UNIVERSIDADE
DE VIGO

ESCOLA SUPERIOR DE ENXEÑARÍA INFORMÁTICA

Memoria do Traballo de Fin de Grao que presenta

D. Iván Martínez Estévez

para a obtención do Título de Graduado en Enxeñaría Informática

Acoplamiento del modelo DualSPHysics con una librería de amarres

Maio, 2019

Traballo de Fin de Grao Nº: EI 18/19 - 010

Titor/a: Javier Rodeiro Iglesias
Cotitor/a: José Manuel Domínguez Alonso
Área de coñecemento: Linguaxes e Sistemas Informáticos
Departamento: Informática

Agradecimientos

Quisiera dedicarle unas palabras a todas las personas que me han ayudado y que sin su ayuda no lo hubiese conseguido.

A Javier Rodeiro, mi tutor de TFG, por haber confiado en mí desde el principio y haberme dirigido este trabajo. Por todas las reuniones, la resolución de mis dudas y por sus consejos. En general, agradecer todo el tiempo que ha invertido en mí, para poder sacar adelante este trabajo.

A mi cotutor José M. Domínguez y a Alejandro J. C. Crespo, por haber confiado en mí y ofrecerme la oportunidad de realizar este trabajo con ellos. Por haberme dedicado el tiempo necesario ayudándome en todo lo posible y por transmitirme sus conocimientos para poder realizar este trabajo de una manera más sencilla.

Agradecer también al resto de mis compañeros de EPhysLab por el trato recibido y hacerme sentir como uno más desde el momento en el que me incorporé al grupo.

Finalmente, y no menos importante, agradecer a mis familiares, amigos, a las personas que están y han estado presentes en mi vida ayudándome en todos los aspectos de mi vida.

Muchas gracias a todos,

Iván Martínez Estévez

Índice de contenidos

Índice de figuras

Memoria

Anexos

Índice de tablas

Memoria

Iván Martínez Estévez

1. Introducción

Este trabajo fin de grado (TFG) ha sido realizado con el grupo de investigación EPhysLab (Environmental Physics Laboratory), del Campus de Ourense, que pertenece al Departamento de Física Aplicada de la Universidad de Vigo.

El grupo EPhysLab es uno de los principales desarrolladores de un modelo computacional de fluidos utilizado principalmente en problemas de ingeniería costera. El modelo, llamado DualSPHysics [1], es código abierto y se distribuye de manera gratuita con licencia LGPL. DualSPHysics ha sido aplicado con éxito a problemas de interacción entre oleaje y estructuras costeras (diques, rompeolas, pasos marítimos), pero también a la simulación de dispositivos flotantes en el mar (boyas de medición, aparatos de conversión de energía de las olas, plataformas, etc…). En el caso del estudio numérico de objetos flotantes, una parte muy importante de la simulación es el efecto de los amarres que conectan el dispositivo con el fondo marino. DualSPHysics resuelve de manera precisa la interacción entre las olas y el objeto flotante, pero no incluye la fuerza (o tensión) ejercida por los amarres. De este modo, en este trabajo se realizará un acoplamiento entre DualSPHysics y una librería que resuelva las fuerzas que ejercen los amarres y que tenga en cuenta las propiedades del amarre (material, densidad, volumen, elasticidad), el rozamiento con el suelo y la posibilidad de definir varios amarres conectados al mismo objeto flotante.

Un objeto flotante es un cuerpo rígido que sumergido en un fluido (en este caso en agua) ascenderá o descenderá si su densidad es menor o mayor al del fluido, respectivamente. Los amarres son un conjunto de líneas (o cadenas) que están conectadas a un mismo objeto flotante. Las conexiones son los extremos de los amarres que conectan al objeto flotante con el fondo. En la Figura 1-1 se muestran dos objetos flotantes amarrados independientemente.

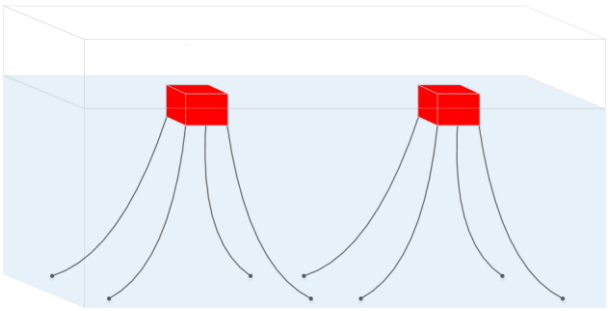


Figura 1-1: Dos objetos flotantes con cuatro líneas de amarre cada uno

La primera tarea de este TFG ha sido realizar una búsqueda de librerías ya existentes que permitan la simulación numérica de amarres y su acoplamiento con el software DualSPHysics para sistemas operativos Windows y Linux. Se encontraron dos librerías que, a priori, reunían los requisitos: **MoorDyn** [2] y **MooDy** [3].

MoorDyn es una librería de código abierto que está desarrollada en C++ por investigadores de la UPEI (University of Prince Edward Island). Esta librería presenta una serie de limitaciones como que la mayor parte de las funciones están en el fichero *MoorDyn.cpp*, por lo que el sistema tiene una baja cohesión. No permitir configurar amarres a varios objetos flotantes. Es decir, no permite definir casos como el de la Figura 1-1, donde hay dos objetos flotantes, cada uno amarrado independientemente. A pesar de estos inconvenientes, también presenta una serie de ventajas para su uso. Una de ellas es que calcula las fuerzas de los amarres con gran precisión. Otra ventaja

fundamental es que se trata de un software de código libre. Para conocer más detalles sobre MoorDyn, ver el anexo *16.1. Descripción de MoorDyn.*

Una segunda opción es la librería **MooDy**, está desarrollada en Matlab por investigadores de la CUT (Chalmers University of Technology). Esta librería no es código abierto. Solamente ofrece los ejecutables de la librería y dos funciones para realizar un acoplamiento con un software externo. Las versiones disponibles son para Mac OSX y Linux. Esto supone una gran limitación, ya que uno de los objetivos de este trabajo es realizar un acoplamiento con DualSPHysics, tanto para Windows como para Linux. Esta limitación, plantea un gran problema debido a que no se tiene acceso al código fuente y no es posible adaptar la librería a las necesidades del acoplamiento con otros modelos.

Por todo esto, MooDy se descarta como solución al problema planteado y se elige a MoorDyn para realizar el acoplamiento con DualSPHysics. Sin embargo, MoorDyn presenta algunos problemas graves que deben mejorarse. Por ejemplo, problemas de programación, tales como errores en tiempo de ejecución al no existir un control de excepciones ni una validación de los datos introducidos por el usuario (P. Ej.: longitud máxima de unión, rangos máximos y mínimos de la elasticidad o tensión de la línea, entre otros). Está escrito en lenguaje C++, pero no realiza una programación orientada a objetos lo que implica que esté limitado a simular únicamente amarres de un solo objeto flotante. Por lo tanto, es necesario re-implementar MoorDyn, manteniendo las ecuaciones físicas ya implementadas, pero realizando una programación orientada a objetos e incluyendo un control de excepciones. Además de corregir estos errores, se realizará el acoplamiento con DualSPHysics. La refactorización mantendrá los nombres de variables, funciones y archivos de MoorDyn con el fin de favorecer el mantenimiento de la nueva implementación por parte de los desarrolladores originales de **MoorDyn**. La nueva librería se llamará **MoorDyn+**.

2. Objetivos

El problema que se va a tratar, es la re-implementación de una librería de amarres para objetos flotantes y su acoplamiento con el modelo DualSPHysics. Para ello se han definido una serie de objetivos necesarios para darle solución al problema planteado.

Objetivo 1. El objetivo principal de este trabajo es la re-implementación de una librería que resuelva la dinámica de un sistema de amarres para objetos flotantes. La librería debe calcular la fuerza ejercida por los amarres en los puntos de conexión con los objetos flotantes.

Objetivo 2. La librería resultante, tiene que ser multiplataforma. Concretamente, desarrollada para su ejecución en plataformas con Windows y Linux.

Objetivo 3. La librería también deberá funcionar como un módulo independiente, donde simula un movimiento predefinido de las conexiones móviles de cada amarre. Esto quiere decir que se variarán las posiciones de las conexiones móviles de la línea de amarre para simular un movimiento.

Objetivo 4. La librería deberá funcionar acoplada a un software externo. En este caso, la implementación está orientada al acoplamiento con el software DualSPHysics. MoorDyn+ debe simular el comportamiento de los amarres y calcular la fuerza que estos ejercen sobre los objetos flotantes, mientras que el software externo debe simular el comportamiento de los objetos flotantes en función de los valores de fuerza proporcionados por MoorDyn+.

Objetivo 5. La librería tiene que seguir el paradigma de programación orientada a objetos e implementarse en el lenguaje de programación C++.

Objetivo 6. Permitirá la definición de múltiples líneas de amarre para múltiples objetos flotantes. Cada conjunto de amarres asociados a un objeto flotante se comportará de forma independiente para facilitar la realización de experimentos más complejos.

Objetivo 7. Definir un fichero de entrada de tipo XML que facilite la definición del sistema de amarres y permita su integración en los propios ficheros XML de configuración de DualSPHysics.

Objetivo 8. Introducir un control de excepciones. Este control de excepciones se encargará de comprobar que los cálculos realizados durante la ejecución sean correctos. Lanzará una excepción si se detecta algún problema y finalizará la ejecución del programa. También se usará en la validación inicial de los datos introducidos por el usuario.

Objetivo 9. La salida de información deberá estar en formato CSV, ya que es un formato compatible con cualquier herramienta de visualización y análisis de datos.

3. Resumen de la solución propuesta

3.1. Metodología

La metodología de desarrollo software utilizada en este trabajo es el modelo Iterativo e Incremental [4]. Se ha optado por esta metodología de desarrollo debido a que, al ser un proyecto de investigación, no todas las funcionalidades o requisitos estaban completamente definidos desde un principio. Este modelo está compuesto por:

- Etapa de inicialización
- Etapa de iteración
- Lista de control de proyecto

La etapa de inicialización se ha compuesto por las fases de análisis y diseño. En la fase de análisis se obtendrá el análisis de requisitos inicial y en la de diseño, el diseño estático del software y arquitectura del sistema.

En la etapa de iteración se definen las fases de implementación y finalización [4]. En la fase de implementación, cada iteración añade una nueva funcionalidad al sistema. En esta fase, cada iteración está compuesta por las siguientes tareas:

- Análisis de requisitos. Se realizará el análisis de requisitos necesario para cada una de las iteraciones de esta etapa que complementará al realizado en la fase de análisis.
- Diseño de la nueva funcionalidad.
- Implementación de la funcionalidad.
- Integración de la funcionalidad.
- Validación del sistema.

La lista de control de proyecto servirá para guiar el proyecto. Gracias a ella, se conocerán las tareas que se van finalizando. También se incluirán nuevas funcionalidades a implementar que puedan surgir a lo largo del proyecto.

3.2. Solución propuesta

En este apartado se define la solución técnica y el resumen de la aplicación de la metodología. En la Figura 3-1, se puede observar cómo será la estructura de la solución propuesta, dónde las partes sombreadas de azul serán las que se desarrollarán o re-implementarán a lo largo de este trabajo.

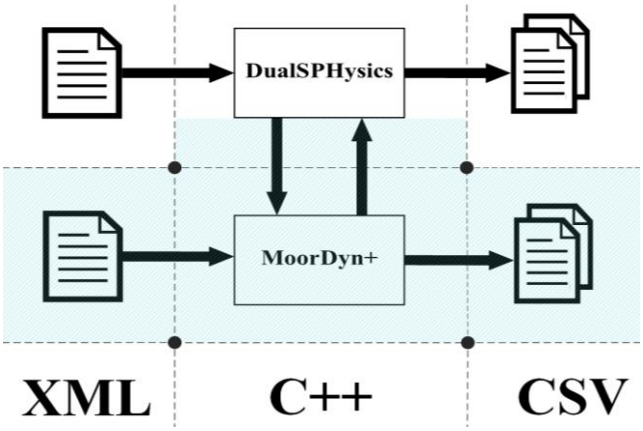


Figura 3-1: Solución propuesta

Para llevar a cabo la re-implementación de la librería, se utilizará el lenguaje de programación C++ 11, lo que cumpliría el *Objetivo 5*. Además, desde un principio, se implementará como una librería multiplataforma (Linux y Windows) para cumplir el *Objetivo 2*. Se compilará con Microsoft Visual C++ 2015 para versiones de Windows y con g++ 7.3.0 para Linux.

En cuanto a la aplicación de la metodología, se hará un resumen del contenido de cada fase que se ha planificado. A lo largo de esta sección, se hará mención a fases e iteraciones. Esta información está reflejada en la *Tabla 1* del apartado *4.1. Planificación*.

En la fase de análisis se hará un estudio del problema y un análisis de las herramientas existentes que podrían dar solución a dicho problema. Esta fase dará lugar a los apartados *1. Introducción* y *2. Objetivos*, dónde se desglosará el problema planteado y los objetivos necesarios para solventarlo. Una vez elegida la solución, se realizará la planificación del trabajo que se presenta en el apartado *4.1. Planificación.* Todas estas actividades se llevarán a cabo en la *Iteración 0*.

En la fase de diseño se definirán las partes y la estructura principal de la librería que se irá refinando a medida que se agreguen los incrementos de la posterior fase de implementación. La documentación relativa al diseño, tanto el diseño inicial como los que se vayan realizando con cada incremento, se presenta en el apartado *8. Diseño del software*. Se ha planificado que esta fase este compuesta por la *Iteración 1 (Diseño de clases)* y la *Iteración 2 (Arquitectura del sistema)*.

En la fase de implementación se llevará a cabo todo lo relevante a la codificación de la librería. Esta fase estará compuesta por seis iteraciones, enumeradas desde la *Iteración 3* hasta la *Iteración 8*.

A continuación, se hace un resumen de las iteraciones que se llevarán a cabo en esta fase:

Iteración 3: Entrada de datos: En esta iteración se definirá el fichero de entrada con todos los parámetros necesarios. Este fichero será con formato XML, cumpliendo así, el *Objetivo 7*. Esta iteración dará como resultado una plantilla del fichero XML. También se implementará la funcionalidad que permita leer el fichero e instanciar los objetos necesarios.

Iteración 4: Validación de entrada de datos. Se realizará la inicialización de los objetos creados y se comprobará que los datos introducidos por el usuario estén dentro de unos rangos establecidos.

Iteración 5: Inicialización del sistema. En esta iteración se codificarán las funciones encargadas de inicializar el sistema. La implementación de esta iteración, estará enfocada a manejar cada conjunto de amarres independientemente, mediante el uso de la programación orientada a objetos. Esto cumplirá el *Objetivo 6*. Además, se introducirá un control de excepciones para detectar valores erróneos (+infinito, -infinito, NaN) a la hora de realizar cálculos, cumpliendo también, el *Objetivo 8*.

Iteración 6: Cálculos de fuerzas. Esta iteración se corresponderá con la funcionalidad más importante. Será la encargada de integrar las funciones donde se realizan los cálculos necesarios para aplicar las fuerzas sobre los objetos flotantes. También se implementarán una serie de funciones que permitan generar movimientos predefinidos permitiendo el uso de la librería como un módulo independiente *(Objetivo 3)*. En este punto, la librería será totalmente funcional, por lo que cumpliría el *Objetivo 1*, es decir, el objetivo principal de este trabajo.

Iteración 7: Definición de la salida de datos. Se diseñará e implementará la salida de datos del sistema. Esta salida de datos generará ficheros en formato CSV, lo que cumpliría el *Objetivo 9*.

Iteración 8: Acoplamiento con el software DualSPHysics. Se realizará un acoplamiento con un software externo para simular de objetos flotantes amarrados. Para llevarlo a cabo, será necesario desarrollar una interfaz que permita la comunicación entre ambos. Los desarrolladores de DualSPHysics serán los encargados de realizar los cambios pertinentes en DualSPHysics para que el intercambio de información sea correcto. Al finalizar esta iteración, se cumplirá el *Objetivo 4*.

La fase de finalización estará compuesta por dos iteraciones. En la *Iteración 9*, se realizarán las pruebas globales de la librería que servirán para validar el sistema. Para llevarlas a cabo, se utilizarán datos experimentales de un objeto flotante amarrado al suelo que se compararán con los resultados obtenidos por la librería. En la *Iteración 10*, se terminarán y perfilarán detalles de la documentación necesaria del trabajo.

Para cada incremento de la etapa de iteración se llevará a cabo la documentación de la funcionalidad añadida.

La representación del sistema se realizará utilizando el lenguaje unificado de modelado (UML). Se ha elegido UML, ya que está diseñado para visualizar, especificar, construir y documentar software orientado a objetos [5].

