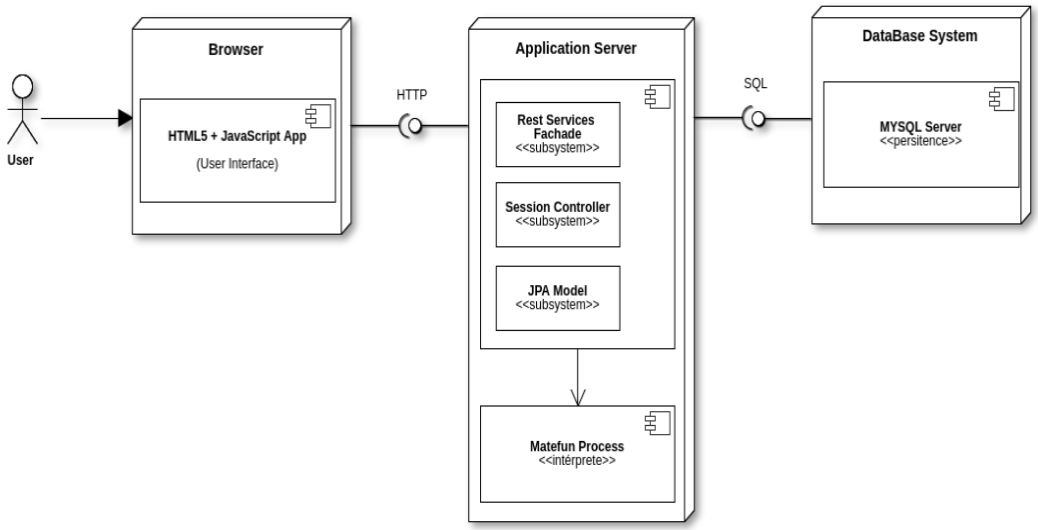


Figura 2.2.1.1: Diagrama de distribución [1].

Browser: Representa el entorno de ejecución del usuario final. El sistema es accesible desde un navegador web que tenga la función de JavaScript activa.

Application Server: Este nodo constituye el servidor de aplicaciones. Aquí se aloja la aplicación JAVA que da soporte al *Backend* del sistema.
El *Backend* de MateFun está compuesto principalmente por los siguientes subcomponentes:

- Rest services fachade: Fachada de web services REST para la comunicación mediante HTTP entre el *Frontend* y el *Backend*.
- Session Controller: Contiene la lógica necesaria para mantener la sesión de los usuarios activos.
- JPA Model: Mantiene el modelo de persistencia.

DataBase System: Contiene el servidor de bases de datos para la persistencia del sistema. [1]

2.3. Subsistemas

En la Figura 2.3.1 se puede apreciar el diagrama de subsistemas y sus dependencias.

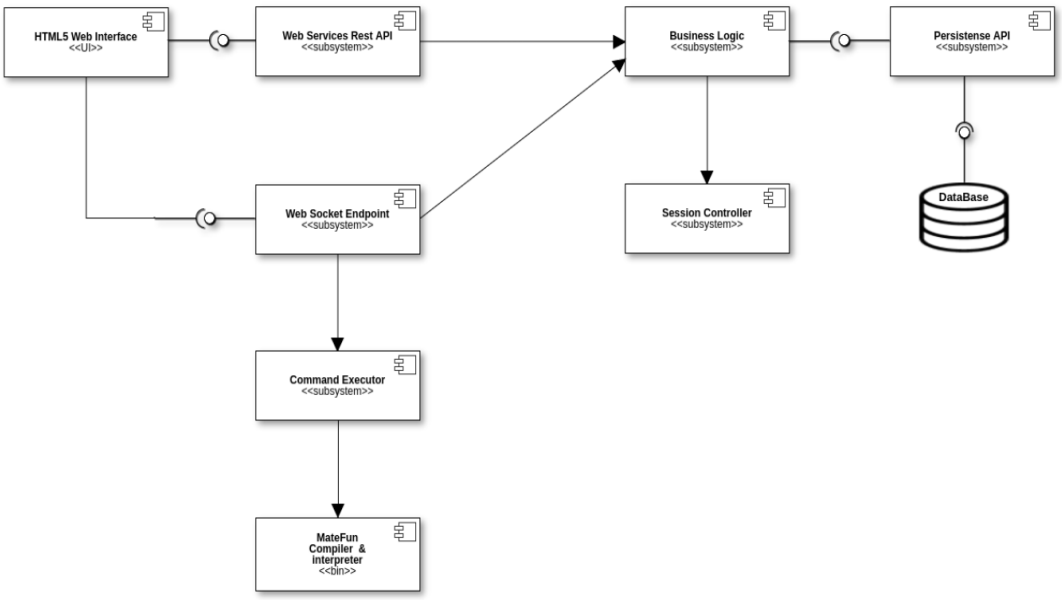


Figura 2.3.1: Diagrama de subsistemas y dependencias. [1]

HTML5 Interfaz Web: En una aplicación Web basada en el framework de JavaScript Angular[2], en conjunto al framework de CSS Bootstrap[3]. También utiliza bibliotecas como JQConsole[4] y CodeMirror[5] para la implementación del intérprete y el editor de texto respectivamente. Este componente corresponde a la capa de presentación. Presenta el sistema al usuario y es responsable de las interacciones de éste con el sistema.

Web Services Rest API. Este subsistema constituye una fachada de servicios web a través de los cuales la capa de presentación se comunica con el servidor. Los servicios están construidos utilizando el estándar REST. Para ello se utilizó la biblioteca JAX RS[6].

