Beca de Colaboración

MIDDLEWARE *K2UM* 2.0

Grupo de Aplicaciones Multimedia y Acústica

Proyecto BLEXER

Daniel Iglesias Canelo

Julio 2019

Contenido

1. Estado de la versión 1.0. *K2UM* y *KinectAsset*

El punto de partida de este proyecto de beca de colaboración es el ***Middleware K2UM*** diseñado e implementado por César Luaces en su PFG (Proyecto Fin de Grado)[1], que consiste en un software de comunicación que permite interpretar la información captada por el hardware *Kinect v2* para *XBOX One*, adaptado para su uso con ordenadores mediante su conexión USB.

El *Middleware K2UM* puede recoger distintos tipos de datos de los sensores de *Kinect* de los cuales la versión de César utiliza principalmente la posición y rotación de las articulaciones de los cuerpos detectados. Este proyecto evolucionó a partir del *Middleware Chiro*, para *XBOX 360*, realizado por Ignacio Gómez-Martinho [2] como PFG. Sin embargo para desarrollar *K2UM* se usaron las herramientas aportadas por el ***SDK*** (*Software Development Kit)* para *Kinect v.2.*[3].

Este software se desarrolla a partir de la herramienta *Microsoft Visual Studio* [4] que permite diseñar y ejecutar código en lenguaje de programación **C#,** así como crear interfaces gráficas para hacer más manejable su uso.

En su estado original, el *middleware* divide su funcionamiento principalmente en dos partes:

- La primera parte consiste en la detección de los cuerpos mediante *Kinect* y obtención de los datos de posición y rotación del cuerpo. Para ello se usa una herramienta de programación denominada **evento**. Un evento se genera cuando se produce una acción provocada por el usuario o por otro programa, el cual provoca que se ejecute una parte concreta del código. Por ejemplo, esto se produce cuando se pulsa un botón de la interfaz gráfica. El software debe reconocer que ese botón ha sido pulsado y actuar en consecuencia según lo programado. En este caso, cada vez que *Kinect* envía un paquete de datos de posición y rotación, denominado **BodyFrame**, se produce un evento recogido por el método **manejador** del evento, denominado **Reader_FrameArrived**. Dentro de este método se realiza el procesado principal de los datos de *Kinect*.

- La segunda parte es la comunicación entre el *Middleware y Unity3D*. Esta comunicación se establece vía UDP entre dos ficheros de código incluidos tanto en el *KinectAsset* como en el *Middleware K2UM* que son **UDPReceive.cs** y **UDPSend.cs.** Los datos que se procesan en **Reader_FrameArrived** se adaptan en función de los datos solicitados por el *asset*, creando distintos paquetes de mensajes diferenciados entre mensajes de control y de datos. Estos mensajes se introducen en una cola de envío mediante el método **AddPacket**. Los paquetes se envían vía UDP a *Unity3D*. El *KinectAsset*, más concretamente la clase **Rotation.cs**, gestiona los paquetes UDP recibidos desglosando la información contenida en ellos para aplicar el movimiento específico al esqueleto, manejando así el avatar del videojuego. Esta parte incluye también un sistema de amplificación de movimiento basado en cinemática inversa el cual no ha sido modificado para las versiones del *asset* y del *middleware* actualizadas y que se encuentra explicada en el proyecto de César [1].

2. Objetivos de modificación

El objetivo de este documento es explicar de forma organizada el trabajo realizado durante los 8 meses de duración de la beca de colaboración. La mayor parte del trabajo se ha realizado a partir de los ejemplos aportados por el SDK de *Windows* para *Kinect* y de la versión anterior del *Middleware* para *Kinect XBOX 360*. Se ha intentado respetar lo máximo posible el diseño original de César Luaces.

Por otra parte la última versión del Middleware que incluye también todos los cambios explicados en este documento es la realizada por Gerardo Cilleruelo. Sin embargo, el

funcionamiento de esa versión únicamente se ha comprobado con el videojuego *West Gun*, que también incluye diversas modificaciones del *asset* de *Kinect*. Si en algún momento se desea continuar realizando cambios, se recomienda hacer compatible el *Middleware* de Gerardo con el resto de videojuegos como *Phiby's adventures* para poder utilizar por ejemplo la funcionalidad de movimiento del puntero del ratón a través del brazo.

De momento, la última versión para usar con videojuegos como *Phiby* es la versión de Abril 2019 que incluye los cambios que se explican a continuación:

- Implementación de la visualización de la cámara y el esqueleto. Permite visualizar las imágenes captadas por la cámara de *Kinect* y en caso de que el dispositivo las identifique como cuerpos, dibuja los esqueletos encima.
- Comunicación con el entorno web *Blexer Med*, plataforma en la que se recogen los datos de los usuarios. Hasta ahora solo recogía los datos del *Middleware Chiro*, los cuales han servido de referencia para elaborar las distintas partes de este proceso de comunicación. El primer proceso es la identificación del usuario con su nombre de usuario y contraseña para descargar la configuración del mismo. El siguiente paso es enviar la configuración del videojuego específico a *Unity3D* usando la comunicación UDP existente. El videojuego extrae los parámetros de configuración y al terminar cada ejercicio envía los resultados al *middleware*. Cuando se ha terminado de realizar ejercicios, se envían todos los resultados a la web.