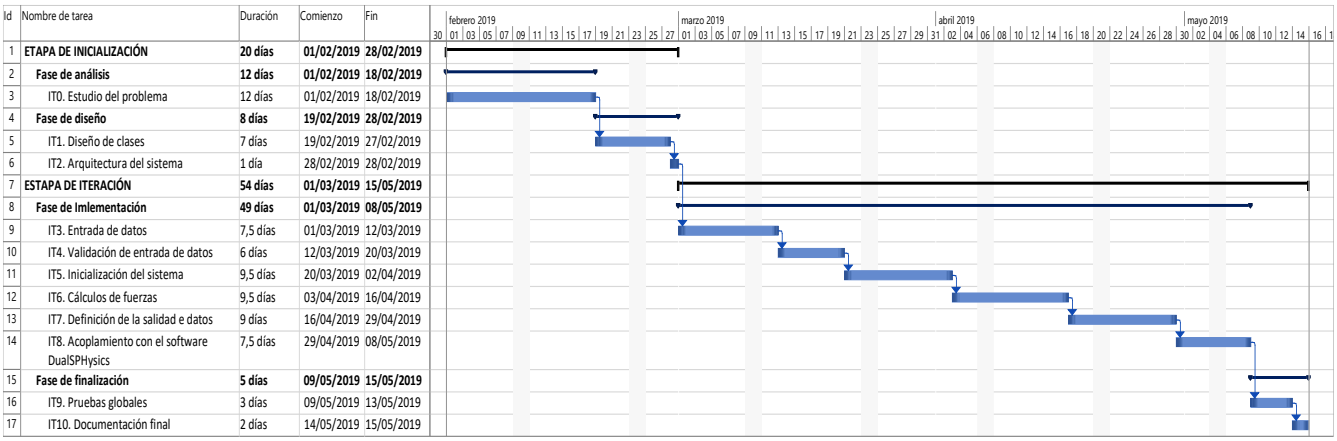
4. Planificación y seguimiento

4.1. Planificación

En la Tabla 1 se puede observar la planificación inicial que se ha elaborado para llevar a cabo el trabajo. Se dividen las fases con sus respectivas iteraciones y el tiempo estimado de duración de estas. Cabe señalar que se planifica dedicar 4 horas de trabajo cada día. La Figura 4-1 muestra la fecha inicial y final de cada una de las fases e iteraciones. Se establece el 1 de febrero de 2019 como fecha de arranque del proyecto y la fecha prevista de finalización es el 15 de mayo de 2019.

ETAPAS	Tiempo estimado (horas)
ETAPA INICIALIZACIÓN	**80**
Fase de Análisis	48
Iteración 0. Estudio del problema.	48
Fase de Diseño	32
Iteración 1. Diseño de clases.	28
Iteración 2. Arquitectura del sistema.	4
ETAPA ITERACIÓN	**216**
Fase de Implementación	196
Iteración 3. Entrada de datos.	30
Iteración 4. Validación de entrada de datos	24
Iteración 5. Inicialización del sistema.	38
Iteración 6. Cálculos de fuerzas.	38
Iteración 7. Definición de la salida de datos.	36
Iteración 8. Acoplamiento con el software DualSPHysics.	30
Fase de Finalización	20
Iteración 9. Pruebas Globales.	12
Iteración 10. Documentación final	8
LISTA DE CONTROL DE PROYECTO	**4**
Lista de control del proyecto	4
TOTAL	**300**

Tabla 1: Planificación



Id	Nombre de tarea	Duración	Comienzo	Fin
1	**ETAPA DE INICIALIZACIÓN**	**20 días**	**01/02/2019**	**28/02/2019**
2	**Fase de análisis**	**12 días**	**01/02/2019**	**18/02/2019**
3	IT0. Estudio del problema	12 días	01/02/2019	18/02/2019
4	**Fase de diseño**	**8 días**	**19/02/2019**	**28/02/2019**
5	IT1. Diseño de clases	7 días	19/02/2019	27/02/2019
6	IT2. Arquitectura del sistema	1 día	28/02/2019	28/02/2019
7	**ESTAPA DE ITERACIÓN**	**54 días**	**01/03/2019**	**15/05/2019**
8	**Fase de Imlementación**	**49 días**	**01/03/2019**	**08/05/2019**
9	IT3. Entrada de datos	7,5 días	01/03/2019	12/03/2019
10	IT4. Validación de entrada de datos	6 días	12/03/2019	20/03/2019
11	IT5. Inicialización del sistema	9,5 días	20/03/2019	02/04/2019
12	IT6. Cálculos de fuerzas	9,5 días	03/04/2019	16/04/2019
13	IT7. Definición de la salidad e datos	9 días	16/04/2019	29/04/2019
14	IT8. Acoplamiento con el software DualSPHysics	7,5 días	29/04/2019	08/05/2019
15	**Fase de finalización**	**5 días**	**09/05/2019**	**15/05/2019**
16	IT9. Pruebas globales	3 días	09/05/2019	13/05/2019
17	IT10. Documentación final	2 días	14/05/2019	15/05/2019

Figura 4-1: Diagrama de Gantt

4.2. Puntos críticos

A la hora de realizar la planificación, se han tenido en cuenta la *Iteración 5* y la *Iteración 6* como puntos críticos que podrían provocar posibles desviaciones. En ambas iteraciones se integrará la mayor parte de las ecuaciones que definen la física de los amarres. Debido a que los conocimientos sobre toda la física que hay no son lo suficientemente amplios, se considera que ambas iteraciones pueden suponer un punto crítico y por esta razón se le asignan más horas de las que en un primer momento podrían estimarse para su realización.

4.3. Ejecución

En la Tabla 2 se muestra la comparativa entre el tiempo estimado y el tiempo de ejecución. El trabajo presenta desviaciones en la *Iteración 6*, *Iteración 7* e *Iteración 8*, lo que provoca que no coincidan sus tiempos de ejecución y planificación.

ETAPAS	Tiempo estimado (horas)	Tiempo ejecución (horas)
ETAPA INICIALIZACIÓN	**80**	**80**
Fase de Análisis	48	48
Iteración 0. Estudio del problema.	48	48
Fase de Diseño	32	32
Iteración 1. Diseño de clases.	28	28
Iteración 2. Arquitectura del sistema.	4	4
ETAPA ITERACIÓN	**216**	**220**
Fase de Implementación	196	200
Iteración 3. Entrada de datos.	30	30
Iteración 4. Validación de entrada de datos	24	24
Iteración 5. Inicialización del sistema.	38	38
Iteración 6. Cálculos de fuerzas.	38	36
Iteración 7. Definición de la salida de datos.	36	32
Iteración 8. Acoplamiento con el software DualSPHysics.	30	40
Fase de Finalización	20	20
Iteración 9. Pruebas Globales.	12	12
Iteración 10. Documentación final	8	8
LISTA DE CONTROL DE PROYECTO	**4**	**4**
Lista de control del proyecto	4	4
TOTAL	**300**	**304**

Tabla 2: Tiempos de ejecución

4.4. Desviaciones

Se han necesitado 10 horas más para llevar a cabo la *Iteración 8*. Esto se debe a que no se había planificado el desarrollo de una serie de funciones que le permitiesen a DualSPHysics conocer el estado del sistema y poder generar archivos de VTK (The Visualization Toolkit) con la forma de los amarres. Estos archivos son necesarios para visualizar las líneas de los amarres mediante un software de visualización científica como puede ser ParaView. Ha llevado tanto tiempo porque ha sido necesario coordinarse con el equipo de desarrollo de DualSPHysics para definir y agregar esta serie de funciones. Estas funciones se han documentado en el anexo *16.2. Nuevas funciones añadidas*.

Por otra parte, para llevar a cabo la *Iteración 7*, no se han necesitado la totalidad de las horas que se han planificado. En un primer momento se estimaba que llevaría más tiempo la definición de la salida

de datos en función de las magnitudes y tipos de conexión elegidas por el usuario, pero se ha realizado en 4 horas menos.

En la *Iteración 6* se había planificado que se necesitarían 2 horas de pruebas para validar el incremento, pero a posteriori se ha decidido que no se realizarían tales pruebas. Esta decisión surge por la necesidad de realizar una comparativa entre los datos simulados por MoorDyn+ y datos experimentales. Únicamente se han obtenido datos experimentales de un caso con objeto flotante, por lo que no se ha podido probar y validar hasta haber realizado el acoplamiento con DualSPHysics. Esto se debe a que DualSPHysics es el encargado de realizar la simulación de los objetos flotantes a partir de las fuerzas ejercidas por los amarres de MoorDyn+. Por esta razón, ha llevado 2 horas menos la realización de esta iteración.

En definitiva, el proyecto se ha retrasado 4 horas en total sobre lo planificado debido a las variaciones en tiempo de ejecución de estas tres iteraciones.

5. Arquitectura

Durante la fase de diseño, en la *Iteración 2*, se ha obtenido la arquitectura que tendrá MoorDyn+. Se puede observar en la Figura 5-1. Para que se inicie el sistema es necesario lanzar un ejecutable (*launcher*). El ejecutable tiene que ser compilado para el Sistema Operativo (S.O.) anfitrión donde será lanzado (Windows o Linux). Una vez lanzado, MoorDyn+ funcionará como un módulo cerrado que se ejecuta en el Sistema Operativo anfitrión sin interacción con el usuario. Interaccionará con un fichero XML que será el archivo de entrada encargado de configurar el sistema. Finalmente, tenemos la parte de la salida de datos, compuesta por los ficheros CSV, donde se guardará toda la información relativa a los cálculos realizados.

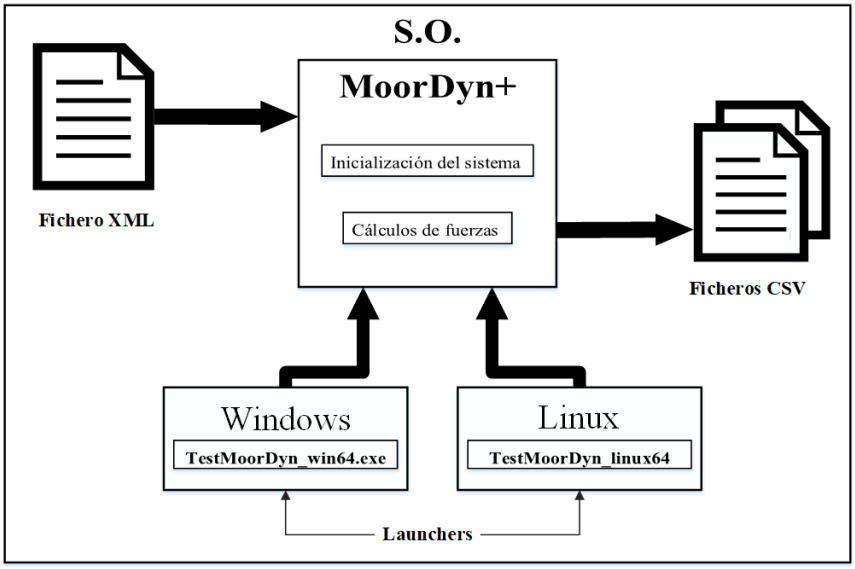


Figura 5-1: Arquitectura de MoorDyn+

Además de la arquitectura, se puede observar en el anexo *16.3. Acoplamiento a nivel de proceso* como se realizará la comunicación con DualSPHysics y como se ejecutarán ambos sistemas software a nivel de proceso.

6. Tecnologías e integración de productos de terceros

C++

Es un lenguaje de programación de propósito general, diseñado en 1979 por *Bjarne Stroustrup* [6]. La intención de su creación fue extender las características del lenguaje de programación C. Las características del lenguaje C++ permiten cuatro estilos de programación: programación procedural, abstracción de datos, programación orientada a objetos y programación genérica.

Este lenguaje tiene un alto rendimiento al permitir hacer llamadas directamente al Sistema Operativo, es multiplataforma y permite acceso directo a la memoria (controlado por el usuario). Tiene una integración directa con el lenguaje ensamblador, permitiendo escribir directamente en dicho lenguaje.

DualSPHysics también está implementada en C++. Por temas de compatibilidad y acoplamiento, se utilizará este leguaje para llevar a cabo la implementación. Además, permite la programación orientada a objetos, siendo uno de los objetivos de este trabajo.

A pesar de los años que tiene, es un lenguaje moderno y actualizado. C++17 es la versión más reciente (2017), reemplazando la anterior especificación C++14 del año 2014. Para este proyecto, se utilizará la versión C++14 por motivos de compatibilidad al utilizar Microsoft Visual C++ 15 como compilador para Windows.

XML

XML, cuyas siglas significan eXtensible Markup Language, es un meta-lenguaje de marcado similar a HTML. Es una adaptación del SGML (Standard Generalized Markup Language). Permite la organización y etiquetado de documentos. No es un lenguaje en sí mismo, sino un sistema que permite definir lenguajes de acuerdo a las necesidades de cada uno. El metalenguaje aparece como un estándar que estructura el intercambio de información entre diferentes plataformas. Ofrece la posibilidad de estructurar y representar datos. Es habitual que ciertos programas incluyan archivos de configuración con este formato. Debido a sus características, XML será muy útil para definir los parámetros de configuración de la librería.

GCC

GNU Compiler Collection (Colección de Compiladores GNU) [7]. Fue desarrollado por *Richard Stallman*, el fundador del Proyecto GNU.

GCC es una colección de compiladores GNU que incluye interfaces para C++, entre otros lenguajes. En concreto, g++ es un comando de compilación para programas escritos en lenguaje C++ especialmente en plataformas Unix. Aunque actualmente también da soporte a Windows. Se utilizará la versión GCC 7.3.0, lanzada en enero del 2018.

Make, también de GNU [8], es una herramienta de gestión de dependencias. Está diseñada para automatizar los pasos necesarios para la compilación de código fuente. Esta automatización se consigue mediante la definición de un fichero *make*, que por lo general se nombra *GNUmakefile*, *makefile* o *Makefile*. Estos son los nombres predeterminados, aunque se puede llamar como se desee especificándoselo explícitamente a make. Un archivo makefile es un fichero de texto ordinario con una sintaxis muy concreta que make puede entender. En el archivo *make*, define las relaciones entre

el código fuente, los archivos intermedios y los ejecutables. Se escriben las dependencias, objetivos y los comandos para compilar archivos de código fuente y generar un ejecutable o librería compilada.

Se utilizarán conjuntamente Make y GCC para la compilación del código fuente de la librería para plataformas Linux.

GDB

GDB o GNU Debugger es el depurador estándar para el compilador GNU [9]. Se utiliza en plataformas Unix y funciona para C++, entre otros lenguajes. Es software libre y ofrece la posibilidad de ver lo que sucede dentro de otro programa mientras se ejecuta, o ver las causas de un posible bloqueo. GDB puede realizar varias tareas para ayudar a detectar errores. Por ejemplo: iniciar el programa especificando cualquier factor que pueda afectar a su comportamiento, hacer que el programa se detenga en condiciones específicas, examinar lo sucedido cuando el programa se detiene en circunstancias imprevisibles, etc. Es una herramienta muy útil para detectar posibles pérdidas de memoria o accesos a memoria no permitidos. Se utilizará la versión GDB 8.2.1, lanzada en diciembre del 2018.

UML

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual de propósito general que se utiliza para especificar, visualizar, construir y documentar artefactos de un sistema software [5]. Captura decisiones y conocimiento sobre sistemas que deben ser construidos. Se usa para comprender, diseñar y controlar la información sobre dichos sistemas. UML capta la información sobre la estructura estática y el comportamiento dinámico del sistema. La estructura estática define los tipos de objetos importantes para un sistema y su implementación, así como las relaciones entre los objetos. El comportamiento dinámico define la historia de los objetos a lo largo del tiempo y la comunicación entre ellos para cumplir los objetivos. En este proyecto se hará uso de los diagramas de clases para representar la estructura estática del sistema y los diagramas de secuencia para representar el comportamiento dinámico.

Visual Studio

Visual Studio (VS) es un entorno de desarrollo integrado (IDE) para sistemas operativos Windows. Existen 3 IDEs (Community, Professional y Enterprise), dónde el VS Community es gratuito. Visual Studio ofrece herramientas como Microsoft Visual C++ que soporta el lenguaje de programación C++. Es un entorno de desarrollo que permite la creación de aplicaciones y librerías, ya que se pueden instalar compiladores para C++. También permite configurar el uso de librerías estáticas adicionales o librerías de enlace dinámico (DLL).

En este proyecto se utilizará la Versión Visual C++ 2015 Community, a pesar de que actualmente está en el mercado la versión 2017. Se optará por esta versión por motivos de compatibilidad, debido a que la última versión de DualSPHysics está compilada para Windows con Visual C++ 2015.