WebSocket Service. Este subsistema es el encargado de exponer un endpoint de WebSocket a través del cual la capa de presentación invoca los comandos escritos por el usuario en el intérprete interactivo. Como resultado de la llamada a los distintos servicios REST, se devuelven mensajes en formato JSON con los

resultados de la invocación de los comandos. Estos resultados luego son dibujados por el módulo del graficador correspondiente.

Comand Executor. Este módulo tiene la responsabilidad de ejecutar los comandos que son recibidos desde la capa de presentación (a través del WebSocket). Por cada usuario conectado este módulo mantiene dos hilos que leen la salida estándar y la salida de error del proceso que ejecuta el binario MateFun.

Intérprete MateFun. Corresponde al intérprete de MateFun, es un binario ejecutable que recibe como entrada un programa MateFun (fichero con extensión `.mf`) y lo compila y ejecuta en modo interactivo.

Business Logic. Aquí se encuentra la lógica de negocio encargada de realizar el procesamiento necesario antes de persistir datos o de enviar datos de respuesta ante la invocación de un servicio web.

Session Controller. Este módulo se encarga de mantener una sesión virtual para los usuarios que acceden al sistema en modo invitado.

Persistense API. Interfaz de persistencia, construida utilizando el Framework JPA[8].

2.4. Módulos a extender para implementar los casos de uso

En la Figura 2.4.1 se pueden observar los módulos señalados en dónde se van a extender las funcionalidades para implementar los casos de uso nuevos a integrar en la aplicación MateFun.
Toda la nueva lógica de comandos a agregar va a estar ubicada en el módulo "MateFun Proceso", mientras que toda la nueva lógica de dibujado 3D va a estar principalmente ubicada en módulo "Graph3D" con algunos ajustes en el Módulo "3D".

En los diagramas de secuencia se abordarán más detalles de cómo se implementan los distintos casos de uso.

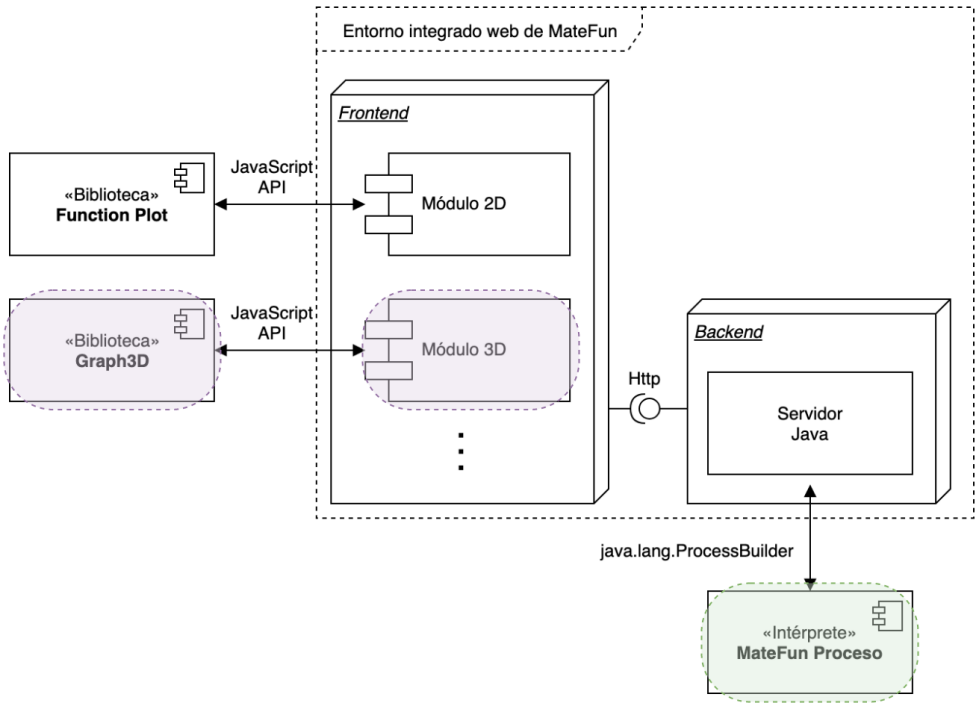


Figura 2.4.1: Arquitectura resumida de MateFun, destacando módulos importantes. [7]

Siguiendo con la propuesta de diseño y de módulos de la Arquitectura de la biblioteca "Graph3D" propuesta en [7], las nuevas primitivas de figuras 3D serán agregadas al módulo "Figuras" (ver Figura 2.4.2), así como cualquier función auxiliar para implementar las mismas. En adición a esto, será necesario modificar los módulos "Escena" y el "Render" para permitir que las nuevas figuras puedan ser dibujadas, así como para permitir adaptar la nueva meta información que será necesaria para llevar los nuevos casos de uso, al igual que para adaptar los casos de uso ya existentes para extenderlos con las nuevas funcionalidades.

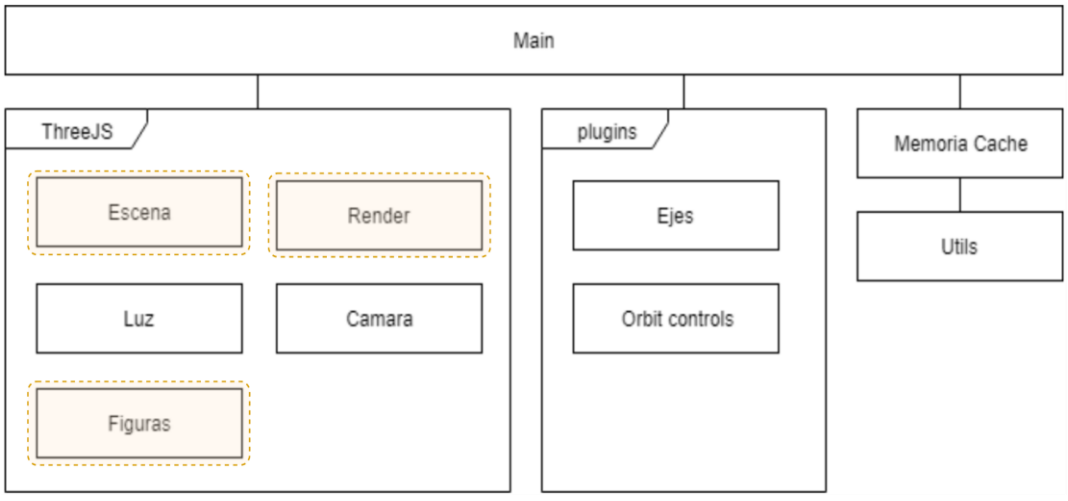


Figura 2.4.2: Arquitectura de la biblioteca "Graph3D".