3. Implementación de la visualización de la cámara y el esqueleto

Para realizar esta parte, se ha necesitado estudiar los ejemplos incluidos en el *SDK* de *Kinect v.2*, escritos en código C#. También se ha estudiado la implementación realizada de esta misma funcionalidad para la versión de *Kinect* para *XBOX 360* del *Middleware Chiro*.

Para empezar es necesario entender cómo se recogen los datos de cada una de las variables que recoge *Kinect*. Por cada uno de los tipos de datos es necesario inicializar un lector específico, que se indican a continuación:

- **ColorFrameReader**: Obtiene datos de píxeles de color de la imagen captada.
- **InfraredFrameReader**: Obtiene los datos captados por el sensor de radiación infrarroja.
- **LongExposureInfraredFrameReader:** Datos de radiación infrarroja pero tomados con mayor exposición.
- **DepthFrameReader:** Lector de datos de profundidad, distancia al sensor en mm.
- **BodyFrameReader**: Lector de datos de articulaciones de los esqueletos detectados.
- **AudioBeamFrameReader:** Lector de datos de audio captados por el micrófono.

Sin embargo es posible obtener los datos a través de un lector múltiple para todos los tipos de datos que se denomina **MultiSourceFrameReader.** Con una frecuencia de 30 veces por segundo se reciben datos del sensor de *Kinect*. Cada vez que llega uno de estos datos, un evento se activa haciendo que se ejecute el método **Reader_FrameArrived(…)**. Este método separa los componentes de datos solicitados obtenidos en el paquete de datos de *Kinect* para procesarlos de forma independiente.

En este caso, los únicos datos que han resultado necesarios han sido **ColorFrame** (datos de píxeles) y **BodyFrame** (datos de articulaciones).

Para procesar los datos de **ColorFrame** se recoge la imagen capturada y se copia a un objeto **Bitmap,** para el cual es necesario reservar un espacio de memoria de tamaño 1080 x 1920

píxeles y de 4 bytes por píxel para almacenar los componentes rojo, verde, azul y alfa de la imagen.

Una vez se han recogido los datos de imagen es necesario determinar si hay cuerpos detectados por *Kinect*. Si no se detectan cuerpos, se muestra la imagen de la cámara tal cual se ha recibido. Sin embargo, si se detectan, hay que procesar los datos de posición de cada una de las articulaciones y dibujar las articulaciones y huesos encima de la persona detectada. Por suerte, el procedimiento para realizar esta operación es muy similar al utilizado para el *Middleware Chiro*.

Se recogen los datos del **BodyFrame** y se obtiene el número de esqueletos detectados. El primer paso es crear un objeto **Graphics** que será el equivalente a un lienzo sobre el que se dibujarán los esqueletos. El fondo de este objeto será la imagen de la cámara captada anteriormente.

Las articulaciones y su posición se guardan en un diccionario de articulaciones. Se dibujan por separado las articulaciones y los huesos de cada esqueleto. Para ello se usa la herramienta de la clase **Pen**, que permite rellenar de un color un objeto dibujado en el lienzo. Los huesos y articulaciones se pintan con el método **DrawBody(…)**. Para dibujar cada hueso se ha creado el método **DrawBone(…)**, que dibuja el hueso indicado por los parámetros que se le indican. El color de los huesos varía en función del índice del esqueleto asignado. Las articulaciones se dibujan con un círculo verde en la posición de la articulación solo cuando la articulación está bien traqueada. Si no está bien traqueada, el color es amarillo. Si alguna articulación queda fuera del rango del sensor de *Kinect*, se dibuja el hueso asociado como una línea gris. Se muestra un ejemplo, aunque un poco mal traqueado debido a la cercanía a la cámara, en la Figura 1.

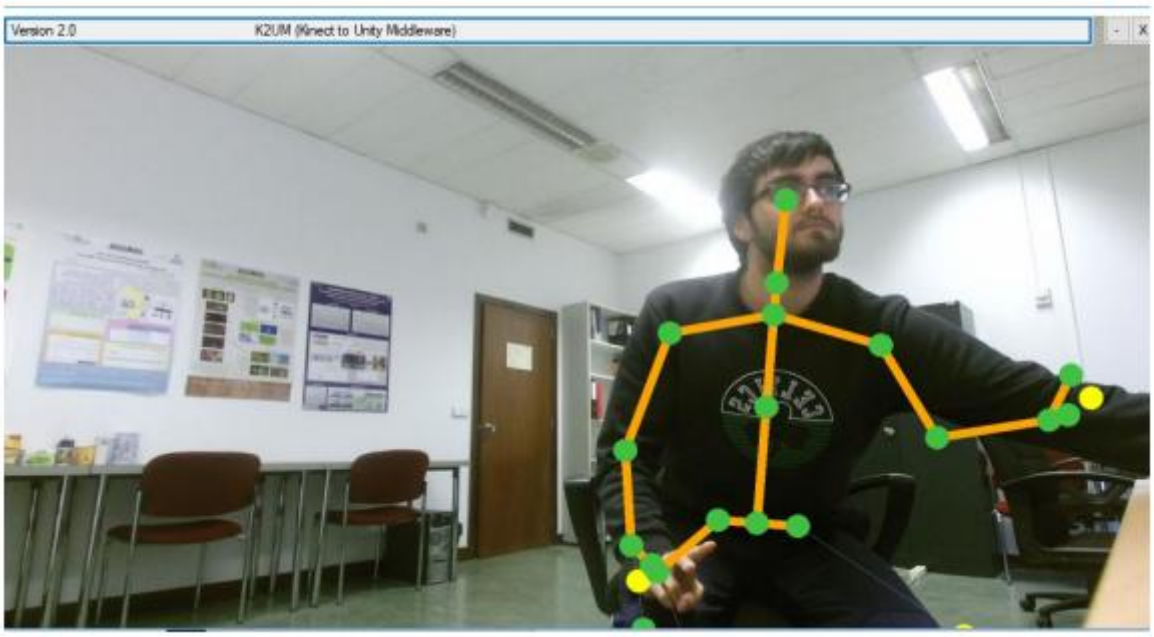

Figura 1. Visualización de la cámara y el esqueleto.