Visual Paradigm

Visual Paradigm (VP) es una herramienta software diseñada para que los equipos dedicados al desarrollo software modelen los sistemas de información y gestionen los procesos de desarrollo que van a llevar a cabo. Visual Paradigm es compatible con lenguajes de modelado como el Lenguaje Unificado de Modelado (UML). Visual Paradigm es una herramienta de pago, pero ofrece suscripciones mediante licencias académicas. Se utilizará la versión 15.1, de la cual se tiene una versión académica a pesar de que exista una versión más actual (15.2).

MoorDyn

MoorDyn [2] es la librería original de simulación de amarres seleccionada para resolver el problema planteado. Se mantienen todas las ecuaciones de física implementadas en MoorDyn puesto que los resultados numéricos han mostrado ser muy exactos y han sido validados con datos experimentales. Al igual que MoorDyn, la nueva implementación también usará las librerías de cálculo *kis_ftt.h*, *QSlines.h* y *Misc.h*. Estas librerías se usarán íntegramente sin refactorizar.

DualSPHysics

DualSPHysics [1] es un software de código abierto con el que se realizará el acoplamiento de la librería que se implementará en este trabajo. DualSPHysics, tiene implementadas las clases *JObject* y *JException*. Estas clases se usan para enviar excepciones de una forma más sencilla y personalizada. *JObject* tiene una serie de funciones que permiten lanzar excepciones de tipo *JException* con información sobre la clase y método que la lanzan, además del mensaje de error deseado. El uso de estas dos clases, permitirá un mismo control de excepciones por parte de ambos sistemas software.

7. Especificación y análisis de requisitos

La mayor parte de la especificación y análisis de requisitos se ha elaborado en la etapa de inicialización, más concretamente, en la fase de análisis, que corresponde con la *Iteración 0*. El resultado de esta fase se ha redactado en los apartados de *1. Introducción, 2. Objetivos, 3. Resumen de la solución propuesta, 4.1. Planificación, 4.2. Puntos críticos* y anexo *16.1. Descripción de MoorDyn*.

A pesar de ello, al seguir una metodología Iterativo Incremental [4], no todos los requisitos están definidos en esta etapa. A continuación, se muestra la especificación y análisis de requisitos que se ha llevado a cabo en cada una de las iteraciones que agregan nuevas funcionalidades, las pertenecientes a la etapa de iteración.

7.1. Iteración 3: Entrada de datos

La librería MoorDyn original, utilizaba un fichero de texto (*.txt)*, lo que provocaba errores por delimitación de campos y obligaba escribir cabeceras para dividir las secciones. Se pueden ver los detalles en el anexo *16.1. Descripción de MoorDyn*.

En esta iteración, se ha hecho el análisis relativo al fichero de configuración que usará la librería. Este fichero tendrá un formato XML, dado que es el *Objetivo 7*. El fichero XML contendrá una jerarquía de etiquetas equivalente, en cierta medida, a la jerarquía de clases que se obtuvo en la fase de diseño, apartado *8.1. Diseño del software estático*. Se puede observar la estructura básica del fichero XML en la Figura 7-1.

Se partirá de un nodo raíz que se llamará *moordyn*. Este nodo será el que englobe toda la configuración de la librería. La librería deberá ser capaz de reconocer el nodo raíz de configuración en cualquier parte del fichero, aunque forme parte de un fichero XML más amplio. Esta especificación se fundamenta en que está planificado hacer un acoplamiento con DualSPHysics (*Iteración 8*), de acuerdo al *Objetivo 4* y se puedan integrar ambos ficheros de entrada (el XML para DualSPHysics y el XML para MoorDyn+) en un único fichero XML.

Se hace un estudio de los parámetros necesarios para la definición del entorno (densidad del agua, profundidad, gravedad, …) y los parámetros globales del sistema para el cálculo (intervalo de tiempo de integración, tiempo de ejecución, …). Se define un bloque llamado *solverOptions* que albergará estos parámetros generales.

Se definen bloques para la configuración de los amarres, llamados *mooring*. Cada bloque *mooring* contiene la configuración relativa a un único objeto flotante y todas las líneas de amarre conectadas a él. Las etiquetas *mooring* podrán incluir un atributo “ref”. Este atributo será opcional y se usará como referencia para vincular los amarres definidos por MoorDyn+ y los objetos flotantes definidos por un software externo. Si no se incluye, internamente el programa le dará un identificador por orden de creación (0, 1, 2, ...). Dentro de *mooring* puede haber una etiqueta *depth*, ya que se estima que en un futuro se puedan crear amarres con distintas profundidades, ver apartado *14. Vías de trabajo futuro*. También se creará un bloque llamado *lineDefault*. Este bloque contendrá elementos como elasticidad, diámetro o masa en el aire que compartirán todas las líneas de amarre dentro de *mooring*. Se usará un bloque que contenga las líneas de amarre, llamado *line*. En él se definirán parámetros de longitud, número de segmentos en los que se divide la línea de amarre. Se definen también dos etiquetas para configurar los extremos de unión de la línea (tipo y posición). La primera de ellas será *fixconnection*, la cual será usada para crear el extremo fijo de la línea. La segunda será *vesselconnection* y representará el extremo conectado a un objeto flotante.

```
<moordyn>
    <solverOptions>
        <gravity value="X" comment="comentario"/>
        <timeMax value="X" comment="comentario"/>
        .
        .
        .
    </solverOptions>
    <mooring ref="X">
        <depth value="X" comment="comentario"/>
        <linedefault>
            <valiable value="X" comment="comentario"/>
            .
            .
            .
        </linedefault>
        <line>
            <length value="X" comment="comentario"/>
            <segments value="X" comment="comentario"/>
            <fixconnection x="X" y="Y" z="Z" />
            <vesselconnection  x="X" y="Y" z="Z" />
        </line>
        .
        .
        .

    </mooring>
    .
    .
    .
</moordyn>
```

Figura 7-1: Estructura básica del fichero XML

La plantilla XML se puede ver en el apartado *9. Gestión de datos e información*. De esta forma, se organizará mejor la información, disminuyendo los errores por parte del usuario. Cada etiqueta tiene un atributo *value* donde se introduce el valor deseado. A mayores se presenta un atributo llamado *comment* que contendrá su significado y posibles valores.

Se desarrollarán las siguientes funciones que realizarán la lectura del XML.

- ***LoadXml()*:** Esta función se encargará de buscar el nodo *moordyn* en el fichero de entrada.
- ***ReadXml()*:** Se ejecutará después de la función anterior y se encargará de leer el contenido del nodo que reciba ***LoadXml()***.

Para llevar a cabo la inicialización del programa, se implementará una función ***Run()*** en la clase *MoorDyn* que se encargará de dirigir la ejecución del programa.

Al finalizar esta tarea de especificación y análisis de requisitos, se ha realizado el diseño correspondiente con esta iteración. En este trabajo, se ha decidido mostrar unificados los diseños de la *Iteración 3* y de la *Iteración 4* debido a que la validación de datos de entrada *(Iteración 4)* está integrada dentro del proceso de entrada de datos *(Iteración 3)*. Se puede observar el diseño en el apartado *8.2.1. Iteración 3 y 4: Entrada de datos y validación*.

7.2. Iteración 4: Validación de entrada de datos

En MoorDyn, la librería original, no se llevaba a cabo ninguna comprobación de los parámetros introducidos por el usuario ni se mantiene una relación de equilibrio para garantizar el correcto funcionamiento del sistema. Esto da lugar a un comportamiento impredecible cuando los datos de entrada son erróneos, sin avisar al usuario.

En esta iteración se hace un análisis para validar los parámetros de entrada introducidos por el usuario y se especifican los valores por defecto en el bloque *solverOptions* para el correcto funcionamiento del sistema. Si alguna etiqueta opcional no existe, se le asignará el valor por defecto que le corresponda y si falta una etiqueta no opcional, se finalizará la ejecución lanzando una excepción explicando el motivo.

También se implementará un control de excepciones que comprobarán si los datos están dentro de ciertos rangos, por ejemplo, que la longitud de la línea sea mayor o igual que la distancia entre los puntos de conexión o que la profundidad sea mayor que la posición de la conexión al objeto flotante. Para controlar los rangos de los parámetros de entrada se usará una función llamada ***CheckInputParameters()*** que lanzará una excepción si detecta alguna inconsistencia o ***CheckReferences()*** que comprobará si hay duplicidades de identificadores en los amarres.

A mayores se hace un estudio sobre la relación entre varios parámetros que debe cumplirse para mantener un equilibrio en el sistema. Los parámetros en cuestión se especifican en la *Tabla 3*.

Notación	Definición	Representación en XML
N	Numero de segmentos	segments
L	Longitud de la línea	length
E	Rigidez de la línea	[calculado internamente] *Se realiza internamente a partir del valor de EA (introducido en el fichero XML) y otros parámetros.
W	Peso por unidad de línea	[calculado internamente]
dtM	Intervalo de tiempo usado en la integración	dtM

Tabla 3: Parámetros para el equilibrio del sistema

A partir de los parámetros de la *Tabla 3*, podemos obtener el valor de la frecuencia natural (Fn) para cada segmento en la ecuación (1).

$$Fn = \frac{2*N}{L} * \sqrt{E * W}\ (\frac{rad}{s}) \quad (1)$$

La relación para el equilibrio tiene que ser la frecuencia natural (*Fn*) en Hertzios, 10 veces menor que 1 / dtM.

$$\frac{Fn[Hz]}{10} < \frac{1}{dtM} \quad (2)$$

Dado que MoorDyn+ usará la misma física para el cálculo de fuerzas implementada en MoorDyn, el método usado para el diseño de las líneas será mediante el Método de Aproximación Catenaria [10]. El problema de equilibrio es no-lineal y debe resolverse de una manera iterativa, mediante pruebas sucesivas.

El valor de dtM será usado para el cálculo a base de iteración. A menor dtM, garantizamos que el sistema funcione manteniendo la relación de equilibrio, ecuación (2). El problema radica en que el número de iteraciones a realizar, se obtiene de la operación 1 / dtM. A menor dtM, el número de iteraciones será mayor, lo que supone que el programa tarde más tiempo en realizar los cálculos. Se desarrollará una función que recalcule computacionalmente el valor óptimo del dtM que garantice la relación de equilibrio, llamada ***CheckDtMRelationship()***. A pesar de que será desarrollada en esta

iteración, su funcionalidad no será probada hasta que se realice la integración y pruebas de la *Iteración 5*. Esto se debe a que en este punto, no se conocerán los parámetros calculados internamente (ver *Tabla 3*) para aplicar la ecuación (1) y poder garantizar la ecuación (2).

Al finalizar esta tarea de especificación y análisis de requisitos (al igual que en la *Iteración 3*), se ha realizado el diseño correspondiente con esta iteración. Se puede observar en la Figura 8-3 en el apartado *8.2.1. Iteración 3 y 4: Entrada de datos y validación*.

7.3. Iteración 5: Inicialización del sistema

En esta iteración se analizan las funciones necesarias para inicializar todo el sistema en el instante inicial (t = 0). Esta parte será crucial, puesto que hay que definir las funciones para cada amarre de forma que cada uno de ellos funcione independientemente. Esto favorece que se pueda realizar la simulación de amarres para múltiples objetos flotantes *(Objetivo 6)*. En este punto, cada conjunto de amarres sobre un objeto flotante, tendrá sus funciones y atributos propios. En la librería anterior, MoorDyn, toda la información de los amarres estaba contenida en un único fichero *(MoorDyn.cpp)*.

Se usarán funciones llamadas ***Setup()*** en todos los objetos que necesiten dicha inicialización. También será necesario que se desarrolle una función llamada ***InitializeSystem(positions, velocities)***. Esta función se usará para sumarle posiciones y velocidades iniciales a cada uno de los extremos de tipo *vessel* (el extremo del amarre conectado al objeto flotante).

En este apartado se implementará una función llamada ***DoDynamicRelaxIC()***. Esta función será la encargada de aplicar el Método de Aproximación Catenaria [10] haciendo llamadas a las funciones ***Rk2()***, ***RHSmaster()*** y ***DoRHS()***. Estas tres últimas funciones serán las encargadas de realizar los cálculos de fuerzas y se tomarán íntegramente del código original de MoorDyn, pero se le añadirá un control de excepciones. De esta forma, se cumplirá con el *Objetivo 8* para evitar que continúe la ejecución cuando se produzca algún error (P. Ej.: intentos de divisiones por 0, operaciones con valores +infinito, -infinito, NaN, etc.).

En este punto, se obtendrá un código con mayor cohesión entre las distintas clases. Cada objeto tendrá las funciones necesarias para representar su comportamiento, a diferencia de la versión original donde la mayor parte de las funcionalidades se implementaban en una sola función del fichero *MoorDyn.cpp*.

Al finalizar esta tarea de especificación y análisis de requisitos, se ha realizado el diseño correspondiente con esta iteración. Se puede observar en el apartado *8.2.2. Iteración 5: Inicialización del sistema*.

7.4. Iteración 6: Cálculos de fuerzas

En esta iteración se realiza un estudio de las funciones necesarias para llevar a cabo el cálculo de las fuerzas ejercidas por los amarres. Estas funciones serán llamadas en cada instante de tiempo de la simulación para actualizar el estado de los amarres y aplicar las fuerzas sobre los objetos flotantes. Se desarrollará una función para llevar a cabo el cálculo de los amarres en cada instante de tiempo. A la función en cuestión, se le incorporará un parámetro que permita identificar el objeto flotante para el que se desean calcular las fuerzas de sus amarres. Este identificador corresponderá con el atributo "ref". Ver la Figura 7-2. Puesto que "ref" es opcional, si no se introduce, se sobrescribirá con el valor correspondiente al orden de creación (0, 1, 2, …).

```
<mooring ref="10">
```

Figura 7-2: Atributo ref que identifica cada objeto flotante

El uso de "ref" es necesario para acoplar la librería a un software externo, *Objetivo 4*. Dicho acoplamiento se ha planificado que se llevará a cabo en la *Iteración 8* por lo que se cree conveniente implementarlo en esta iteración. La función se llamará, al igual que en MoorDyn, ***FairleadsCalc()***. Su cabecera se puede observar en la Figura 7-3.

```
//==============================================================================
/// This function now handles the assignment of fairlead boundary conditions,
/// time stepping, and collection of resulting forces at fairleads.
/// It can also be called externally for fairlead-centric coupling.
//==============================================================================
int MoorDyn::FairleadsCalc(const unsigned ftid, double **rFairIn, double **rdFairIn,
                           double ** fFairIn, double* t_in, double *dt_in)
```

Figura 7-3: Función de cálculo de fuerzas

Se necesitará la posición (***rFairIn***) y velocidad (***rdFairIn***). Ambos serán vectores de tres posiciones con los valores para X, Y, Z. El tiempo actual de ejecución (***t_in***) y el intervalo de tiempo (***dt_in***). El identificador del objeto (***ftid***) y una variable donde MoorDyn+ escribirá las fuerzas resultantes (***fFairIn***).

Los cálculos de física realizados en esta función, serán los que tiene implementados MoorDyn. También se implementarán dos funciones para que MoorDyn+ pueda simular el movimiento de un objeto flotante y obtener las fuerzas sobre dicho objeto de forma independiente, sin acoplarse a otro modelo. Se llamarán ***InitPos()*** y ***UpdatePos()***. Estas funciones serán las encargadas de simular la posición del objeto en cada instante de tiempo y permitirán simular el movimiento de los objetos como un módulo independiente, *Objetivo 3*.

Al finalizar esta tarea de especificación y análisis de requisitos, se ha realizado el diseño correspondiente con esta iteración. Se puede observar en el apartado *8.2.3. Iteración 6: Cálculos de fuerzas*.

7.5. Iteración 7: Definición de la salida de datos

Se analizan los tipos de salida de datos que se necesitan. Las magnitudes requeridas son: tensión, velocidades y posiciones de las líneas de cada amarre. El formato de la salida será un fichero CSV *(Objetivo 9)*. Se ha elegido este formato por ser compatible con cualquier herramienta de visualización y análisis de datos.

Existirán dos tipos de salida de datos:

- **Tipo 1**. Se encargará de almacenar las magnitudes elegidas que existen en cada nodo de una línea de amarre. El fichero que almacenará estos datos será *MooringXXLineYY.csv*, donde XX representarán el valor de "ref" almacenado en el programa e YY representarán el identificador de la línea de amarre que corresponde con el orden de creación.

- **Tipo 2**. Se encargará de almacenar las magnitudes que presenta la línea en sus extremos. El fichero que almacenará estos datos será *MooringXX.csv*, donde XX representarán el valor de "ref" almacenado en el programa. El fichero contendrá las magnitudes requeridas para todas las líneas de amarre.