Por último, desde el punto de vista del módulo "MateFun Proceso", continuando con la propuesta de diseño y módulos propuesta en [9], en el módulo "Core" se agregan las nuevas primitivas así como su respectivo tipado; en el módulo "Figures" se encontrará la implementación de las nuevas primitivas, funciones auxiliares y tipos de datos necesarios; en el módulo "ReservedNames" se encontrará los nombres reservados para las funciones a agregar en el intérprete, y finalmente en el módulo "InternationalizationHelper" se encontrarán las traducciones de las nuevas primitivas y de los mensajes de errores de las mismas.

La arquitectura del módulo "MateFun Proceso" se puede apreciar en la Figura 2.4.3. Cada círculo corresponde a un módulo implementado en Haskell y cada flecha indica la dependencia entre pares de módulos.

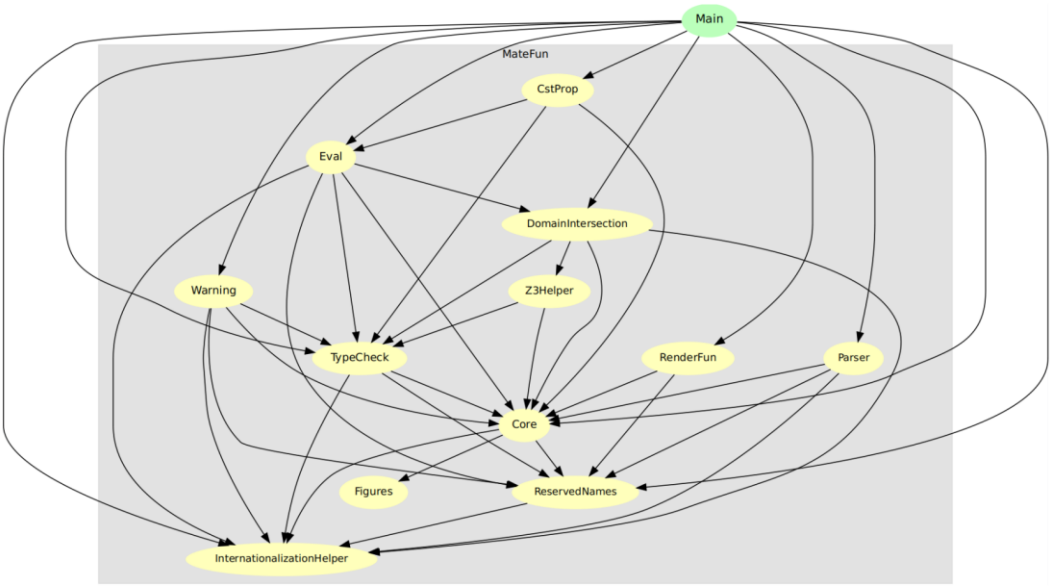


Figura 2.4.3: Arquitectura y grafo de dependencias de los módulos de "MateFun Proceso".

3. Diagramas de secuencia

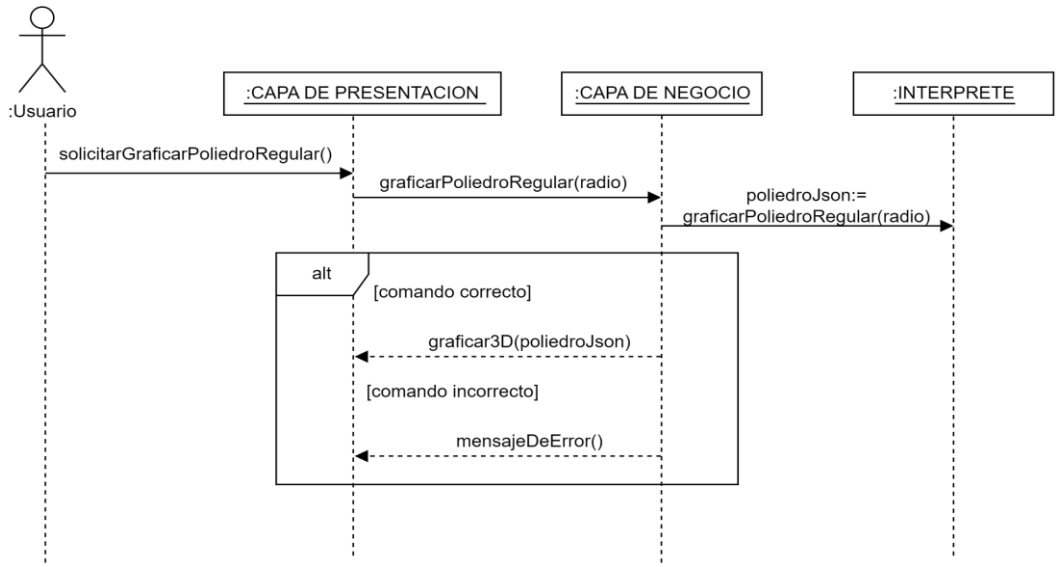
En esta sección se encuentran los diagramas de secuencia asociados a los casos de uso definidos en el documento de Requerimientos y Casos de Uso.