Una vez se han dibujado todos los esqueletos, se actualiza la imagen del *bitmap* que está asociada al fondo de la ventana **Form2**. Esta ventana se abre como una ventana independiente y se puede ajustar su tamaño. Para abrirla es necesario pulsar el botón **Esqueleto** de la ventana principal del *middleware* (Figura 2).

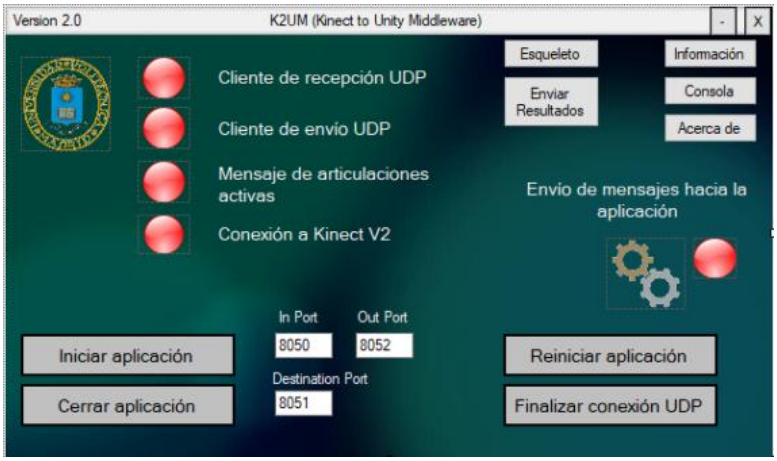

Figura 2. Ventana principal del Middleware K2UM.

4. Comunicación con el entorno web *Blexer-Med*

El segundo de los objetivos de esta beca de colaboración es desarrollar la comunicación con el entorno web *Blexer Med*, de la misma forma que fue desarrollado para el *Middleware Chiro* [2], de hecho la mayor parte del desarrollo se ha realizado reutilizando parte del código de este *middleware,* aunque en otros casos se ha desarrollado de forma autónoma el código restante.

4.1. Obtención de la configuración *Blexer-Med → K2UM*

El primer paso es obtener la configuración de un usuario concreto a partir de su autenticación con la plataforma web. Este proceso se ha copiado prácticamente idéntico del *Middleware Chiro* cambiando únicamente lo necesario. El código correspondiente a esta parte se recoge en el archivo **LoginForm.cs**.

Para ver el código de un archivo «.cs» que se corresponde con una interfaz de usuario hay que pulsar botón derecho en el nombre del archivo en el **Explorador de Soluciones** y elegir la opción «**ver código**» (Figura 3).

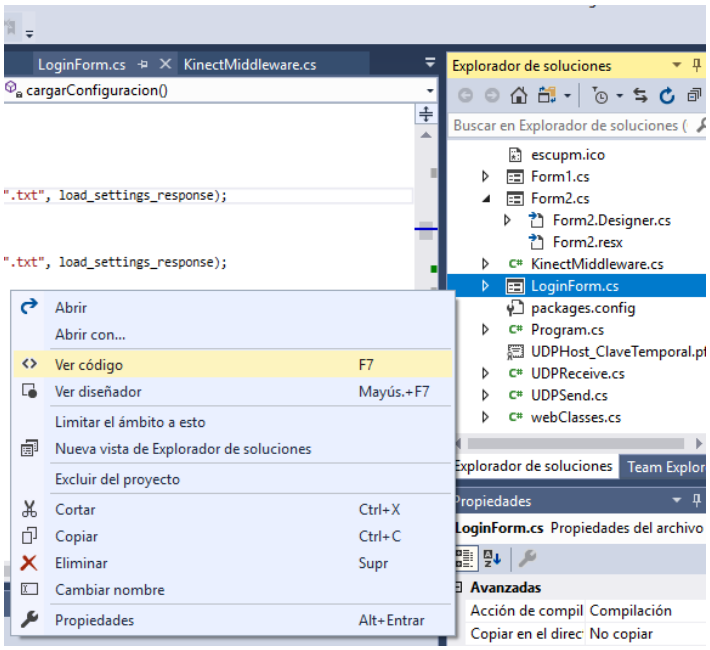

Figura 3. Abrir código de una interfaz de usuario en Visual Studio.

La primera ventana que se abre al iniciar el programa solicita los datos del usuario. Si el usuario se encuentra registrado como paciente en la plataforma *Blexer Med* deberá introducir sus datos

de usuario y contraseña. En caso contrario deberá continuar sin cargar un archivo de configuración. Esta ventana se muestra en la Figura 4.