Para poder llevar a cabo esta implementación se analizan las partes necesarias a introducir en el fichero XML. Para la salida de **tipo 1**, simplemente se utilizará una etiqueta llamada *outputsFlags*, donde se seleccionará el tipo de magnitud que se desee para cada nodo de cada línea. Ver la Figura 7-4.

```
<linedefault>
        <ea value="2.9e1" comment="stiffness(N)" />
        <diameter value ="3.656E-3"comment="(m)" />
        <massDenInAir value="0.0607" comment="massDenInAirDenInAir (kg/m)" />
        <ouputsFlags value="pv" comment="Node output properties.(default=-) [-=None, p=positions, v=velocities, t=tension]" />
</linedefault>
```

Figura 7-4: Configuración de la salida de datos por nodo de línea

La salida de **tipo 2** será la más utilizada en posteriores análisis de datos, ya que la información más importante de las líneas es la que está en sus extremos. Se creará un bloque llamado *output*, contenido dentro del bloque *Mooring*. Se puede ver su estructura en la Figura 7-5. Dentro de *output*, estarán las etiquetas *time* que permitirá elegir el inicio y fin de la grabación de datos, además de la frecuencia de grabación de datos *(dtOut)*. También tendrá una etiqueta por cada magnitud resultante del análisis anterior. Para la tensión *(tension)*, velocidades *(velocity)* y posiciones *(position)*. En todas ellas se podrá especificar el extremo del que se desean guardar los datos: extremo fijo *(fixed)*, extremo móvil *(vessel)* o todos *(all)*.

```
<output>
  <time startTime="0" endTime="10" dtOut="0.01" comment="Default [startTime= 0; endTime= 0; dtOut=0.01]"/>
  <tension type="all" comment="type=[fixed|vessel|all].  (default: type=all)."/>
  <position type="all" comment="type=[fixed|vessel|all]. (default: type=all)." />
  <velocity type="all" comment="type=[fixed|vessel|all]. (default: type=all)." />
</output>
```

Figura 7-5: Configuración de la salida de datos de las líneas

Al finalizar esta tarea de especificación y análisis de requisitos, se ha realizado el diseño correspondiente con esta iteración. Se puede observar en el apartado *8.2.4. Iteración 7: Definición de la salida de datos*.

7.6. Iteración 8: Acoplamiento con el software DualSPHysics

En esta iteración se hace un análisis para llevar a cabo un acoplamiento con DualSPHysics [1]. De esta forma se cumplirá el *Objetivo 4*. Se prevé que serán necesarias tres funciones para llevarlo a cabo. Una de ellas fue desarrollada en la *Iteración 6*, ver la Figura 7-3. A mayores, se necesitarán una función de inicialización, llamada ***LinesInit()***, y otra de finalización de la comunicación, ***LinesClose()***. La función de inicialización deberá tener la información relativa a las condiciones iniciales del entorno de los objetos flotantes (posiciones y velocidades). El software externo será el encargado de dar la información de la ubicación del fichero de entrada y del nodo a partir del cual estará definida la configuración de MoorDyn+. De esta forma, se podrán integrar los dos ficheros de configuración XML (el de DualSPHysics y el de MoorDyn+).

DualSPHysics será el encargado de definir los objetos flotantes a los que les asignará un identificador. Este identificador deberá coincidir con el atributo "ref" que será asignado a cada amarre en la configuración de MoorDyn+, ver la Figura 7-2. Por ello, será necesario que el software externo envíe dichos identificadores a MoorDyn+. MoorDyn+ tendrá que comprobar si los identificadores recibidos se corresponden con los amarres que fueron definidos, mediante una función que se llamará ***CheckFtIds()***.

También se considera necesario que el software externo envíe la ruta para almacenar la salida de información. La definición de las funciones necesarias para llevar a cabo el acoplamiento se puede ver en la Figura 7-6.

```
//==============================================================================
///Function to Init the system. Also is used when there is a coupling with external software
//==============================================================================
int MoorDyn::LinesInit(double X[], double XD[], const std::string filexml, const std::string nodexml,
                       const char *dir, const unsigned ftmkbound[], const unsigned numFt)

//==============================================================================
/// This function now handles the assignment of fairlead boundary conditions, time stepping,
/// and collection of resulting forces at fairleads. It can also be called externally for
/// fairlead-centric coupling.
//==============================================================================
int MoorDyn::FairleadsCalc(const unsigned ftid, double **rFairIn, double **rdFairIn,
                           double ** fFairIn, double* t_in, double *dt_in)

//==============================================================================
/// Function for finish the program and free the memory
//==============================================================================
int MoorDyn::LinesClose()
```

Figura 7-6: Funciones de MoorDyn+ para el acoplamiento con otros modelos

La intención es reducir el acoplamiento con la clase *MoorDyn*, por ello en vez de acoplar DualSPHysics directamente con las funciones de la Figura 7-6, se desarrollará un nuevo fichero que actúe como interfaz entre ambos. El fichero se llamará *Moorings*. Las funciones que se desarrollarán se pueden observar en la Figura 7-7.

```
//==============================================================================
/// Initializes MoorDyn and returns true in case of error.
//==============================================================================
bool MoorDyn_LinesInit(const std::string filexml, const std::string nodexml,
                       const std::string dirout, const unsigned numFt,
                       const unsigned ftmkbound[], const tdouble3 vellin[],
                       const tdouble3 velang[]);

//==============================================================================
/// Deallocates the variables used by MoorDyn (returns true in case of error).
//==============================================================================
bool MoorDyn_LinesClose();

//==============================================================================
/// Force calculation of moorings by MoorDyn (returns true in case of error).
//==============================================================================
bool MoorDyn_FairleadsCalc(unsigned ftid, double **fairpos, double **fairvel,
                           double ** fairforce, double time, double stepTime);
```

Figura 7-7: Interfaz de comunicación con MoorDyn+

A continuación, se define la funcionalidad que implementará cada función del fichero *Moorings* y sus parámetros.

- **MoorDyn_LinesInit:** Esta función será la utilizada para inicializar el sistema de amarres. En ella se enviará la siguiente información:
    - El fichero XML del que leerá la configuración (***filexml***).
    - El nodo XML por el que empezará a leer (***nodexml***).
    - El directorio de salida (***dirout***).
    - El número de objetos flotantes que se desea amarrar (***numFt***).
    - Array con los identificadores de los objetos flotantes(***ftmkbound***).
    - Velocidades lineales en el instante inicial (***vellin***).
    - Velocidades angulares en el instante inicial (***vellan***).
- **MoorDyn_FairleadsCalc:** Esta función se usará para actualizar los valores de fuerza ejercidos sobre el objeto flotante en cada instante de tiempo. Será llamada para cada amarre enlazado con un objeto flotante. Los parámetros de entrada serán los siguientes:
    - El identificador del objeto flotante actual (***ftid***).
    - Posición actual del objeto (***fairpos***).
    - Velocidad actual del objeto (***fairvel***).
    - Un array donde MoorDyn+ escribirá la fuerza calculada para el objeto en función de la posición y velocidad, entre otros factores (***fairforce***).
    - El instante de tiempo actual (***time***).
    - El intervalo de tiempo de cálculo (***stepTime***).
- **MoorDyn_LinesClose:** Esta función será usada para liberar la memoria, destruir los objetos creados y cerrar los ficheros de salida.

Al finalizar esta tarea de especificación y análisis de requisitos, se ha realizado el diseño correspondiente con esta iteración. Se puede observar en el apartado *8.2.5 Iteración 8: Acoplamiento con el software DualSPHysics*.

8. Diseño del software

En este apartado se incluyen los diagramas que representan el diseño del software estático y dinámico de MoorDyn+. En dichos diagramas no se muestran los *getters* ni *setters*, ni todas las funciones y atributos de los objetos debido a su extensión. Solamente se muestran los más representativos y los que aportan mayor funcionalidad. En la fase de diseño, *Iteración 1*, únicamente se realiza el diseño del software estático. El diseño del software dinámico es el resultado del diseño de cada una de las iteraciones de la fase de implementación.

8.1. Diseño del software estático

En este apartado se hace uso del diagrama de clases (artefacto de UML) para representar la vista estática del sistema [5]. En la Figura 8-1 se muestra la vista estática del software de la librería original, MoorDyn.

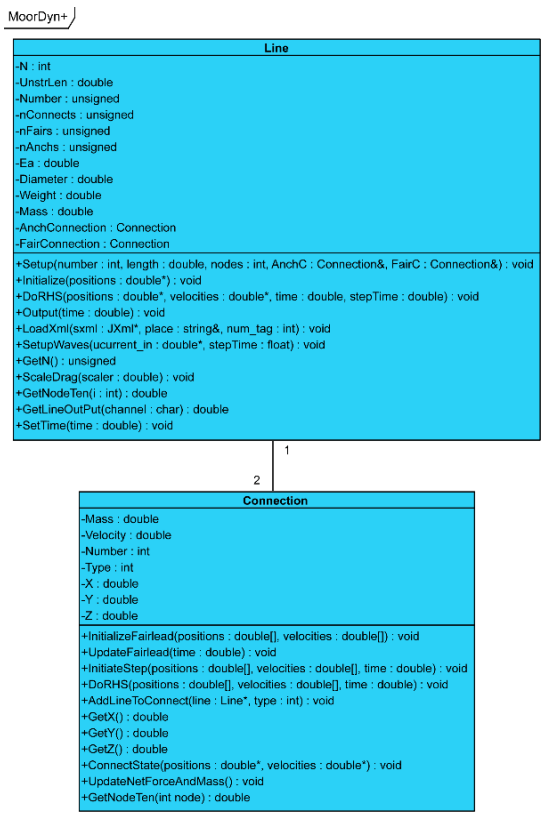


Figura 8-1: Diagrama de clases de MoorDyn

Cabe destacar que MoorDyn, aunque solamente hace uso de dos clases, implementa un fichero *MoorDyn.cpp* que contiene la lógica de la librería, maneja toda la información de los amarres y se comunica con las clases *Line* y *Connection* realizando las tareas pertinentes para el cálculo de fuerzas.

Para llevar a cabo este trabajo, se ha rediseñado la estructura de la librería, dando lugar a MoorDyn+. Se puede ver en la Figura 8-2. Se ha decidido que cada objeto flotante estará ligado a un amarre *(mooring)* y cada amarre estará compuesto por un conjunto de líneas de amarre *(line)*. Cada línea de amarre a su vez, contará con dos conexiones *(connection)*. De esta forma, cuando se haga mención a *mooring*, se hace referencia al conjunto de líneas de amarre que están conectadas a un objeto flotante.

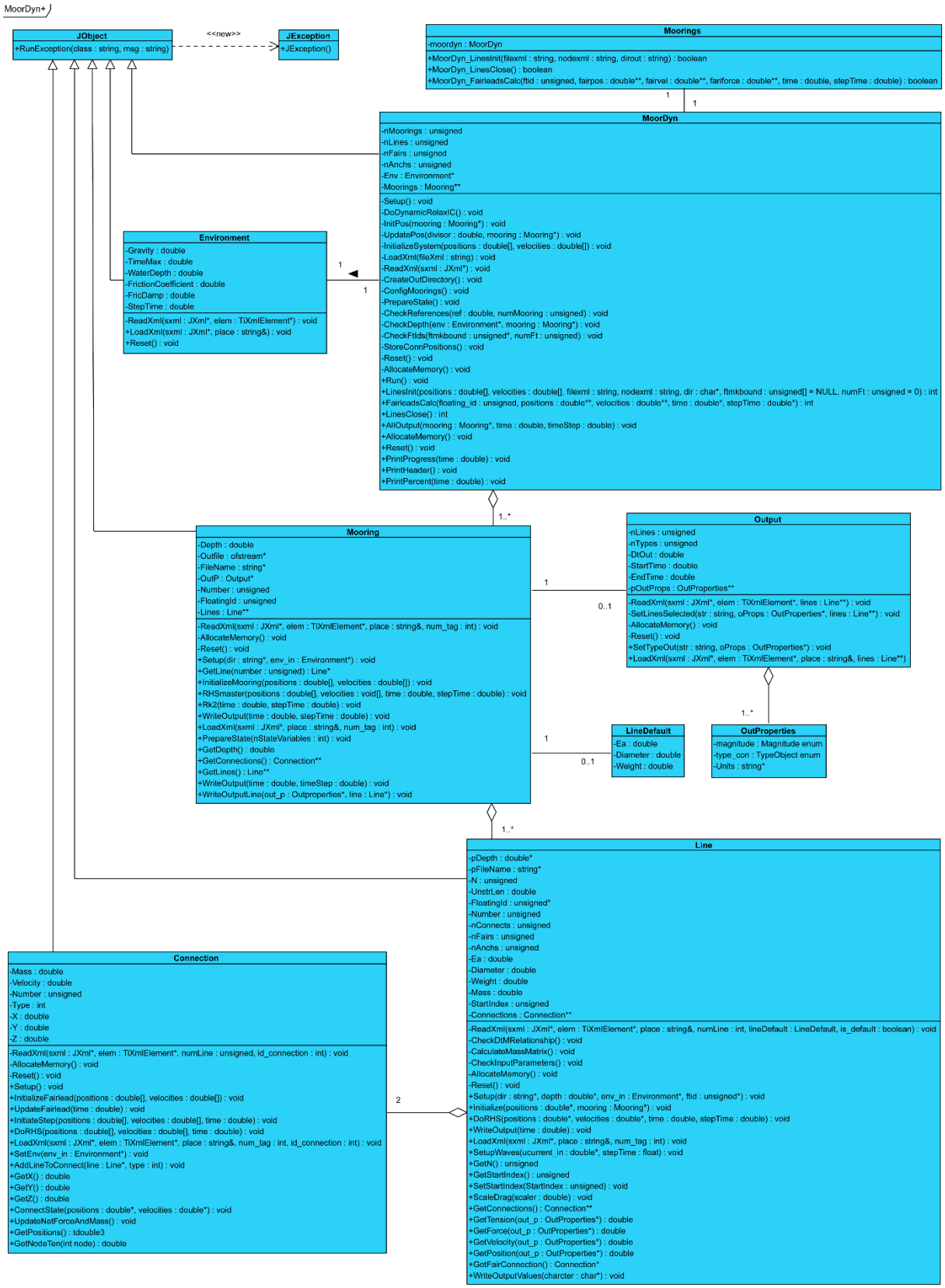


Figura 8-2: Diagrama de clases de MoorDyn+

8.2. Diseño del software dinámico

En este apartado se muestran los diagramas de secuencia que representan la vista dinámica del sistema [5]. En los diagramas solamente se representa el escenario de éxito. Esto se debe a que el sistema se ejecutará correctamente siempre y cuando no se produzca ninguna excepción. En caso de producirse una excepción, se abortará la ejecución del programa. Se puede ver un listado del control de excepciones que se ha implementado en el anexo *16.4. Listado del control de excepciones*.

Para representar el diseño dinámico de las iteraciones *Iteración 3* e *Iteración 4*, se ha decidido que se unificarán en un único diagrama de secuencia. Esto se debe a que las funciones desarrolladas para la validación de datos *(Iteración 4)*, se integrarán dentro del proceso de entrada de datos a partir del fichero de configuración XML *(Iteración 3)*.

En el diagrama de secuencia de la Figura 8-3, los objetos se muestran en la línea de vida en el momento de su creación (mediante *create*), a diferencia de los demás diagramas dónde todos los objetos ya están creados, por lo que se representan en la parte superior.