3.1. Graficar poliedro regular

3.1.1. Descripción

El caso de uso comienza cuando un usuario quiere graficar un poliedro regular. Para ello, el usuario ingresa alguno de los comandos (o el comando) para graficar un poliedro regular en el intérprete, y el sistema dibuja en el graficador 3D el resultado de la ejecución de la función, o se devuelve un error en el intérprete en caso de que se haya instanciado mal el comando.

3.1.2. Diagrama de secuencia



3.1.3. Diseño en MateFun

El anterior diagrama de secuencia muestra los flujos existentes para cualquiera de las cuatro posibles funciones para graficar un poliedro regular (`graficarPoliedroRegular` en el diagrama de secuencia). Estas nuevas funciones para graficar poliedros regulares se definen de la siguiente forma:

- **`tetraedro :: R -> Fig3D`**

  Dado el radio, gráfica un poliedro regular tetraedro.

- **`octaedro :: R -> Fig3D`**

  Dado el radio, gráfica un poliedro regular octaedro.

- **`dodecaedro :: R -> Fig3D`**
  Dado el radio, gráfica un poliedro regular dodecaedro.

- **`icosaedro :: R -> Fig3D`**
  Dado el radio, gráfica un poliedro regular icosaedro.

En inglés la funciones quedarían como **`tetrahedron`**, **`octahedron`**, **`dodecahedron`** e **`icosahedron`**, respectivamente.

Casos bordes.

- Las funciones admiten cualquier valor para el radio. Luego, el módulo "Graph3D" se encarga de ajustar el radio que le llega en el correspondiente JSON para graficar un poliedro.
  - Radio menor o igual a cero, será lo mismo que cero.
  - Radio positivo no sufre modificación alguna.

Para implementar los JSONs asociados a estas nuevas funciones, se agregan 4 nuevos tipos (**`kind`**) de figuras: **`tetrahedron`**, **`octahedron`**, **`dodecahedron`** e **`icosahedron`**. Cada uno representa su respectivo poliedro, y en conjunto a esto, cada poliedro tiene asociado su radio, **`r`**, como parte de su meta información. A modo de ejemplo, a continuación se muestra el JSON asociado a la función **`tetraedro(3)`**

```
FIG3D: [
  {
    "kind": "tetrahedron",
    "r": 3.0,
    "x": 0.0,
    "y": 0.0,
    "z": 0.0,
    "color": "white",
    "rot": (0.0, 0.0, 0.0),
    "transparency": 0.0
  }
]
```

La misma idea aplica para el resto de los tipos (**`kind`**) asociados a los nuevos poliedros regulares; el campo **`r`** se mantiene para todos los poliedros regulares. Destacamos que, el campo **`transparency`** cobrará más sentido en la sección 3.5.3.

Adicionalmente, si este caso de uso se lo compone con otros casos de usos existentes en MateFun, para mover y colorear una figura, usando la **`mover3D(color3D(tetraedro(3), Rojo), (2,4,8))`** el siguiente sería el JSON asociado.

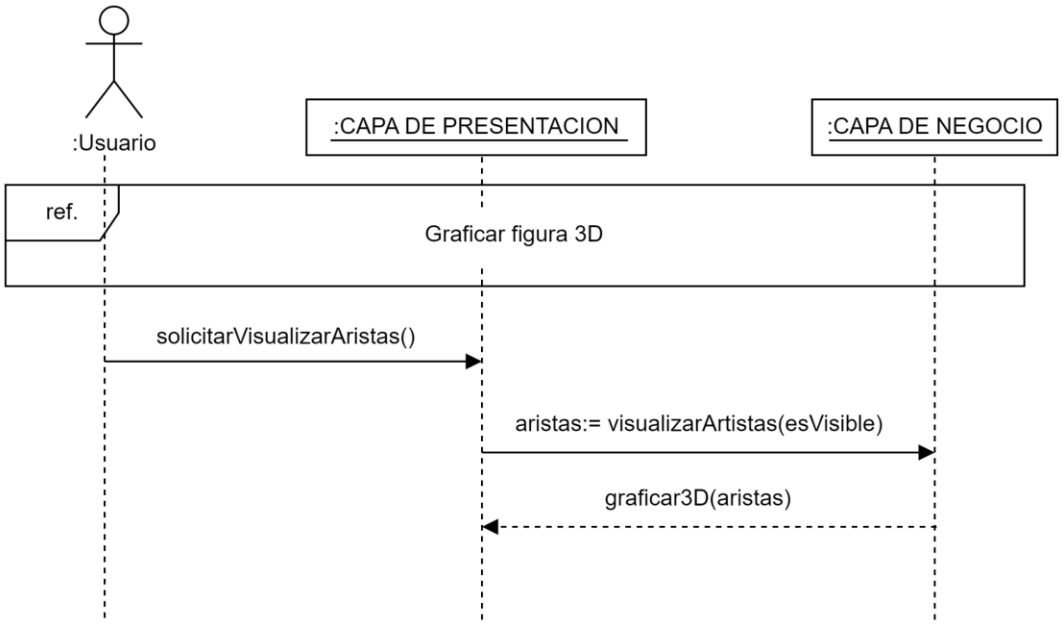
```
FIG3D: [
  {
    "kind": "tetrahedron",
    "r": 3.0,
    "x": 2.0,
    "y": 4.0,
    "z": 8.0,
    "color": "red",
    "rot": (0.0, 0.0, 0.0),
    "transparency": 0.0
  }
]
```

3.2. Visualizar aristas de figuras 3D

3.2.1. Descripción

El caso de uso comienza cuando un usuario quiere visualizar las aristas de las figuras 3D del panel de gráficos 3D. Para ello, el usuario despliega el menú de configuración, clickea el CheckBox cuya leyenda es "Aristas", y el sistema dibuja en el graficador 3D el resultado de la nueva configuración.