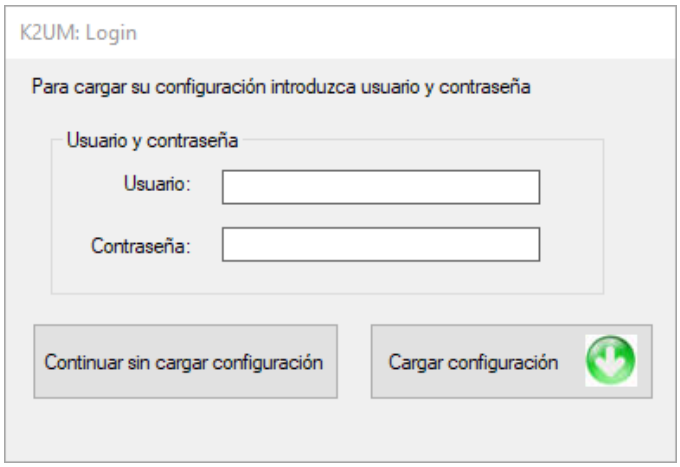

Figura 4. Ventana de *login* del *middleware*.

En caso de tratar de cargar configuración sin introducir usuario o contraseña se muestra un mensaje de error como el de la Figura 5 a la izquierda. Si se introduce un usuario y contraseña se envía una petición mediante el protocolo HTTP (*Hypertext Transfer Protocol*) al entorno web *Blexer Med* incluyendo el nombre de usuario y contraseña. La plataforma responde con un mensaje indicando si la autenticación es correcta o no. Los distintos mensajes que se pueden recibir son:

- **OK**. Indica que la autenticación se ha realizado con éxito. Se guarda el nombre del usuario en un fichero de texto dentro del directorio «**.\k2umfiles\login\login.txt**».
- **NOK**. Indica que el usuario no se encuentra registrado en la web o la contraseña introducida no es correcta. En este caso se muestra una ventana como la de la Figura 5 a la derecha.
- **ERR**. Se recibe este mensaje cuando el servidor web no se encuentra disponible desde internet. En caso de que exista una configuración antigua disponible se usa dicha configuración y se muestra un mensaje como el de la Figura 5 abajo. En otro caso de muestra un mensaje que indica “ERROR EN CONEXIÓN CON DB” y se debe continuar sin configuración.

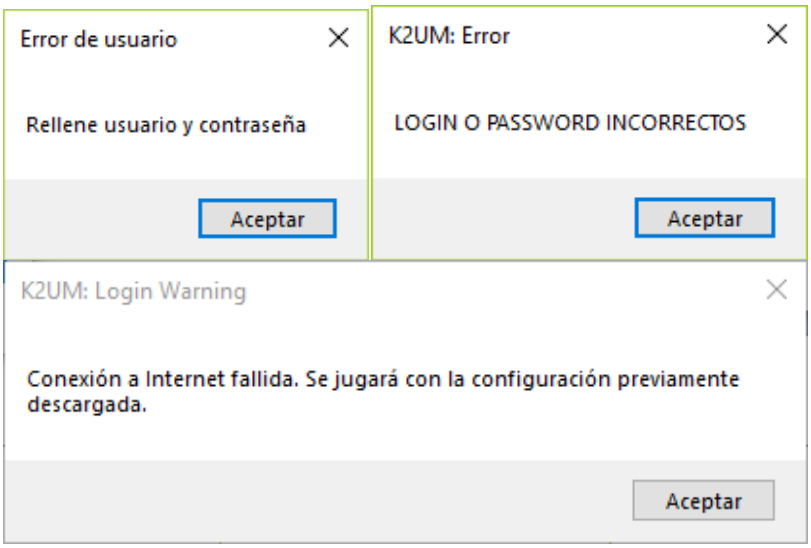

Figura 5. Mensajes de error de usuario. Izq: Error por no introducir usuario o contraseña. Dcha: Error por usuario o contraseña no registrados en la web. Abajo: Error por conexión fallida con el servidor.

En caso de que el ordenador del usuario no tenga conexión a internet disponible en ese momento, se produce una excepción al enviar la solicitud de autenticación y se realiza el mismo procedimiento que en caso de mensaje de error recibido. En caso de haber una configuración previa guardada, ésta se utilizará; si no se muestra un mensaje como el de la Figura 6.

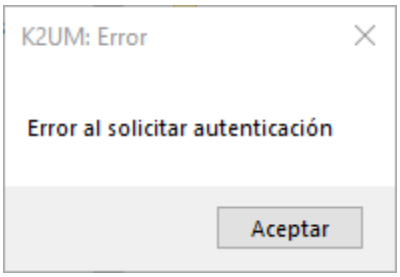

Figura 6. Mensaje de error debido a que el usuario no tiene conexión a internet y no hay un fichero de configuración previo guardado.

En caso de que se haya realizado correctamente la autenticación se procede a solicitar la configuración mediante otra solicitud HTTP, cuya respuesta es una cadena de texto que se guarda en un fichero nombrado como el nombre de usuario. Este fichero se guarda en la ruta «**.\k2umfiles\users-settings\"*login*".txt**», siendo *"login"* el nombre de usuario. Una vez se ha guardado la configuración en el fichero se muestra un mensaje como el de la Figura 7.

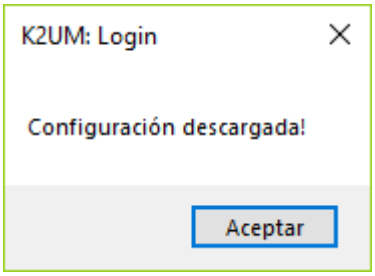

Figura 7. Mensaje de configuración descargada.