8.2.1. Iteración 3 y 4: Entrada de datos y validación

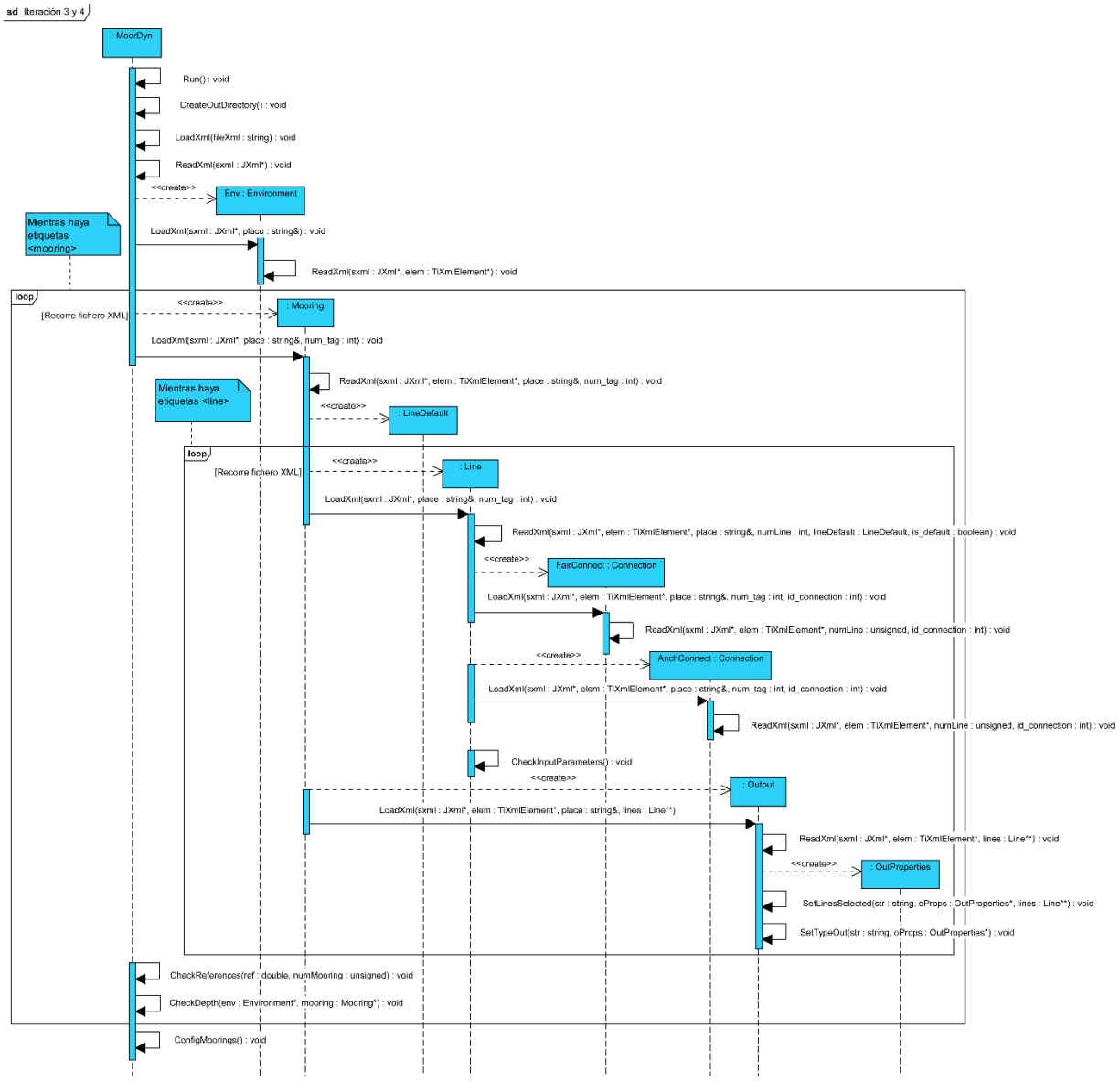


Figura 8-3: Diagrama de secuencia de entrada y validación de datos

8.2.2. Iteración 5: Inicialización del sistema

Para esta iteración, dada la extensión del diagrama de secuencia resultante, se devide en cuatro bloques ordenados por orden de ejecución del programa. Al finalizar cada uno de los bloques, continuará el siguiente.

En la Figura 8-7, se ha representado un *package* de color azul, debido a que el bloque de código incluido en él, será repetido en el diagrama de secuencia de la Figura 8-8.

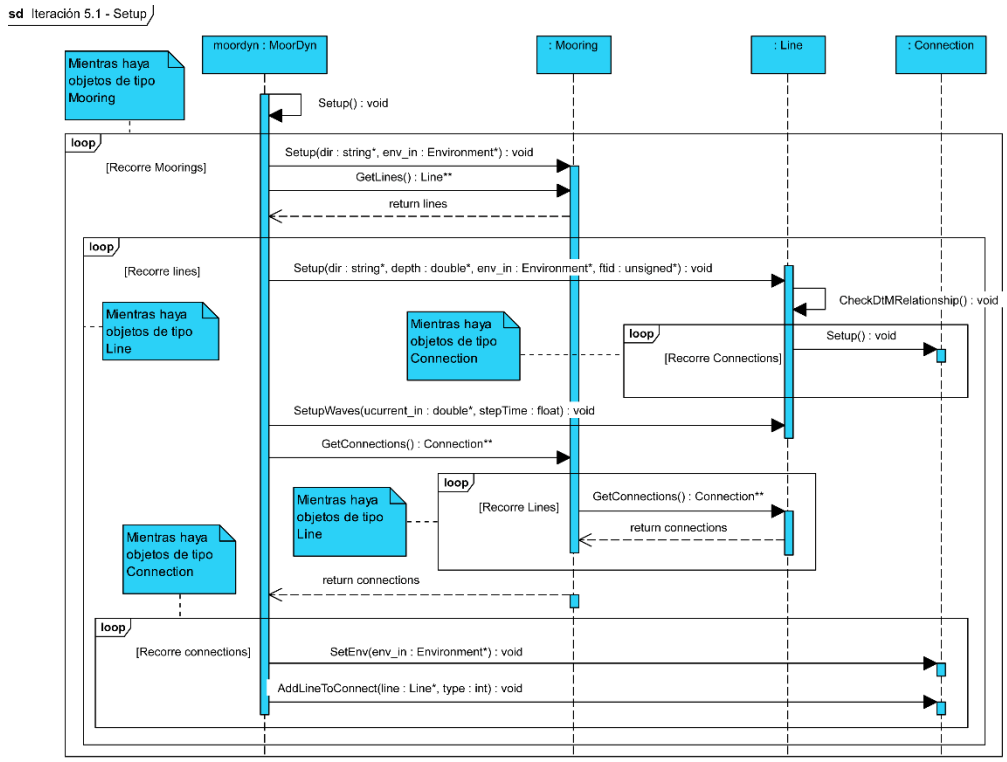


Figura 8-4: Diagrama de secuencia del Bloque 1 – Setup

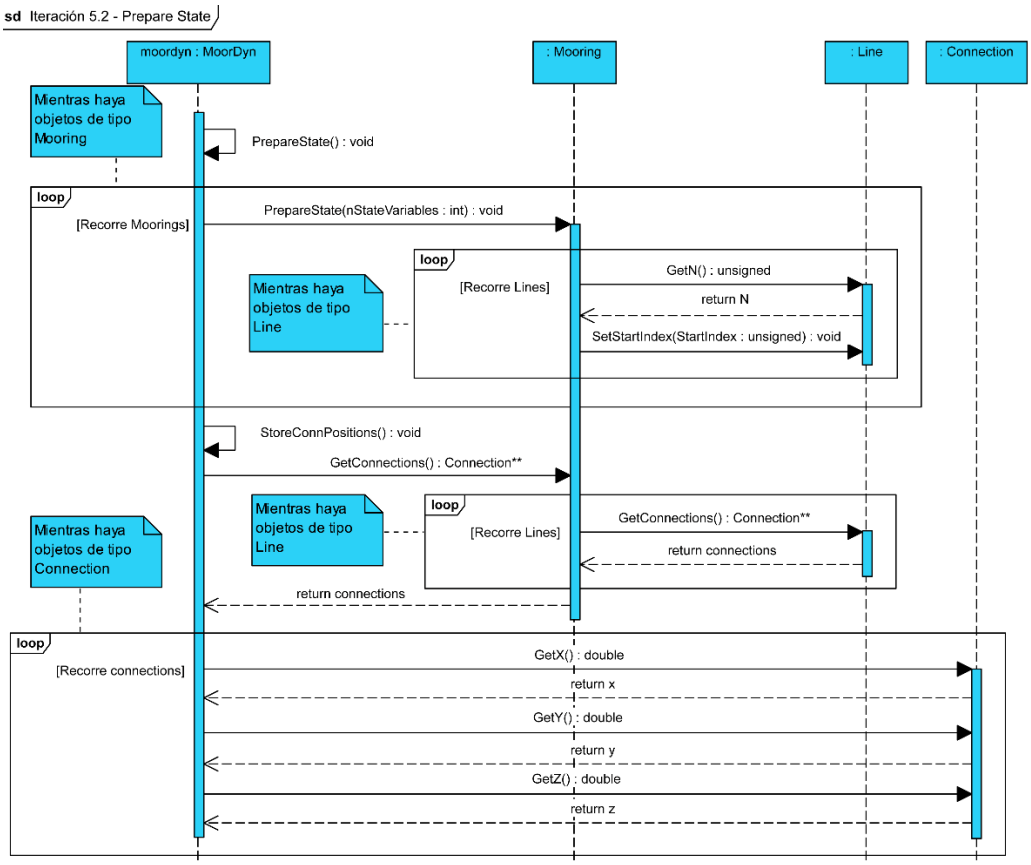


Figura 8-5: Diagrama de secuencia del Bloque 2 – Preparación de estados

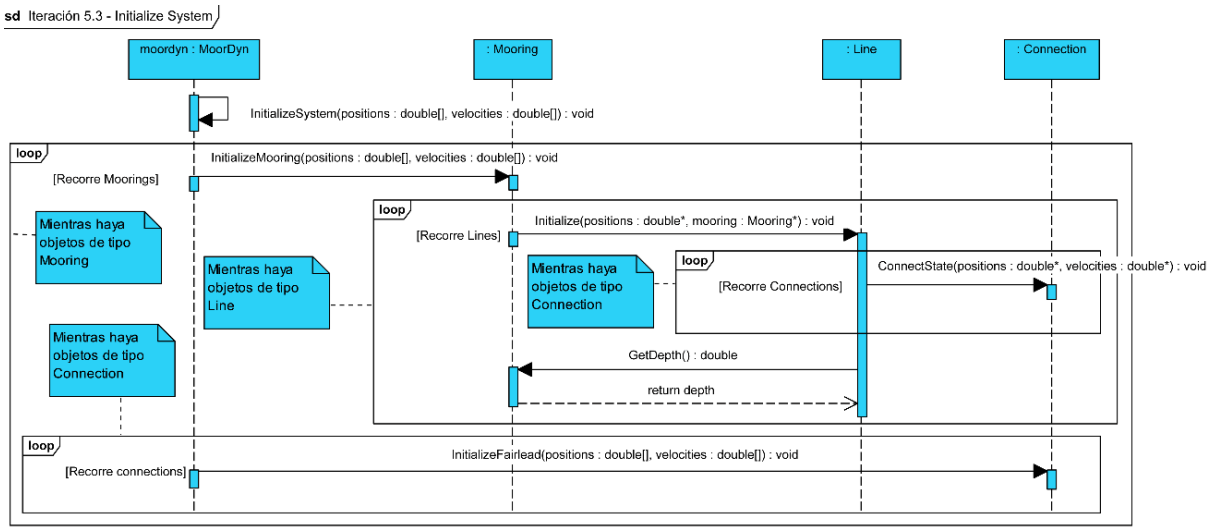


Figura 8-6: Diagrama de secuencia del Bloque 3 – Inicialización del sistema

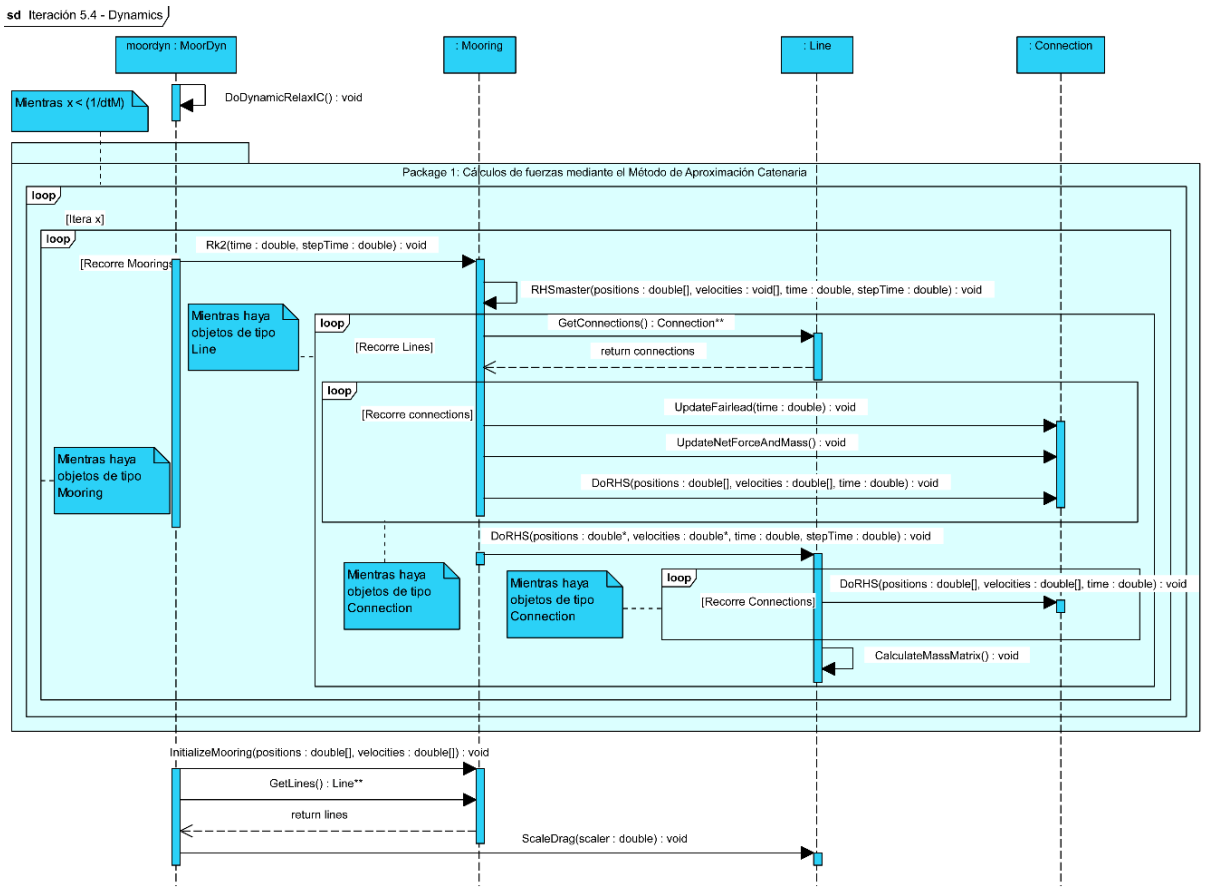


Figura 8-7: Diagrama de secuencia del Bloque 4 – Dinámica de amarres

8.2.3. Iteración 6: Cálculos de fuerzas

En la Figura 8-8 se añade una nota haciendo referencia a que en ese momento se repetirá el proceso contenido en el *package* de la Figura 8-7.