3.2.2. Diagrama de secuencia



3.2.3. Diseño en MateFun

Como se muestra en el diagrama de secuencia, este caso de uso no cuenta con la participación del intérprete, ya que no se implementa el caso de uso mediante alguna función del intérprete. Toda la interacción para efectuar el caso de uso de esta sección estará disponible como un nuevo CheckBox -con leyenda "Aristas"- en el menú de configuración del graficador 3D.

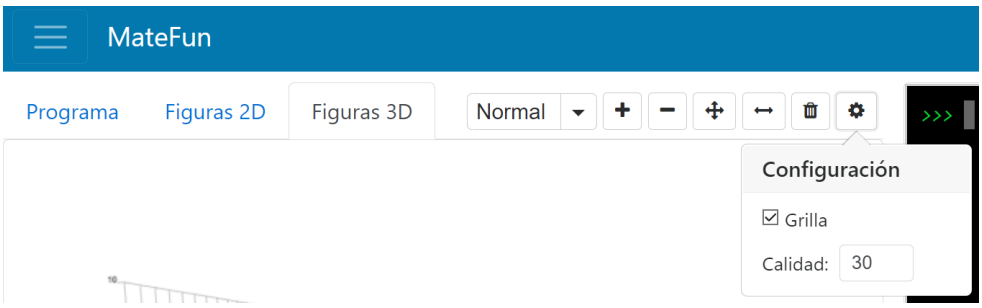


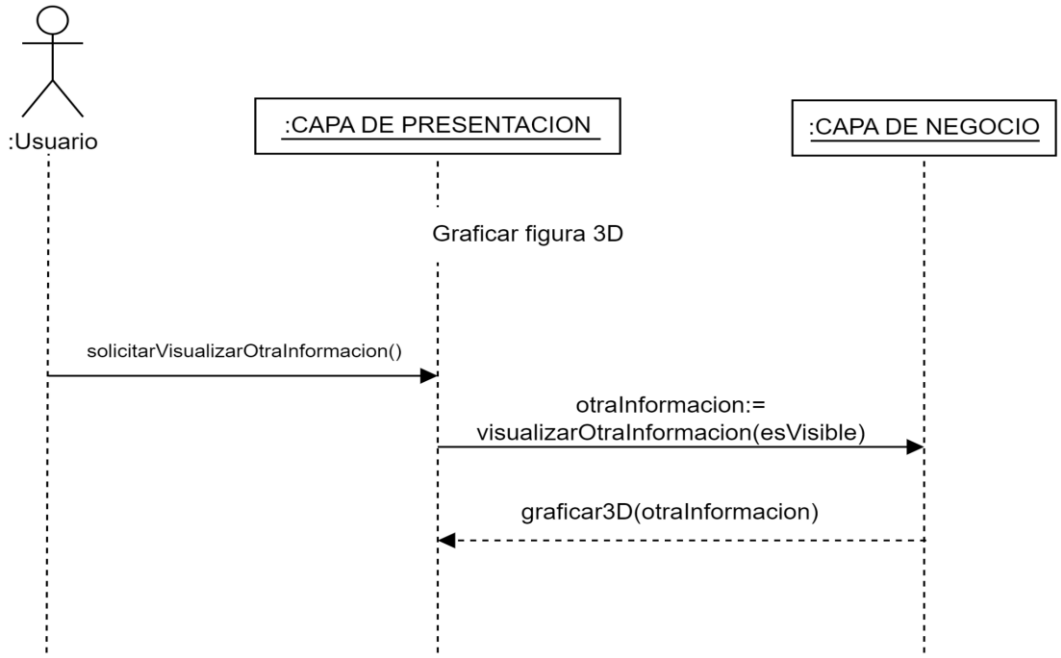
Figura 3.2.3.1: Menú de configuración del graficador 3D.

3.3. Visualizar otra información de figuras 3D

3.3.1. Descripción

El caso de uso comienza cuando un usuario quiere visualizar la otra información de las figuras 3D del panel de gráficos 3D. En los distintos casos, puede ser la altura y/o el radio de la figura. Para ello, el usuario despliega el menú de configuración, clickea el CheckBox cuya leyenda es "Otra información", y el sistema dibuja en el graficador 3D el resultado de la nueva configuración.