Antes de continuar, se comprueba si hay algún fichero de resultados que no se pudiera enviar en la última sesión. De ser así se ejecutará un método incluido en el código denominado **comprobarResultadosSinEnviar()** que se explica en la parte de envío de resultados. Tras esto se inicia la ventana principal del *middleware*.

Como resumen de este proceso de solicitud de configuración se muestra el diagrama de flujo de la Figura 8.

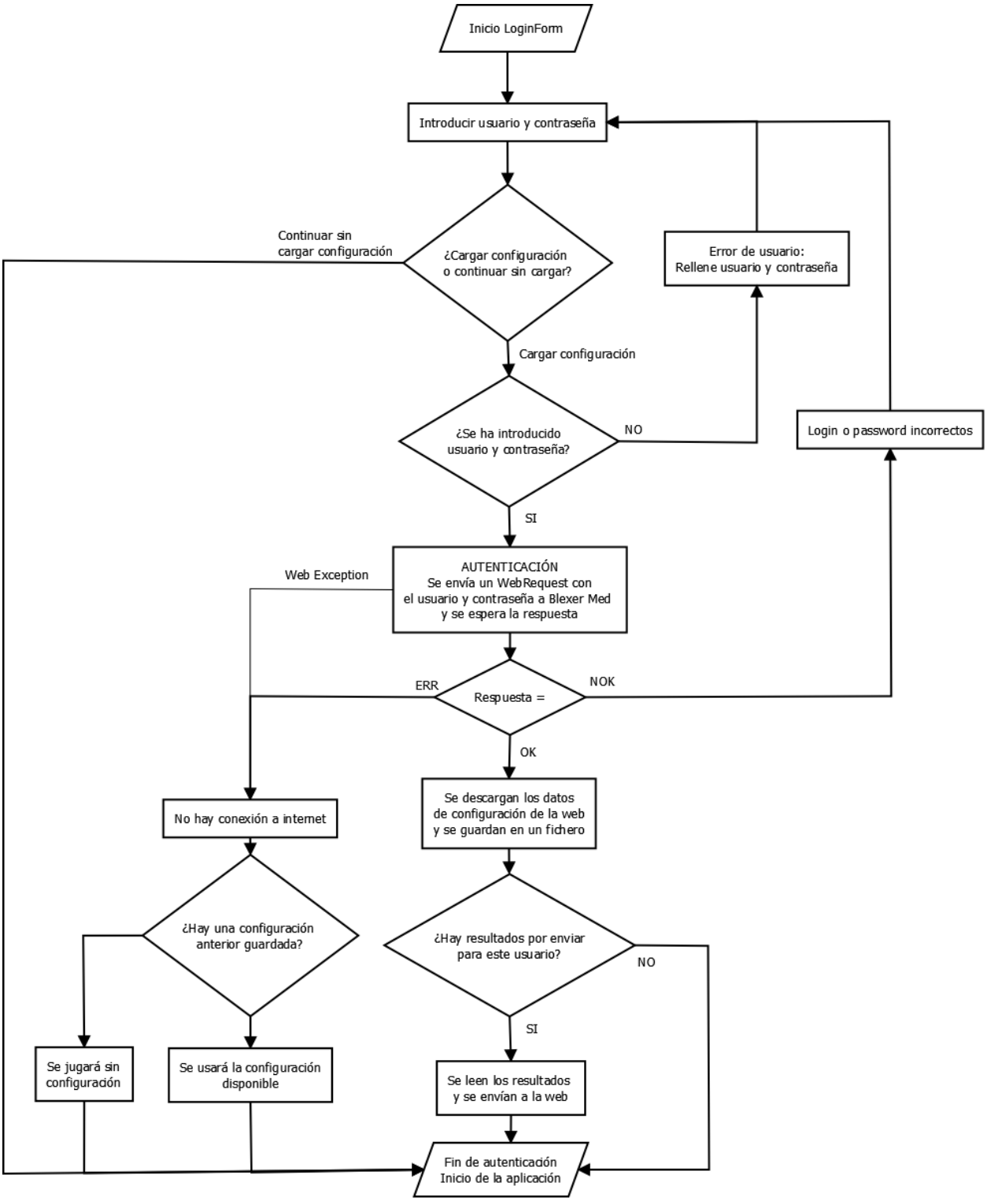

Figura 8. Diagrama de flujo de solicitud de la configuración.

4.2. Solicitud de configuración *Unity3D → K2UM*

Una vez se ha descargado la configuración del usuario se debe iniciar la conexión con *Kinect*. Para poder configurar el videojuego, éste debe haber sido desarrollado con la nueva versión de los scripts del *asset* de *Kinect* para *Unity3D.* En esta versión, este *asset* únicamente ha sido modificado para permitir la configuración del videojuego, siendo compatible con la primera versión del *middleware* para operar sin configuración.

El primer paso de este apartado es implementar un mensaje de solicitud de configuración que se envíe desde *Unity3D* al *middleware* cada vez que se inicia el videojuego. Este mensaje se envía como mensaje de control de "*Setting Request*" con el formato de la Tabla 1.

Tabla 1. Mensaje de solicitud de configuración.