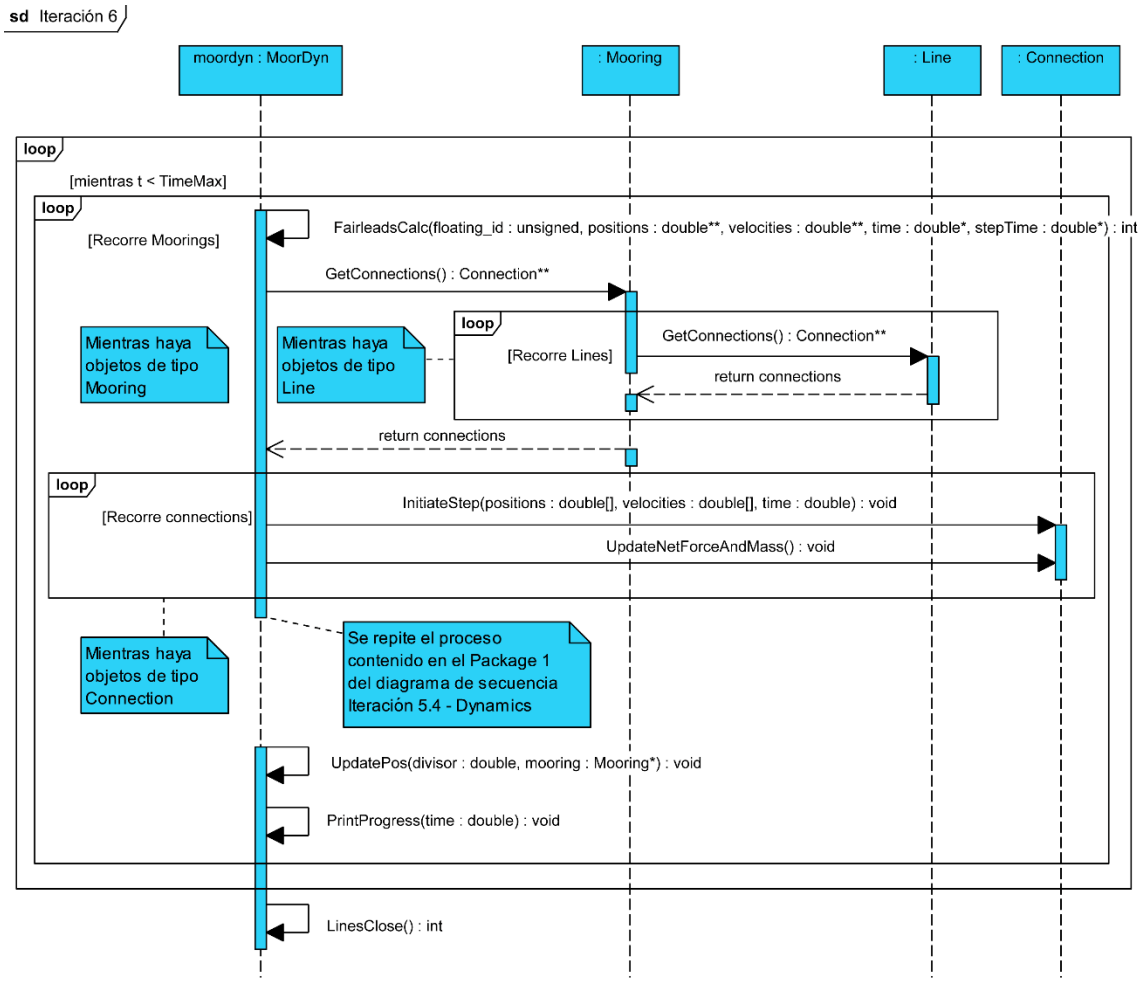


Figura 8-8: Diagrama de secuencia de cálculos de fuerzas

8.2.4. Iteración 7: Definición de la salida de datos

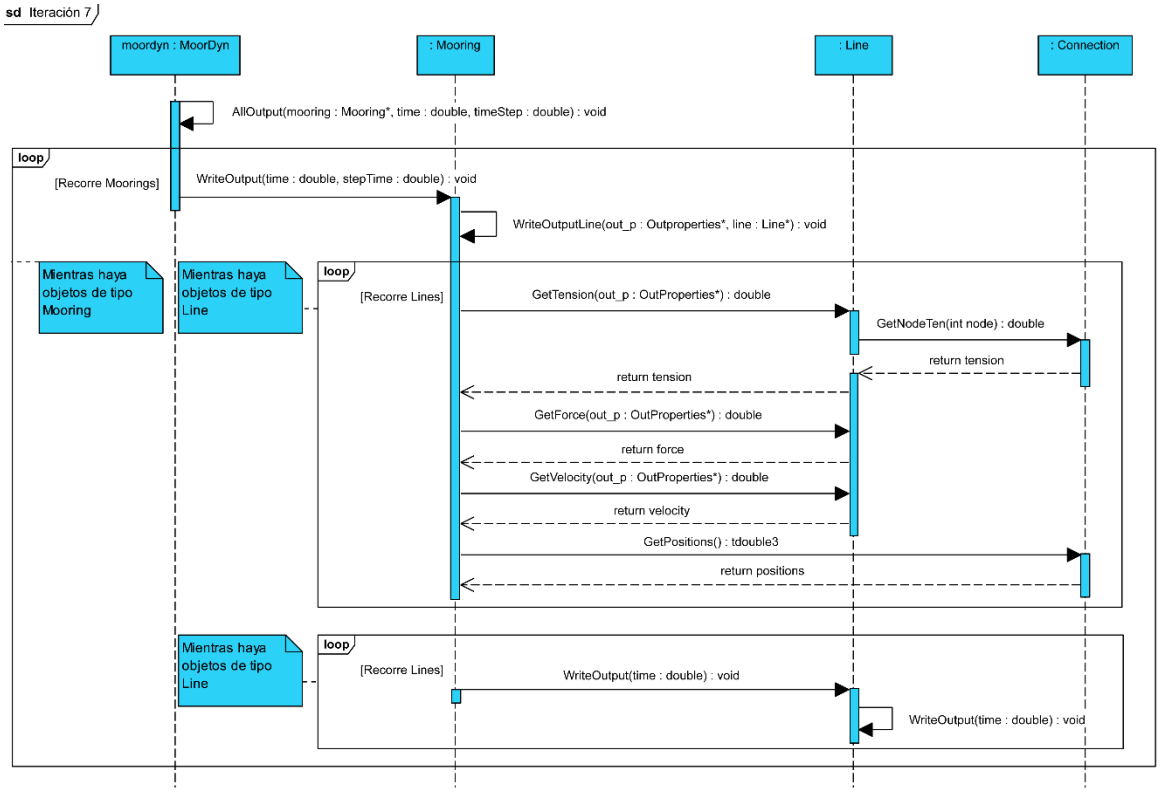


Figura 8-9: Diagrama de secuencia de salida de datos

8.2.5. Iteración 8: Acoplamiento con el software DualSPHysics

En el diagrama de secuencia de la Figura 8-10, se añade a DualSPHysics como actor. Esto se debe a que será una entidad externa al sistema (MoorDyn+). DualSPHysics provoca en MoorDyn+ la ejecución de una acción.

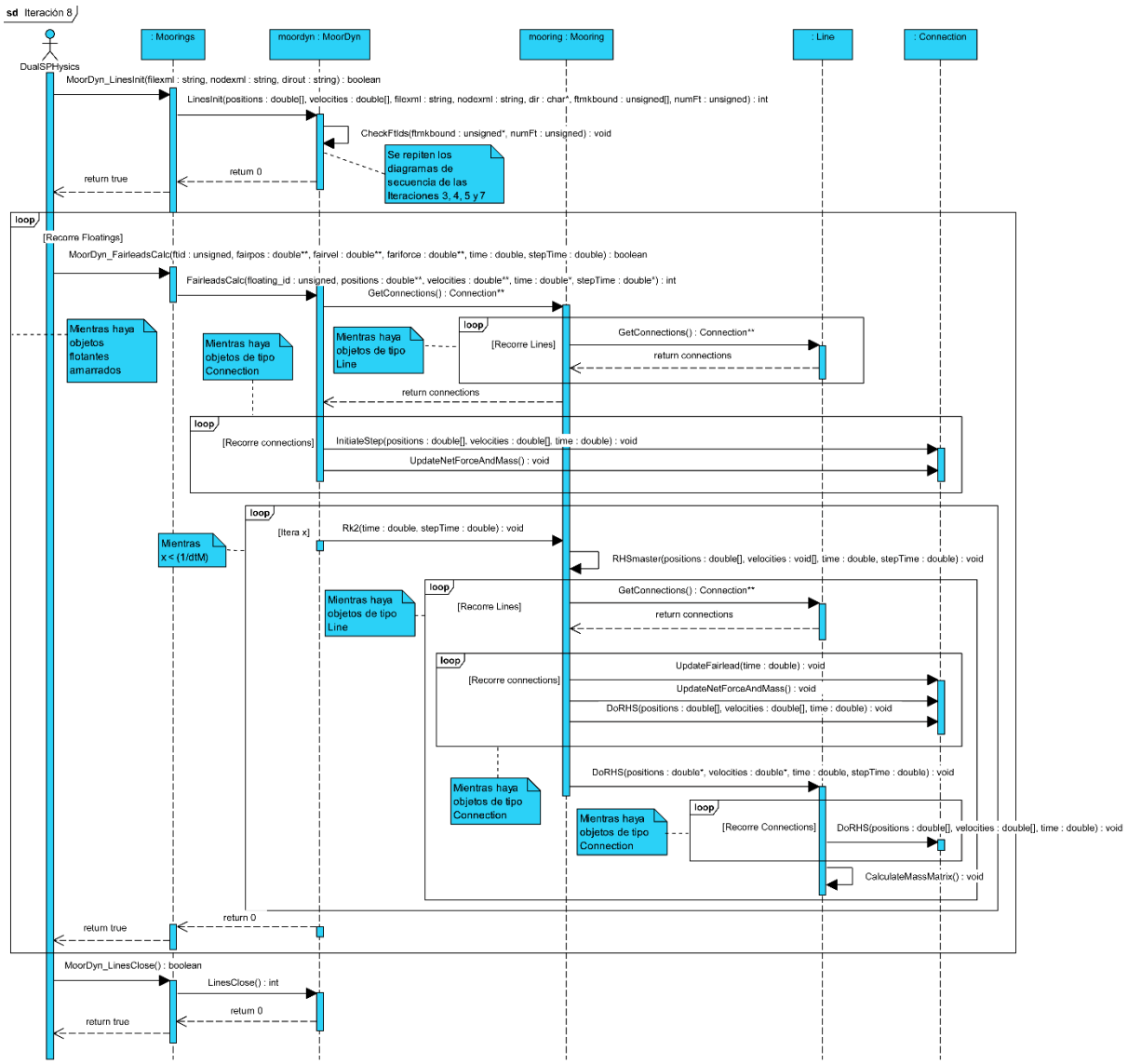


Figura 8-10: Diagrama de secuencia del acoplamiento con DualSPHysics

9. Gestión de datos e información

En este apartado se hace referencia a la entrada y la salida de datos de MoorDyn+. La entrada de datos será mediante un fichero XML. En este fichero se define la configuración inicial de la librería, permitiendo que se creen los objetos necesarios para la simulación. Se puede observar su estructura en la Figura 9-1. El fichero estará compuesto por un bloque llamado *moordyn* y dentro de este, se definen los siguientes bloques:

- *solverOptions*: Donde se definen las condiciones del entorno y los parámetros para la resolución de los cálculos.
- *mooring*: Donde se definen los amarres. Se pueden definir tantos bloques *mooring* como se deseen, uno a continuación del otro. Cada bloque *mooring* representa el conjunto de líneas de amarre conectadas a un objeto flotante. Dentro de este bloque, se definen los siguientes:
    - *lineDefault*: En este bloque se definen los parámetros que compartirán todas las líneas de este amarre.
    - *line:* Este bloque sirve para definir cada una de las líneas de amarre. Se pueden definir tantas como se quieran, una a continuación de la otra. Dentro de este bloque también se definen los extremos de cada línea:
        - *fixconnection*: Extremo fijo.
        - *vesselconnection*: Extremo móvil.
    - *output*: Este bloque se utiliza para definir el tipo de salida que se desea en función de los tipos de conexión y de las magnitudes de los extremos de conexión de las líneas de este amarre. Las magnitudes son:
        - *tension*: Almacena las tensiones.
        - *velocity*: Almacena las velocidades.
        - *position*: Almacena posiciones.

```
<?xml version="1.0" encoding="UTF-8" ?>
<moordyn>
    <solverOptions>

    </solverOptions>
    <mooring>
        <linedefault>

        </linedefault>
        <line>
            <fixconnection />
            <vesselconnection />
        </line>
        <output>

        </output>
    </mooring>
</moordyn>
```

Figura 9-1: Estructura básica del fichero de entrada

Mediante el lenguaje XML Schema, se ha elaborado la estructura que tienen que cumplir los ficheros de entrada de MoorDyn+. Se puede observar en detalle en el anexo *16.7. XML Schema de MoorDyn+*.

En la Figura 9-2 se muestra un ejemplo completo de la definición del fichero de entrada para realizar la simulación de dos amarres con una línea cada uno. Este ejemplo será utilizado para llevar a cabo las pruebas en el apartado *10. Pruebas llevadas a cabo*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<moordyn>
  <solverOptions>
    <gravity value="9.81" comment="(m/s^2)"/>
    <waterDepth value="2.5" comment="(m)"/>
    <kBoot value="3.0e6" comment="Stifness (Pa/m). (default=3.0e6)" />
    <cBot value="3.0e5" comment="Damping Pa/m/s). (default=3.0e5)" />
    <dtM value="0.0002" comment="time step to use in mooring integration. (default=0.0001)"/>
    <writeUnits value="yes" comment="write units line (default=yes) [yes no]" />
    <frictionCoefficient value="0" comment="General bottom friction coefficient, as a start. (default=0.0)" />
    <dtIC value="1.0" comment="[ICdt] convergence analysis time step. (default=1.0)"/>
    <cdScaleIC value="2" comment="[ICDfac] factor by which to scale drag coefficients during dynamic relaxation IC gen. (default=5)" />
    <tmaxIC value="10" comment="max time for IC generation.(default=120)"/>
    <timeMax value="10" comment="Time of simulation(s)" />
  </solverOptions>
  <mooring ref="50">
    <linedefault>
      <ea value="2.9e3" comment="stiffness(N)" />
      <diameter value="3.656E-3" comment="(m)" />
      <massDenInAir value="0.0607" comment="massDenInAirDenInAir (kg/m)" />
      <ouputsFlags value="p" comment="Node output properties.(default=-) [-=None, p=positions, t=tension]" />
    </linedefault>
    <line>
      <length value="1" comment="(m)"/>
      <segments value="20" />
      <fixconnection x="0.5" y="0.0" z="-2.50" />
      <vesselconnection x="-0.55" y="0" z="-2" />
    </line>
    <output>
      <time startTime="0" endTime="6" dtOut="0.2" comment="Default [startTime= 0; endTime= 0; dtOut=0.01]"/>
      <tension type="all" comment="type=[fixed|vessel|all].(default: type=all) Stores the tensions of connections types selected for each line. "/>
      <position type="all" comment="type=[fixed|vessel|all]. Stores the positions of connections types selected for each line" />
    </output>
  </mooring>
  <mooring ref="45">
    <linedefault>
      <ea value="2.9e3" comment="stiffness(N)" />
      <diameter value="3.656E-3" comment="(m)" />
      <massDenInAir value="0.0607" comment="massDenInAirDenInAir (kg/m)" />
      <ouputsFlags value="p" comment="Node output properties.(default=-) [-=None, p=positions, t=tension]" />
    </linedefault>
    <line>
      <length value="1" comment="(m)"/>
      <segments value="20" />
      <fixconnection x="-0.5" y="0.0" z="-2.50" />
      <vesselconnection x="0.5" y="0.0" z="-2.0" />
    </line>
    <output>
      <time startTime="0" endTime="6" dtOut="0.2" comment="Default [startTime= 0; endTime= 0; dtOut=0.01]"/>
      <tension type="all" comment="type=[fixed|vessel|all].(default: type=all) Stores the tensions of connections types selected for each line. "/>
      <position type="all" comment="type=[fixed|vessel|all]. Stores the positions of connections types selected for each line" />
    </output>
  </mooring>
</moordyn>
```

Figura 9-2: Ejemplo de fichero de entrada con dos amarres y una línea cada uno.

La salida de datos se hará en ficheros CSV. Hay dos tipos de ficheros, ver el apartado *7.5. Iteración 7: Definición de la salida de datos.* La estructura de ambos ficheros se caracteriza por la división en columnas.

El fichero de **Tipo 1** tiene la siguiente estructura de columnas:

- Time [s]. Representa el instante de simulación en el que se realiza el grabado de datos.
- Nodo. Se crea una columna por cada nodo y la magnitud elegida. Se puede ver la estructura en la Figura 9-3. El número de nodo se representa con *Num*. La magnitud se representa como *Flag*, que corresponde al caracter que se ha elegido en la etiqueta *outputsFlags*. Los caracteres son:
  - p: Posición.
  - v: Velocidad.
  - t: Tensión.
- Por cada Nodo-Magnitud se crearán 3 columnas, donde cada una de ellas representan el valor en X, Y, Z.

Time [s]	NodeNumFlagX [unidades]	NodeNumFlagY [unidades]	NodeNumFlagZ [unidades]

Figura 9-3: Estructura de fichero de salida de Tipo 1

El fichero de **Tipo 2** tiene la siguiente estructura de columnas:

- Time [s]. Representa el instante de simulación en el que se realiza el grabado de datos.

- Magnitud-Conexión. La conexión representa el extremo de cada línea. Puede ser *Vess* para conexiones móviles o *Fix* para conexiones fijas. Las magnitudes representadas son:
    - *Pos*: Posición.
    - *Vel*: Velocidad.

  Por cada Magnitud-Conexión se pondrá el número de la línea a la que hace referencia mediante *Linea* y se muestran los valores para las posiciones X, Y, Z. Ver la Figura 9-4.

Time [s]	MagnitudConexionLineaX [unidades]	MagnitudConexionLineaY [unidades]	MagnitudConexionLineaZ [unidades]

Figura 9-4: Estructura de fichero de salida de Tipo 2

También se representará la tensión mediante *Ten* que se graba en una única columna para cada tipo de conexión. Para el tipo de conexión fija se utilizará *Anch (anchor)* y para la conexión móvil se utilizará *Fair (fairlead)*, seguidos de *Linea* que representa el número de línea. Ver la Figura 9-5.

Time [s]	AnchTenLinea [N]	FairTenLinea [N]

Figura 9-5: Estructura de fichero de salida de Tipo 2 para tensiones

A continuación, se muestran dos ejemplos de los ficheros de salida resultantes. En el primero (fichero **tipo 1**), se muestra la información relativa a cada nodo de la línea que se corresponde con la etiqueta *outputsFlags* (en este caso únicamente los nodos de los extremos 0 y 20) y corresponde con la Figura 9-6. El segundo (fichero **tipo 2**), muestra la información de las líneas del amarre relativa a las magnitudes elegidas en el bloque *output* y corresponde con la Figura 9-7.

	A	B	C	D	BJ	BK	BL
1	Time	Node0px [m]	Node0py [m]	Node0pz [m]	Node20px [m]	Node20py [m]	Node20pz [m]
2	0.200007	-0.5	0	-2.5	0.467853	0	-1.99175
3	0.400001	-0.5	0	-2.5	0.423943	0	-1.97047
4	0.600013	-0.5	0	-2.5	0.382128	0	-1.94235
5	0.800025	-0.5	0	-2.5	0.34923	0	-1.90544
6	1.00003	-0.5	0	-2.5	0.319889	0	-1.86094
7	1.20002	-0.5	0	-2.5	0.287813	0	-1.81424
8	1.40001	-0.5	0	-2.5	0.251881	0	-1.76838
9	1.60001	-0.5	0	-2.5	0.213204	0	-1.72354
10	1.80001	-0.5	0	-2.5	0.175703	0	-1.68336
11	2.00001	-0.5	0	-2.5	0.141557	0	-1.65095

Figura 9-6: Fichero de salida MooringXX_LineYY.csv

	A	B	C	D	E	F	G	H	I
1	Time [s]	AnchTen0 [N]	FairTen0 [N]	PosFix0X [m]	PosFix0Y [m]	PosFix0Z [m]	PosVess0X [m]	PosVess0Y [m]	PosVess0Z [m]
2	0.200007	270.028	270.302	-0.5	0	-2.5	0.467853	0	-1.99175
3	0.400001	188.055	188.412	-0.5	0	-2.5	0.423943	0	-1.97047
4	0.600013	126.274	126.614	-0.5	0	-2.5	0.382128	0	-1.94235
5	0.800025	106.191	106.546	-0.5	0	-2.5	0.34923	0	-1.90544
6	1.00003	114.463	114.82	-0.5	0	-2.5	0.319889	0	-1.86094
7	1.20002	128.793	129.155	-0.5	0	-2.5	0.287813	0	-1.81424
8	1.40001	142.187	142.549	-0.5	0	-2.5	0.251881	0	-1.76838
9	1.60001	157.302	157.68	-0.5	0	-2.5	0.213204	0	-1.72354
10	1.80001	173.637	174.039	-0.5	0	-2.5	0.175703	0	-1.68336
11	2.00001	185.93	186.347	-0.5	0	-2.5	0.141557	0	-1.65095

Figura 9-7: Fichero de salida MooringXX.csv

10. Pruebas llevadas a cabo

En este apartado se muestran las pruebas llevadas a cabo en cada una de las iteraciones de la fase de implementación que sirven para validar los incrementos que agregan una nueva funcionalidad. Las pruebas son relativas a las comprobaciones más representativas que se han llevado a cabo, ya que, debido su extensión, no es posible mostrar las pruebas de todas las funciones implementadas.

Las pruebas mostradas en esta sección son realizadas en un sistema operativo Linux Ubuntu 18.04. Se define un mismo fichero de configuración XML. Consiste en crear dos amarres *(mooring)* con una línea *(line)* cada uno y cada línea con dos extremos *(vesselconnection* y *fixconnection)*. Ver la Figura 9-2.

Las pruebas realizadas no permiten mostrar la totalidad del control de excepciones integrado. Muchas de las excepciones son usadas para evitar cálculos que deriven en operaciones no permitidas (P. Ej: división por 0). No se puede forzar el lanzamiento de todas las excepciones debido a que, para llevar a cabo los cálculos, se usa el Método de Aproximación Catenaria [10], basado en cálculos mediante iteraciones. Se puede ver un listado en el anexo *16.4. Listado del control de excepciones*.

10.1. Iteración 3: Entrada de datos

En esta iteración se prueba el correcto funcionamiento de la parte encargada de leer la configuración del fichero de entrada. Solamente se hacen comprobaciones de la estructura del fichero XML, detectando etiquetas mal cerradas o ausencia de bloques o etiquetas. Primero se define un caso XML en los que se introducen errores conscientemente para probar que el programa lo detecta e informa al usuario. En este caso, se modifica una de las etiquetas para que el bloque *line* no quede cerrado. Ver la Figura 10-1.