3.3.2. Diagrama de secuencia



3.3.3. Diseño en MateFun

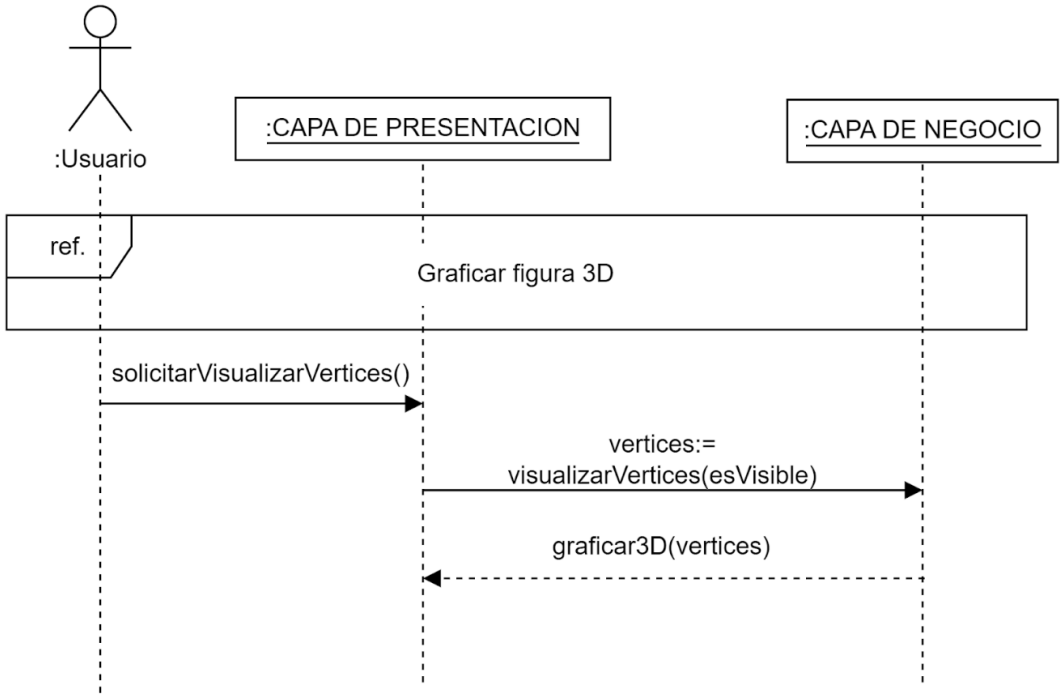
Como se muestra en el diagrama de secuencia, este caso de uso no cuenta con la participación del intérprete, ya que no se implementa el caso de uso mediante alguna función del intérprete. Toda la interacción para efectuar el caso de uso de esta sección estará disponible como un nuevo CheckBox -con leyenda "Otra información"- en el menú de configuración del graficador 3D (ver Figura 3.2.3.1).

3.4. Visualizar vértices de figuras 3D

3.4.1. Descripción

El caso de uso comienza cuando un usuario quiere visualizar los vértices de las figuras 3D del panel de gráficos 3D. Para ello, el usuario despliega el menú de configuración, clickea el CheckBox cuya leyenda es "Vértices", y el sistema dibuja en el graficador 3D el resultado de la nueva configuración.

3.4.2. Diagrama de secuencia



3.4.3. Diseño en MateFun

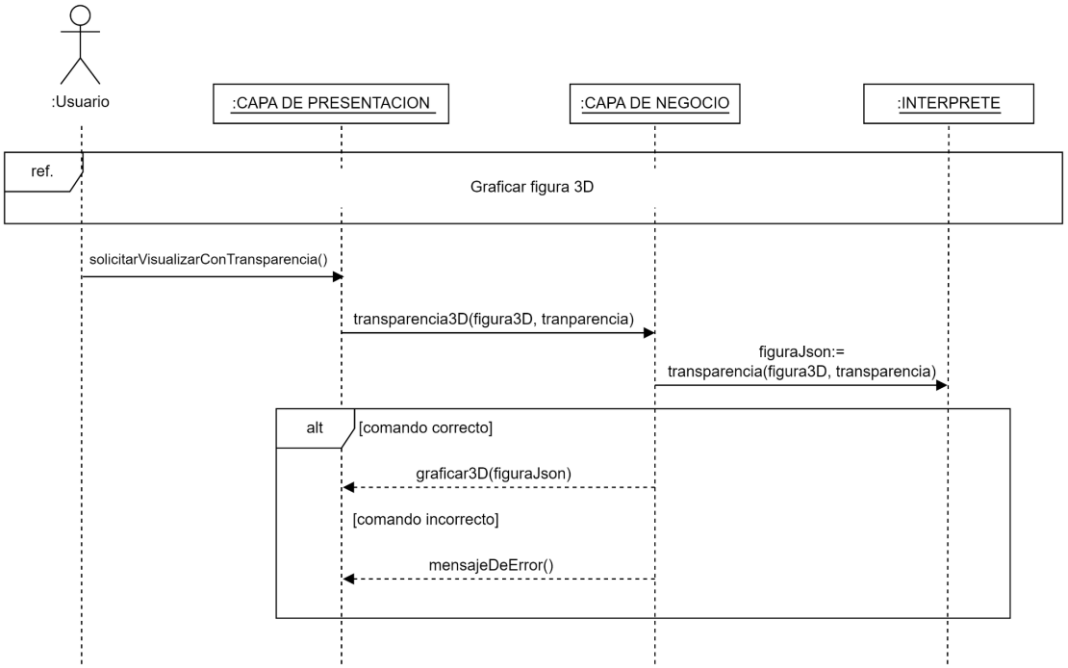
Como se muestra en el diagrama de secuencia, este caso de uso no cuenta con la participación del intérprete, ya que no se implementa el caso de uso mediante alguna función del intérprete. Toda la interacción para efectuar el caso de uso de esta sección estará disponible como un nuevo CheckBox -con leyenda "Vértices"- en el menú de configuración del graficador 3D (ver Figura 3.2.3.1).

3.5. Configurar transparencia de una figura 3D

3.5.1. Descripción

El caso de uso comienza cuando un usuario quiere configurar la transparencia de una figura 3D. Para ello, el usuario ingresa alguna función para graficar una figura 3D, la compone en otra función que habilita la transparencia de la figura, y el sistema dibuja en el graficador 3D el resultado de la ejecución de la función, o se devuelve un error en el intérprete en caso de que se haya instanciado mal la función.

3.5.2. Diagrama de secuencia



3.5.3. Diseño en MateFun

El anterior diagrama de secuencia muestra los flujos existentes cuando se quiere utilizar el comando para configurar la transparencia de una figura 3D. Esta nueva función se define de la siguiente forma:

- `transparencia3D :: (Fig3D X R) -> Fig3D`
  Dada una figura 3D y un valor de transparencia, actualiza el valor existente de transparencia de la figura.