K2UM	**CC**	**####**	**SR**	**"[gameId, gameCode, gameTitle]"**
Cabecera	Tipo	Longitud Datos	Clase	Datos
8 bytes	2 bytes	4 bytes	2 bytes	N bytes

La gestión del envío de este mensaje se realiza en el script **SettingManager.cs**. Cada 100 frames del videojuego se envía esta solicitud de configuración hasta que se recibe un mensaje con los datos del usuario.

En el *middleware* se recibe este mensaje de solicitud y se procesa extrayendo los datos del juego solicitado. Estos datos obtenidos en la solicitud son necesarios para identificar la configuración de un juego concreto dentro del archivo de texto de configuración en formato JSON.

Los formatos de los datos de configuración y resultados se recogen en el fichero **webClasses.cs** que están implementados de forma que sean serializables. Sin embargo hay una incompatibilidad entre la serialización en la web y la serialización en el middleware, provocada porque no hay un tipo de dato en el lenguaje C# en el que se pueda deserializar correctamente los datos de configuración de los videojuegos. Es decir, no se ha encontrado un tipo de dato que coincida con el que se ha usado en la web a la hora de serializar la configuración . Para solucionar este proceso se ha diseñado un método que deserializa los datos del juego solicitado manejando únicamente cadenas de caracteres. Este método se denomina **deserializarJuegoSolicitado()** y se encuentra dentro de la clase **KinectMiddleware.cs**.

Los datos de configuración están formados por los siguientes campos:

- **id_user.** Identificador de usuario, número entero.
- **login.** Apodo usado en la cuenta del usuario. Cadena de caracteres.
- **first_name.** Nombre del paciente. Cadena de caracteres.
- **last_name.** Apellido del paciente. Cadena de caracteres.
- **gameRequest.** Campo de tipo **Game** que recoge los datos del juego solicitado.

A su vez, el tipo **Game** está formado por los siguientes parámetros:

- **gameInfo.** Cadena de caracteres que identifica el juego concreto, que tiene el siguiente formato: "[game_id, game_code, title]".
- **exercises.** Array de objetos de tipo **Exercise**, que incluye los datos de todos los ejercicios del videojuego solicitado.

Los objetos de tipo **Exercise** contienen información sobre los datos del ejercicio y los parámetros de configuración del mismo. Se divide en:

- **ExerciseInfo.** Cadena de caracteres que identifica el ejercicio. Su formato es el siguiente: "[idExercise, codeExercise, exerciseName]".
- **setting.** Objeto de tipo **Setting**, que incluye los parámetros de configuración del ejercicio. Estos parámetros de configuración son un número entero que identifica la

configuración denominado **id_setting**; y cuatro cadenas de caracteres: **param1**, **param2**, **param3** y **param4**.

Los datos deserializados se recogen en la variable global **loadSettingResponse**.

4.3. Envío de configuración *K2UM → Unity 3D*

Una vez se tiene en memoria la información de configuración solicitada del videojuego se debe enviar un mensaje vía conexión UDP a *Unity3D*. Este mensaje tiene el formato que se muestra en la Tabla 2.

Tabla 2. Mensaje de envío de configuración.

K2UM	CC	####	SS	«Datos de configuración (JSON)»
Cabecera	Tipo	Longitud Datos	Clase	Datos
8 bytes	2 bytes	4 bytes	2 bytes	N bytes

Para generar este mensaje, primero se serializa la variable **loadSettingResponse** en una cadena de texto usando el método **serializarConfiguración()**. Este mensaje se introduce en la cola de mensajes para enviar a *Unity3D* usando el método **enviarConfiguración()**.

El *script* **UDPReceive.cs** del *Kinect Asset* recibe el mensaje, el cual pasa al script **Rotation.cs**. En este script se extrae el mensaje de datos de configuración que se guarda en la variable **configuracion**. Esta variable puede tomar 3 estados:

- Si no se ha recibido un mensaje de clase **SS**, quiere decir que el *middleware* no ha recibido o procesado la solicitud de configuración, en este caso se debe seguir enviando la solicitud de configuración. Para identificar este caso, la variable **configuracion** permanece como una cadena vacía (**“”**).
- Si el mensaje recibido es de clase **SS** pero los datos recibidos también están vacíos, quiere decir que se ha recibido en el *middleware* la solicitud de configuración pero el usuario eligió la opción de **continuar sin cargar configuración.** Por tanto, no hay configuración que procesar. Para identificar este caso, se le asigna a la variable **configuracion** la cadena de caracteres “**null**”. De esta forma, cada vez que se quiere identificar si se juega sin configuración solo hay que comprobar si el valor de la variable coincide con la cadena “**null**”.
- Si se recibe el mensaje de clase **SS**, cuyos datos no estén vacíos, se guarda su contenido en la variable **configuracion**.

Cada 100 frames, el *script* **SettingManager.cs** comprueba si se ha recibido la configuración. De ser así, deja de enviar la solicitud de configuración y procesa la información recibida. Para comprobar si la información es correcta accede a la variable **configuracion** del *script* **Rotation.cs**. Si el valor de la variable es la cadena **“null”**, termina el proceso de configuración puesto que la opción elegida es **continuar sin cargar configuración**. Si no, se deserializa la información de la variable **configuracion** usando el método **deserializarConfiguracion().** Este método utiliza la clase **JsonUtility** incluida en las librerías de código de *Unity3D*, la cual permite deserializar fácilmente la información.