```
<line>
        <length value="1" comment="(m)"/>
        <segments value="20" />
        <fixconnection x="0.5" y="0.0" z="-2.50" />
        <vesselconnection x="-0.55" y="0" z="-2" />
<line>
```

Figura 10-1: Etiqueta de cierre XML mal definida

Ante este error, se envía una excepción con la información relativa al error por consola de comandos. Esta excepción avisa de que hubo un error leyendo la etiqueta de cierre. Ver la Figura 10-2.

```
---------- START RUN --------

terminate called after throwing an instance of 'JException'
  what():  Exception (MoorDyn::LoadFile)
Text: Cannot load the xml file: Error reading end tag. (row:20 col:5)
File: moordyn.xml

Abortado (`core' generado)
```

Figura 10-2: Excepción por ausencia de etiqueta de cierre

A continuación, se define un caso XML correcto. Con esta prueba se pretende comprobar que se crean los objetos deseados. Se puede observar en la Figura 10-3 que se crean dos amarres, dos líneas (una línea por amarre) y cuatro conexiones (dos conexiones de cada tipo).

```
---------- START RUN --------

Summary:
        Number of Moorings: 2
        Number of Lines: 2
        Number of Connections: 4
                Number of Fixed Connections: 2
                Number of Vessel Connections: 2
```

Figura 10-3: Validación de la Iteración 3

Al finalizar las pruebas, se ha determinado que la funcionalidad que se desea agregar funciona correctamente. Por ello, el incremento queda integrado y validado. Al validar esta iteración, se cumple el *Objetivo 7*.

10.2. Iteración 4: Validación de entrada de datos

En esta iteración se ha añadido un conjunto de funciones que leen los datos introducidos por el usuario. Se pretende validar que dichos datos están dentro de los rangos establecidos. Se prueba introduciendo una longitud de línea de **0.4m**, menor que la distancia de entre los puntos. El programa lanza una excepción al detectarlo, ver la Figura 10-4.

```
---------- START RUN --------

MoorDyn::ReadXml() - Num Moorings: 2
Mooring: 0
num lines: 1
        ReadLine(): 0
                connection: 1 type: 0
                        x= 0.5000 ; y= 0.0000 ; z= -2.5000
                connection: 2 type: 1
                        x= -0.5500 ; y= 0.0000 ; z= -2.0000
terminate called after throwing an instance of 'JException'
  what():  Exception (JObject::Line::CheckInputParameters)
Text: ERROR: The length of the line 0 must be at least of 0.502494m.

Abortado (`core' generado)
```

Figura 10-4: Excepción por longitud de línea insuficiente

El código relativo al error de la Figura 10-4 se puede observar en la Figura 10-5.

```
//==============================================================================
/// Checks if the input parameters are into the range
//==============================================================================
void Line::CheckInputParameters() {
    //-Checks distance between 2 points
    double distance = sqrt((pow((abs(AnchConnect->GetX()) - abs(FairConnect->GetX())), 2))
        + (pow((abs(AnchConnect->GetY()) - abs(FairConnect->GetY())), 2))
        + (pow((abs(AnchConnect->GetZ()) - abs(FairConnect->GetZ())), 2))
    ); //distance between nodeAnch and nodeFair
    if (UnstrLen < distance) {
        string tex = "Error: The length of the line " + to_string(Number)
            + " must be at least of " + to_string(distance) + "m.";
        RunException("Line::CheckInputParameters", tex);
    }
    return;
}
```

Figura 10-5: Función CheckInputParameters

A continuación, se introducen datos válidos para comprobar que la funcionalidad está implementada correctamente y cada elemento creado tiene la información introducida por el usuario. Ver la Figura 10-6.

```
 ---------- START RUN --------

MoorDyn::ReadXml() - Num Moorings: 2
Mooring: 0
num lines: 1
        ReadLine(): 0
                connection: 1 type: 0
                        x= 0.5000 ; y= 0.0000 ; z= -2.5000
                connection: 2 type: 1
                        x= -0.5500 ; y= 0.0000 ; z= -2.0000
Mooring: 1
num lines: 1
        ReadLine(): 0
                connection: 1 type: 0
                        x= -0.5000 ; y= 0.0000 ; z= -2.5000
                connection: 2 type: 1
                        x= 0.5000 ; y= 0.0000 ; z= -2.0000

Summary:
        Number of Moorings: 2
        Number of Lines: 2
        Number of Connections: 4
                Number of Fixed Connections: 2
                Number of Vessel Connections: 2
```

Figura 10-6: Validación de la Iteración 4

Se puede observar que las posiciones de las conexiones de cada línea coinciden con las introducidas en el fichero de configuración, por esta razón, esta iteración queda validada e integrada.

10.3. Iteración 5: Inicialización del sistema

En esta iteración se ha probado que el sistema está correctamente inicializado. Para ello se deben ejecutar los métodos ***Setup()***, ***PrepareState()***, ***InitializeSystem()*** y ***DoDynamicRelaxIC()*** correctamente. Una de las comprobaciones que validan que el sistema esté correctamente inicializado, es que no se produzca una excepción. Esta excepción se lanza si se intenta representar la línea entre los extremos, mediante el Método de Aproximación Catenaria [10], y detecta *Not-a-Number values (NaN values)*. Esto se puede producir al intentar calcular la tasa de tensión de cada segmento. Es probable que, al introducir un valor de rigidez de una línea de amarre muy elevado, se produzca un *NaN value*, por ello se añade la siguiente comprobación que lanzaría la excepción, avisando del error y de la línea en la cual se produce. Ver la Figura 10-7.

```
//-Stores the distance between line nodes
for (int J = 0; J < 3; J++) {
    ldstr_top += (r[i + 1][J] - r[i][J])*(rd[i + 1][J] - rd[i][J]);
    if (std::isnan(ldstr_top)) {
        string tex = "Error:  NaN value detected in MoorDyn state at dynamic relaxation time "
            + to_string(time) + " s. Probably, the EA value for line " + to_string(Number)
            + " of mooring " + to_string(*FloatingId) + " is very high.";
        RunException("Line::DoRHS", tex);
    }
}
```

Figura 10-7: Excepción por la detección de valores Not-a-Number

Se introduce un valor de dtM mayor que el necesario para mantener equilibrio del sistema (para ver detalles de la relación de equilibrio, ir al apartado *7.2. Iteración 4: Validación de entrada de datos*). Se puede apreciar en la Figura 10-8, que se muestra el "*warning*" por el recálculo del valor del dtM.

```
 ---------- START RUN --------

MoorDyn::ReadXml() - Num Moorings: 2
Mooring: 0
num lines: 1
        ReadLine(): 0
                connection: 1 type: 0
                        x= 0.5000 ; y= 0.0000 ; z= -2.5000
                connection: 2 type: 1
                        x= -0.5500 ; y= 0.0000 ; z= -2.0000
Mooring: 1
num lines: 1
        ReadLine(): 0
                connection: 1 type: 0
                        x= -0.5000 ; y= 0.0000 ; z= -2.5000
                connection: 2 type: 1
                        x= 0.5000 ; y= 0.0000 ; z= -2.0000

******* Warning: Adjusted dtM to 0.000010 for fairlead calcs

Summary:
        Number of Moorings: 2
        Number of Lines: 2
        Number of Connections: 4
                Number of Fixed Connections: 2
                Number of Vessel Connections: 2
```

Figura 10-8: Validación del recálculo de dtM

El código encargado de recalcular el valor de dtM se puede observar en la Figura 10-9.

```
//==============================================================================
/// Checks if the relationship between dtM, N and EA is rigth
//==============================================================================
void Line::CheckDtMRelationship() {

    int NdtM = (int)ceil(Env->GetICdt() / Env->GetDtM0());   // Number of mooring model time steps per outer time step
    double dtM = Env->GetICdt() / NdtM;     // mooring model time step size (s)
    double weight = ((rho - Env->GetRho_w())*(pi / 4.*d*d))*9.81; //Weigth per unit length
    double frecuency = (2 * N / UnstrLen * sqrt(E / weight) * 0.159155) / 10; //Converts rad/s to Hz and divided by 10 for that the relationship was rigth
    double stepLimit = (1 / dtM); //Used to know the limit of the frecuency.
    if (frecuency > stepLimit) { //If natural frecuency is greater than the limit
        dtM = (UnstrLen / (2 * N  * sqrt(E / weight) * 0.159155)) * 10; // Multiply by 10 for the rigth relationship
        NdtM = (int)ceil(Env->GetICdt() / dtM);
        double dtm0_temp = Env->GetICdt() / NdtM;  // Number of mooring model time steps per outer time step
        // - Round the dtm to 0.0{0,n}1 values
        int exp = (int)(round(log10(dtm0_temp))); //Used to know the value of exponent 10^exp
        dtm0_temp = pow(10, exp) * 0.1; //Makes more small the dtm Value.
        if (dtm0_temp < Env->GetDtM0()) { //If the new dtM calculated is more small than the initial
            Env->SetDtM0(dtm0_temp); //Storse the new dtm value
            if (debug) { printf("\n********* Warning: Adjusted dtM to %.*f for fairlead calcs\n", (exp*(-1) + 1), Env->GetDtM0()); }
        }
    }
}
```

Figura 10-9: Cálculo del valor de dtM

Si no se produce ninguna excepción, la representación de la línea mediante el Método de Aproximación Catenaria se hizo correctamente, es decir, que la función ***DoDynamicRelaxIC()*** se ejecutó de forma satisfactoria. En este punto, el sistema estaría correctamente inicializado a partir de los datos de entrada, al no detectar ningún error en los cálculos. Esta prueba, valida esta iteración al dar por inicializado todo el sistema. Ver la Figura 10-10.

```
 ---------- START InitializeSystem --------

Creating mooring system. 2 moorings, 2 fairleads, 2 anchors, 4 connections.

 ---------- START DoDynamicRelaxIC --------

Finalizing ICs using dynamic relaxation ( 5.00 X normal drag )
Fairlead tensions converged to 0.1% after 5 seconds.

 ---------- END DoDynamicRelaxIC --------

 ---------- END InitializeSystem --------
```

Figura 10-10: Inicialización del sistema

10.4. Iteración 6: Cálculos de fuerzas

En esta iteración no se realizan pruebas que validen este incremento. Para validar este incremento es necesario analizar los datos de fuerzas calculados para cada una de las líneas de amarre y compararlos con datos experimentales. La validación de los cálculos de las fuerzas se llevará a cabo en la *Iteración 9*.

La única comprobación llevada a cabo, es que el sistema funciona como módulo independiente y realiza la ejecución hasta el tiempo de simulación indicado *(TimeMax)*. En el caso base de la Figura 9-2, se han establecido 4 segundos de simulación. Se puede ver en la Figura 10-11 que se ha completado con éxito.

```
  Finalizing ICs using dynamic relaxation ( 2.00 X normal drag )
   t = 5 s, tension at first fairlead is 342.4084 N
  Fairlead tensions converged to 0.1 % after 5 seconds.


==================================================
Time [s]        TimeMax [s]     Progress [%]
==================================================
0.50            4.00            12.50 %
1.00            4.00            25.00 %
1.50            4.00            37.50 %
2.00            4.00            50.00 %
2.50            4.00            62.50 %
3.00            4.00            75.00 %
3.50            4.00            87.50 %
4.00            4.00            100.00 %

 Lines Close.