El valor de transparencia varía entre 0 y 1. Con valor 0 la figura se ve completamente en el graficador 3D y con valor 1 la figura deja de ser visible en el graficador 3D. En inglés la función quedaría como `transparency3D`.

Casos bordes.

- La función admite cualquier valor de transparencia real. Luego, el módulo "Graph3D" se encarga de ajustar el valor de transparencia que le llega en el correspondiente JSON para graficar una figura 3D.
    - Los valores menores a cero serán tomados como cero.
    - Los valores mayores a uno serán tomados como uno.

- Si a una figura se le aplica la función **`transparencia3D`** más de una vez, esta tomará el último valor de transparencia que tenga la última función **`transparencia3D`** aplicada. Esta implementación es responsabilidad del módulo "MateFun Proceso".
    - **`transparencia3D(esfera(3), 0.5)`**
      Toma valor de transparencia al 50%.
    - **`transparencia3D(transparencia3D(esfera(3), 0.5), 0.25)`**
      Toma valor de transparencia al 25%.
    - **`esfera(3)`**
      Toma valor de transparencia al 0%.

Debido a este caso de uso, ahora todos los JSONs de las figuras 3D se les agrega un nuevo campo, **`transparency`**, al final. A modo de ejemplo, esta sería la estructura con el nuevo campo:

```
FIG3D: [
  {
    "kind": "<tipo de la figura>",
    <otra meta información asociada a la estructura de la figura a representar>
    "x": 0.0,
    "y": 0.0,
    "z": 0.0,
    "color": "white",
    "rot": (0.0, 0.0, 0.0),
    "transparency": <valor numérico>
  }
]
```

Ejemplo para la función `rotar3D(transparencia3D(cubo(3,5,7), 0.75), (90,45,75))`

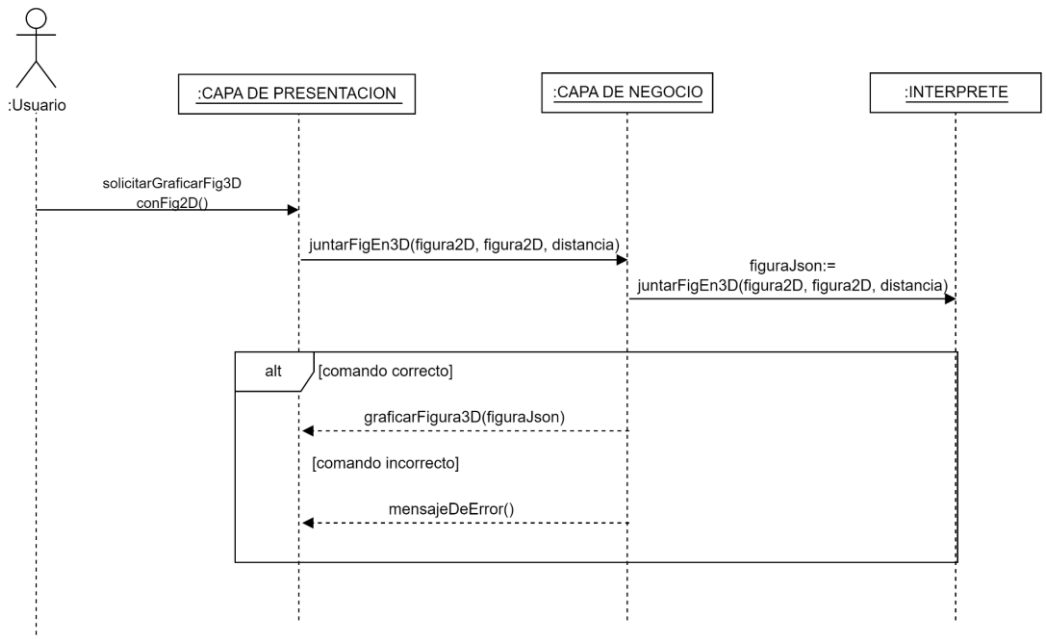
```
FIG3D: [
  {
    "kind": "cube",
    "w": 3.0,
    "h": 5.0,
    "l": 7.0,
    "x": 0.0,
    "y": 0.0,
    "z": 0.0,
    "color": "white",
    "rot": (90.0, 45.0, 75.0),
    "transparency": 0.75
  }
]
```

3.6. Crear una figura 3D mediante dos polígonos

3.6.1. Descripción

El caso de uso comienza cuando un usuario quiere graficar una figura 3D (un poliedro) a partir de dos polígonos, cuya cantidad de vértices coincide. Para ello, el usuario ingresa la función en el intérprete y le pasa como parámetro dos polígonos y la distancia que hay entre ellos, y el sistema dibuja en el graficador 3D el resultado de la ejecución de la función, o se devuelve un error en el intérprete en caso de que se haya instanciado mal la función.

3.6.2. Diagrama de secuencia



3.6.3. Diseño en MateFun

El anterior diagrama de secuencia muestra los flujos existentes para el comando que permite unir dos figuras en una figura 3D. Este nuevo comando se define de la siguiente forma:

- **`juntarFigEn3D :: (Fig X Fig X R) -> Fig3D`**
  Dadas dos figuras y una distancia, se dibuja una figura 3D que representa la unión en 3D de las dos figuras, las cuales están separadas por la distancia definida. Cada figura toma el rol de una base en la nueva figura 3D.

El valor de distancia puede ser cualquier real no negativo. En inglés la función quedaría como **`joinFigIn3D`**.

Casos bordes.

- La función admite cualquier valor de distancia real. Luego, el módulo "Graph3D" se encarga de ajustar el valor de distancia que le llega en el correspondiente JSON para graficar la figura 3D.
  - Los valores menores a cero serán tomados como cero.
  - Los valores mayores a uno no serán modificados.