La información de configuración de los ejercicios se guarda en un **diccionario** de ejercicios, almacenado en el *script* **SettingSustain.cs**. Este *script* está diseñado para que el objeto al que está asociado no se destruya al cambiar de escena. Se crea un objeto vacío llamado **SettingObject** al que asignar el *script* y éste no será eliminado al cambiar de escena. Así, la información de configuración queda guardada en memoria durante el resto de la partida.

4.4. *Kinect Asset:* Configuración y resultados de ejercicios

Para cargar la configuración de cada ejercicio se aporta el *script* **exerciseManager.cs** que sirve como ejemplo de código para recoger los parámetros de cierto ejercicio. Es necesario conocer los datos del ejercicio denominados **idExercise**, **codeExercise** y **exerciseName** puesto que son los que se utilizan para obtener la configuración desde el diccionario de ejercicios. Para asignar los valores correctos a estos parámetros se puede modificar directamente el código o asignarlos en el inspector de *Unity3D*.

El primer paso, mientras se carga la escena concreta, es comprobar si es necesario cargar la configuración. Para ello se busca el objeto **SettingObject,** que es el objeto al que se ha asignado la configuración. Si existe este objeto, se han obtenido datos de configuración y por tanto hay que cargarlos obteniéndolos del diccionario de ejercicios del *script* **SettingSustain.cs**. Si no, se salta todo el proceso de configuración y envío de resultados.

Los parámetros extraídos: **param1**, **param2**, **param3** y **param4**; se deben asignar a variables las cuales puedan ser accesibles posteriormente. El tratamiento de la configuración de los ejercicios a partir de este paso queda en manos del desarrollador del videojuego.

Al finalizar el ejercicio, se deben recoger los resultados y mandarlos al *middleware*. Para ello, el *script* **exerciseManager.cs** también incluye la gestión de los resultados a través del método **OnGameSuccess()**. El formato de los resultados también se recoge en el *script* **webClasses.cs** y se desglosa a continuación, definido en la clase **ExerciseResult**:

- **date:** Fecha de finalización del ejercicio.
- **id_exercise:** Número identificador del ejercicio. Ej: 3892.
- **code_exercise:** Código en cadena de caracteres que identifica al ejercicio. Ej: “CHOP”.
- **id_setting:** Código numérico que identifica el archivo de configuración utilizado.
- **param1:** Parámetro 1 de configuración utilizado.
- **param2:** Parámetro 2 de configuración utilizado.
- **param3:** Parámetro 3 de configuración utilizado.
- **param4:** Parámetro 4 de configuración utilizado.
- **duration:** Duración en segundos del ejercicio.
- **attemps:** Número de intentos realizados del ejercicio.
- **corrects:** Número de intentos superados.

La variable de tipo **ExerciseResult** se serializa en formato JSON y se transforma en una cadena de caracteres que se envía a continuación.

4.5. Envío de resultados *Unity 3D → K2UM*

Dentro del *script* **exerciseManager.cs,** se envían los resultados del ejercicio con el formato de la Tabla 3 al *middleware.*

Tabla 3. Mensaje de resultado de un ejercicio.

K2UM	CC	####	FR	«Resultado de un ejercicio (JSON)»
Cabecera	Tipo	Longitud Datos	Clase	Datos
8 bytes	2 bytes	4 bytes	2 bytes	N bytes

Para enviar este mensaje al middleware se usa el método **sendString()** del *script* **UDPSend.cs**.

El mensaje es procesado en el *middleware* que guarda el resultado de cada ejercicio en una lista. Cada vez que llega un mensaje con resultados, se deserializa el resultado usando el método **deserializarExerciseResult()** y se añade a la lista usando el método **addResult()** de la clase **KinectMiddleware.cs**. Esta lista de resultados se recoge dentro de la variable global **list**.

4.6. Envío de resultados *K2UM → Blexer Med*

Una vez se han obtenido uno o más resultados de ejercicios se pueden enviar a la plataforma *BlexerMed.* Al pulsar el botón **enviarResultados** de la ventana gráfica se genera un evento que es recogido por el método **button8_Click_1()** de la clase **Form1.cs**. Dentro de este método se usa otro llamado **generarResultados()** que se encuentra en **KinectMiddleware.cs**. Este fragmento de código guarda los resultados obtenidos en un fichero de texto en formato JSON.

Tras generar el fichero de resultados se llama al método **comprobarResultadosSinEnviar()** de la clase **LoginForm.cs** que envía los resultados al entorno web si se dispone de conexión a internet. El método de envío es similar al de solicitud de configuración.

En primer lugar se comprueba que hay un fichero de resultados. Si este fichero existe, se leen los resultados existentes uno a uno y se envían usando el método **enviarResultado()** el cual genera un mensaje que se envía a la web usando el protocolo HTTP.

La plataforma web puede responder con 3 tipos de mensajes:

- **OK.** El resultado se ha recibido correctamente.
- **SQL_ERROR**. Alguno de los parámetros en el archivo de resultados no coincide con lo esperado en la web.
- **CONEX_ERROR**. Error en la conexión con la web.

En caso de que alguno de los resultados no se pudiera enviar a la web, se guardarán en el ordenador en un nuevo fichero de texto para intentar enviarlos en la próxima sesión.

Los resultados también se intentarán enviar automáticamente al cerrar la aplicación en caso de que no se hayan enviado anteriormente.

Como resumen del proceso de configuración y resultados se aporta un diagrama que se muestra en la Figura 9.

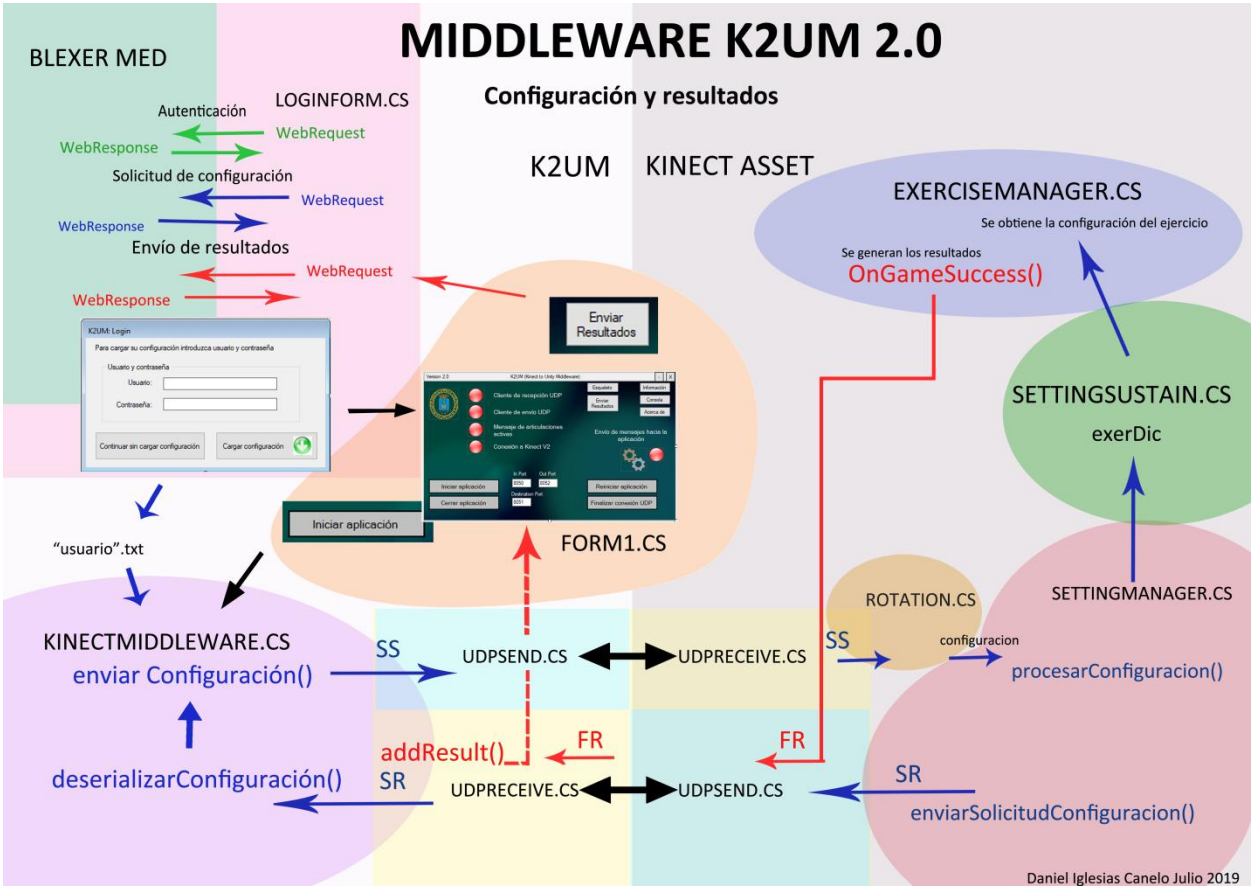

Figura 9. Diagrama del proceso de configuración y resultados.

5. Conclusiones

Se han cumplido los objetivos principales de este proyecto de beca de colaboración que eran la visualización del esqueleto y la comunicación con el entorno web *Blexer Med*. Para llevarlo a cabo se ha tratado de mantener la compatibilidad con la versión anterior del *Middleware K2UM* y con la plataforma web. Esto implica que futuros cambios en el formato de la configuración y los resultados también deberán ser reflejados en el *middleware*.

Existe una versión posterior denominada *K2UM 3.0* desarrollada por Gerardo Cilleruelo que ya incluye todos los cambios aquí mencionados. Por tanto, si se plantea estudiar el código de la aplicación se recomienda hacerlo en esa versión del proyecto.

Se espera que el trabajo realizado sea de utilidad para llevar hacia delante este proyecto.

6. Bibliografía y referencias

[1] C. L. Vela, «Diseño e implementación de un entorno virtual de ejercicios físicos, basados en captura de movimiento.,» UPM, ETSIST, Madrid, 2018.

[2] I. Gómez-Martinho, «Desarrollo e implementación de middleware entre Blender, Kinect y otros dispositivos,» UPM ETSIST, Madrid, 2016.

[3] «Download Kinect for Windows SDK 2.0 from Official Microsoft Download Center,» Microsoft, [En línea]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=44561. [Último acceso: 10 julio 2019].

[4] «IDE de Visual Studio, editor de código, Azure DevOps y App Center - Visual Studio,» Microsoft, [En línea]. Available: https://visualstudio.microsoft.com/es/?rr=https%3A%2F%2Fwww.google.com%2F. [Último acceso: 10 julio 2019].