- Las combinaciones válidas de figuras son las siguientes.
  - **`circ(<x>)`**, **`circ(<y>)`**
  - **`rect(<x>)`**, **`rect(<y>)`**
  - **`poli(<x>)`**, **`poli(<y>)`** → Donde ambos polígonos tienen la misma cantidad de vértices.

  En caso de que se de una combinación inválida de figuras 2D, el módulo "MateFun Proceso" se encargará de retornar un mensaje de error, como se muestra en el diagrama de secuencia.

- Si se aplica **`color3D`** a la figura resultante de **`juntarFigEn3D`**, no se tendrán en cuenta los colores que se definan para cada figura 2D en particular. Por otro lado, si la figura resultado de **`juntarFigEn3D`** no se le aplica **`color3D`**, sucederá que esta figura resultado será coloreada con el color que tenga la primera figura 2D (la figura 2D como parámetro más a la izquierda en la primitiva). Estos comportamientos quedan a cargo del módulo "Graph3D", no serán "pre-procesados" los colores en este sentido por el módulo "MateFun Proceso".

Para implementar el JSON de este nuevo caso de uso, se agrega el tipo (`kind`) `joinFigIn3D`. Además, este tipo de figuras tendrán dos campos particulares de las mismas, `f1` y `f2`. Estos campos contienen toda la información asociada a la primera figura 2D y a la segunda figura 2D, respectivamente. Dicha información es necesaria para poder dibujar la figura 3D correctamente a partir de las figuras 2D.

Ejemplo para la función **`color3D(juntarFigEn3D(circ(2), mover(circ(4), (2,4)), 5), Rojo)`**

```
FIG3D: [
  {
    "kind": "joinInFig3D",
    "f1": {
      "kind": "circle",
      "r": 2.0,
      "x": 0.0,
      "y": 0.0,
      "color": "white",
      "rot": 0.0,
    },
    "f2": {
      "kind": "circle",
      "r": 4.0,
      "x": 2.0,
      "y": 4.0,
      "color": "white",
      "rot": 0.0,
    },
    "h": 5.0,
    "x": 0.0,
    "y": 0.0,
    "z": 0.0,
    "color": "red",
    "rot": (0.0, 0.0, 0.0),
    "transparency": 0.0
  }
]
```

Ejemplo para la función **`juntarFigEn3D(poli([(1,2), (2,2), (2,1)]), color(poli([(4,5), (5,5), (5,4)]), Verde), 7)`**

```
FIG3D: [
  {
    "kind": "joinFigIn3D",
    "f1": {
      "kind": "poly",
```

```
      "pts": [
        [1.0, 2.0],
        [2.0, 2.0],
        [2.0, 1.0]
      ],
      "color": "white",
      "rot": 0.0,
    },
    "f2": {
      "kind": "poly",
      "pts": [
        [4.0, 5.0],
        [5.0, 5.0],
        [5.0, 4.0]
      ],
      "color": "green",
      "rot": 0.0,
    },
    "h": 7.0,
    "x": 0.0,
    "y": 0.0,
    "z": 0.0,
    "color": "white",
    "rot": (0.0, 0.0, 0.0),
    "transparency": 0.0
  }
]
```

Ejemplo para la función **juntarFigEn3D(mover(color(rect(0,0), Amarillo),(3,3)), rect(6,6), 9)**

```
FIG3D: [
  {
    "kind": "joinFigIn3D",
    "f1": {
      "kind": "rect",
      "w": 0.0,
      "h": 0.0,
      "x": 3.0,
      "y": 3.0,
      "color": "yellow",
      "rot": 0.0,
    },
    "f2": {
      "kind": "rect",
      "w": 6.0,
```

```
      "h": 6.0,
      "x": 0.0,
      "y": 0.0,
      "color": "white",
      "rot": 0.0,
    },
    "h": 9.0,
    "x": 0.0,
    "y": 0.0,
    "z": 0.0,
    "color": "white",
    "rot": (0.0, 0.0, 0.0),
    "transparency": 0.0
  }
]
```

Referencias

[1] Cameto, G. y Méndez, M. (2017). *Proyecto de Grado MateFun. Documento de Arquitectura y diseño*. https://gitlab.fing.edu.uy/matefun/Frontend/uploads/41c0624b45ad9d14d855c08262cc31cd/arquitectura-y-diseno__1_.pdf

[2] Google. (2020). *Documentación de Angular*. https://docs.angular.lat

[3] Bootstrap. (2022). *Bootstrap - The most popular HTML, CSS, and JS library in the world*. https://getbootstrap.com

[4] Replit. (2019). *JQConsole*. https://github.com/replit-archive/jq-console

[5] lt-codemirror (2022). *Angular - Codemirror component*. https://www.npmjs.com/package/lt-codemirror

[6] Andronache, M. (2022) *JAX-RS is just an API!* https://www.baeldung.com/jax-rs-spec-and-implementations

[7] Rey, D., Fagian, I. y Rosano, L. (2019) *Representación gráfica interactiva de funciones matemáticas, figuras geométricas y animaciones en la Web*. https://gitlab.fing.edu.uy/matefun/Frontend/-/wikis/uploads/4a9f5d4a3ee140511353e54f4f27e1c4/Matefun__1_.pdf

[8] Oracle. (2013). *Introduction to JPA*. https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html

[9] Vázquez, N. (2019). *Proyecto de Grado. Mejoras al intérprete MateFun*. https://gitlab.fing.edu.uy/matefun/MateFun/-/wikis/uploads/a88ea09300459d7feb33752ad0601713/MateFun_Informe.pdf