 Execution completed successfully. Press any key to continue...
```

Figura 10-11: Ejecución completa de MoorDyn+ sin acoplamiento con otro modelo

10.5. Iteración 7: Definición de la salida de datos

En esta iteración, para el caso de la Figura 9-2 , se prueba el bloque *output* para la línea del primer amarre *(mooring ref=45)*. Se eligen las magnitudes de posición y tensión para ambos extremos de la línea. A mayores también se selecciona la opción *ouputsFlags* para mostrar las posiciones de los nodos de la línea de este amarre con la opción “p”. El resultado son los dos ficheros CSV que se aprecian en la Figura 10-12. En *Mooring45.csv* se muestran las tensiones, velocidades y posiciones de ambos extremos de la línea, mientras que en *Mooring45Line00.csv* se muestran las posiciones de cada nodo de la línea. Hay 21 nodos [0-20], dado que se definieron 20 segmentos mediante la etiqueta *segments*.

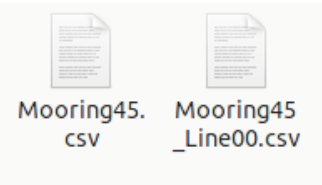


Figura 10-12: Ficheros de salida resultantes

De esta forma, la funcionalidad queda integrada al sistema y validada. El contenido de *Mooring45Line00.csv* se puede observar en la Figura 9-6, mientras que el contenido de *Mooring45.csv* se puede observar en la Figura 9-7.

10.6. Iteración 8: Acoplamiento con el software DualSPHysics

En esta iteración se comprueba el acoplamiento entre DualSPHysics y MoorDyn+. Para ello, se simula el comportamiento de dos objetos amarrados al suelo durante 3 segundos. La configuración de los amarres de MoorDyn+ se puede ver en la Figura 9-2. Para validar esta iteración se comprueba que ambos se comunican correctamente y que el resultado de la simulación se corresponde con lo esperado. En la Figura 10-13 se puede observar el estado de la simulación en distintos instantes.

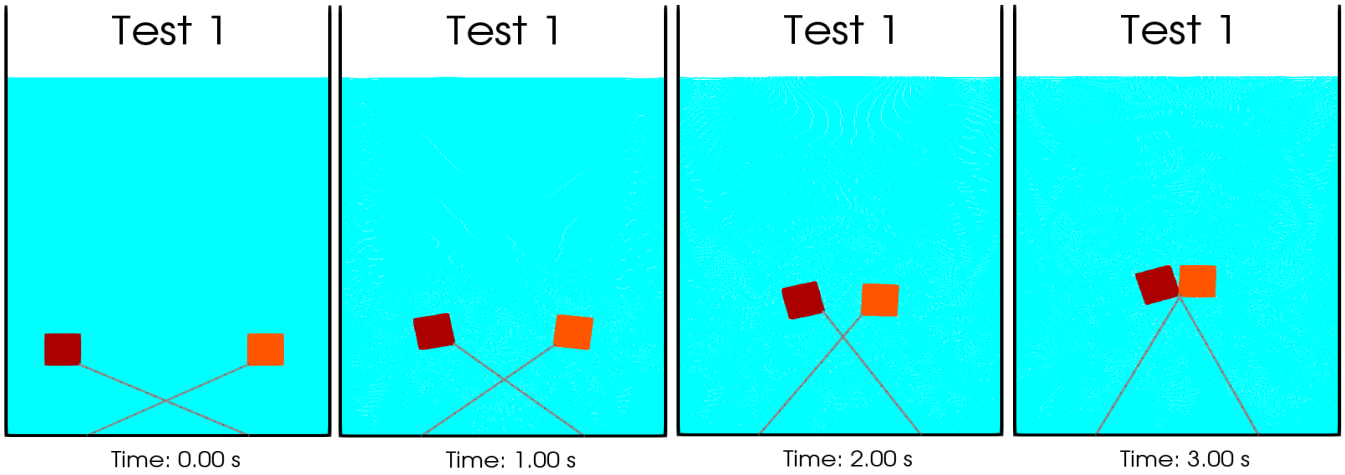


Figura 10-13: Distintos instantes de la simulación de 2 objetos flotantes amarrados al suelo.

Ilustración obtenida mediante el software de visualización ParaView.

La visualización del estado de los amarres se ha llevado a cabo con unas funciones que no estaban especificadas en el análisis de requisitos de esta iteración. Estas nuevas funciones se han documentado en el anexo *16.2. Nuevas funciones añadidas*.

El resultado de la simulación es el esperado. Los objetos flotantes tienden a ascender a la superficie, pero los amarres limitan su movimiento y finalmente acaban chocando entre sí. De esta forma se valida el acoplamiento con un software externo (DualSPHysics) y también que MoorDyn+ permite definir amarres a varios objetos flotantes. Esto supone que se cumplen el *Objetivo 4* y el *Objetivo 6* respectivamente.

10.7. Iteración 9: Pruebas globales

Una vez implementado el sistema, se realizan pruebas globales para validarlo. Para ello se comparan los datos obtenidos en la simulación de MoorDyn+ acoplado a DualSPHysics con datos experimentales.

Los datos experimentales se obtuvieron en los experimentos realizados en el canal de oleaje del Grupo de Investigación de Ingeniería Costera de la Universidad de Gante, Bélgica (UGENT). Este experimento forma parte del proyecto Europeo MaRINET2 EsfLOWC. El proyecto consiste en estudiar la interacción fluido-estructura entre olas y estructuras flotantes amarradas al fondo. El

objeto flotante es una caja cerrada con dimensiones 20x20x13,2cm. El sistema de amarre conecta la caja flotante al suelo a través de cuatro cadenas de la misma longitud, ver la Figura 10-14.

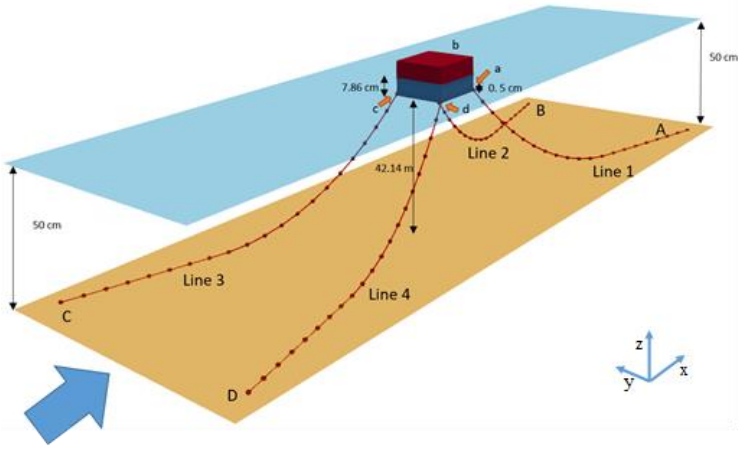


Figura 10-14: Configuración del experimento realizado por la Universidad de Gante

Se configura la simulación con DualSPHysics usando las dimensiones del experimento y el sistema de amarres para MoorDyn+. Se simula el caso y se obtienen los datos relativos a las tensiones de las conexiones A y B de las líneas 1 y 2 respectivamente. Posteriormente, se realiza una comparativa entre los datos obtenidos de la simulación y los del experimento. Esta comparativa se puede observar en la Figura 10-15 y en la Figura 10-16. Los resultados experimentales se corresponden con el color rojo (EXP) y los resultados de la simulación numérica se corresponden con el color azul (MoorDyn+).

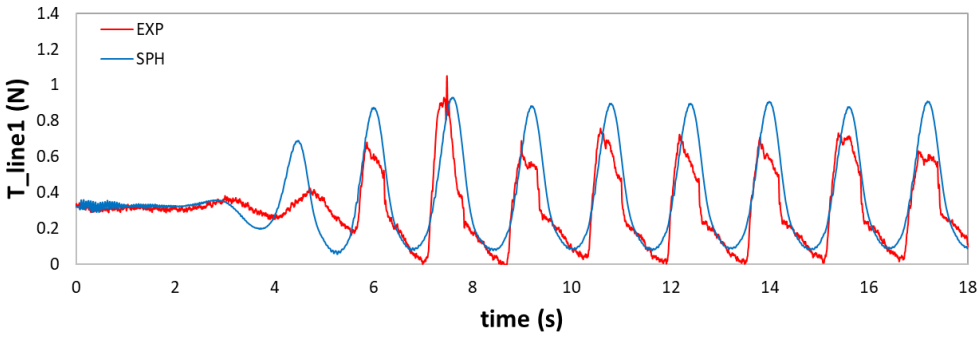


Figura 10-15: Tensiones experimentales y numéricas en la conexión A (Línea 1)

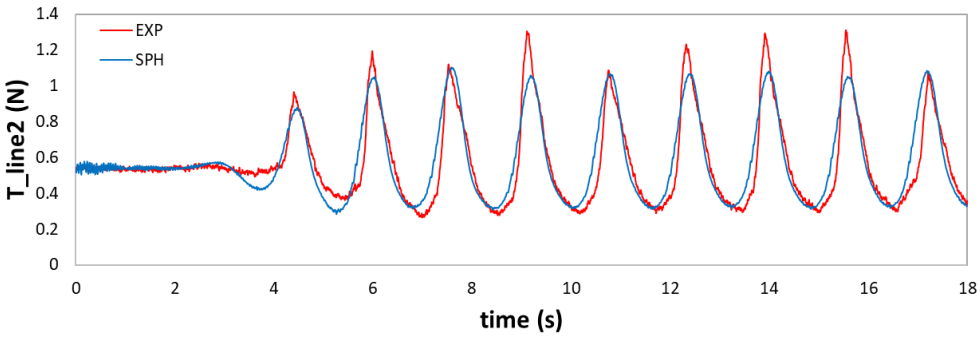


Figura 10-16: Tensiones experimentales y numéricas en la conexión B (Línea 2)

A mayores, se comparan los datos del movimiento del objeto flotante obtenidos por DualSPHysics con los experimentales. Esta comparación pretende demostrar que gracias a las fuerzas calculadas por MoorDyn+, DualSPHysics es capaz de simular los movimientos de la caja de forma muy parecida al experimento. Los movimientos analizados son, el desplazamiento en el eje X (surge), ver la Figura 10-17. El cabeceo o rotación sobre el eje Y (pitch), ver la Figura 10-18. El desplazamiento en el eje Z (heave), ver la Figura 10-19.

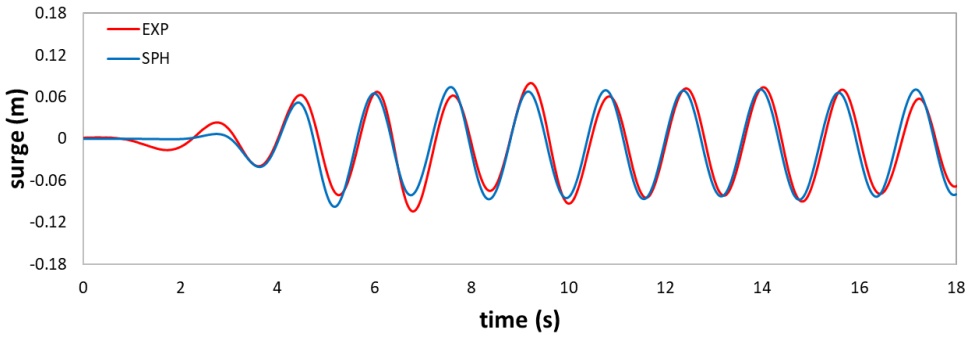


Figura 10-17: Desplazamiento del objeto flotante en el eje X

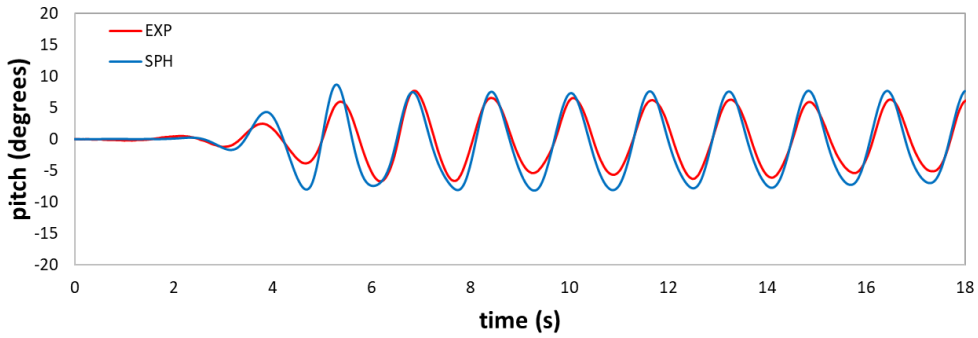


Figura 10-18: Rotación del objeto flotante en el eje Y

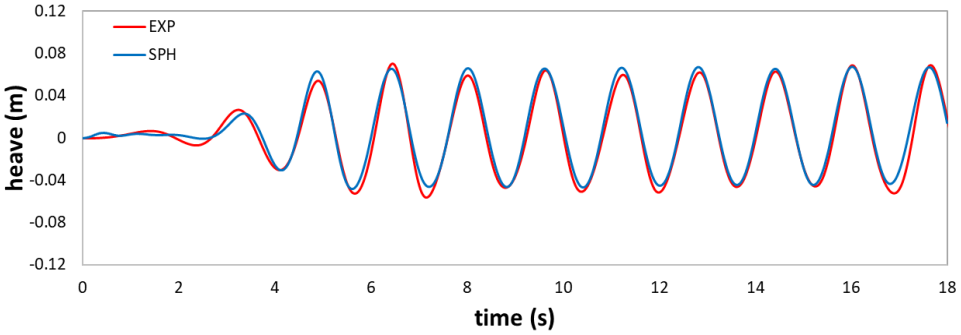


Figura 10-19: Desplazamiento del objeto flotante en el eje Z

Tras analizar los datos obtenidos y ver que la librería realiza unos cálculos muy próximos a los obtenidos experimentales, el sistema se da por validado. Gracias a ello se cumple el objetivo principal *(Objetivo 1)*.

11. Manual de usuario

Este manual está orientado a explicar el uso de MoorDyn+ acoplado con DualSPHysics. Es necesario disponer del directorio **DSPH-MoorDyn+** que se suministra en el CD. Cada vez que se haga referencia a un directorio o fichero contenido en DSPH-MoorDyn+, se puede observar la ruta en el anexo *16.5. Estructura del directorio DSPH-MoorDyn+*. Para llevar a cabo la ejecución de los ejemplos, será necesario copiar el directorio DSPH-MoorDyn+ en el equipo local.

Los apartados de configuración y ejecución de este manual, están enfocados a un ejemplo en concreto (*examples/01_TwoMoorings*). Cabe destacar, que en el directorio DSPH-MoorDyn+ ya se incluyen los ejecutables para probar los ejemplos sin necesidad de compilar el código fuente.

11.1. Compilación

En caso de realizar cambios en MoorDyn+, es necesario compilar el código fuente. Los siguientes apartados explican el proceso de compilación para Windows y Linux.

DualSPHysics se compila incluyendo la librería estática resultante de la compilación de MoorDyn+. Por esta razón, es necesario compilar también DualSPHysics. DualSPHysics se puede compilar para CPU y GPU. La diferencia entre ambas versiones se puede ver en detalle en el siguiente enlace: https://github.com/DualSPHysics/DualSPHysics/wiki/4.-CPU-and-GPU-implementation.

Para poder compilar el código en GPU, es necesario tener instalados los drivers de **Cuda 9.2** o posterior. En caso de no tenerlos, solamente se podrá compilar y ejecutar la versión de CPU. Se puede ver en detalle en el siguiente enlace: https://github.com/DualSPHysics/DualSPHysics/wiki/7.-Compiling-DualSPHysics.

11.1.1. Compilación en Windows

Es necesario abrir la solución *src/VS/**DualSPHysics4ReMoorDyn_vs2015.sln*** con Visual Studio 2015. La solución, contiene el proyecto de *DualSPHysics* y dos proyectos de MoorDyn+ *(MoorDyn* y *TestMoorDyn)*. El proyecto *MoorDyn* contiene el código fuente de MoorDyn+ y genera la librería estática para acoplarlo con DualSPHysics. *TestMoorDyn* contiene el fichero *Main.cpp* y genera el ejecutable de MoorDyn+ para usarlo como módulo independiente. Pasos para la compilación:

1. Seleccionar la configuración de la solución para plataformas *x64*.
   1.1. Modo *ReleaseCPU* para la versión de CPU:

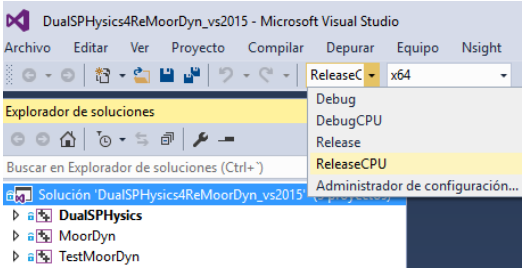


Figura 11-1: Configuración de la solución para CPU

1.2. Modo *Release* para la versión de GPU:

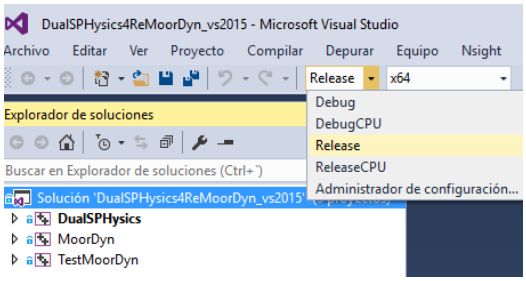


Figura 11-2: Configuración de la solución para GPU

2. Compilar la solución pulsando en “Compilar solución”.

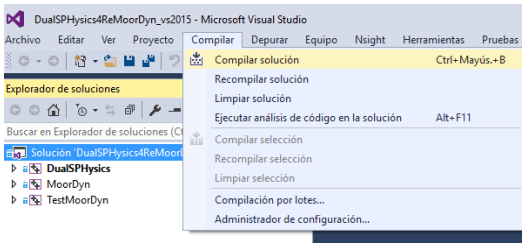


Figura 11-3: Compilación con Visual Studio 2015

3. Una vez compilado, se genera en el directorio *src/Run* un ejecutable llamado ***TestMoorDyn_win64.exe*** para utilizarlo como un módulo independiente y una librería estática llamada ***Moordyn_x64_v140_Release.lib*** en el directorio *src/Libs*. Esta librería estática se usa para acoplar Moordyn+ con otro software. El proceso de compilación del proyecto *DualSPHysics* incluye la librería estática para realizar el acoplamiento.

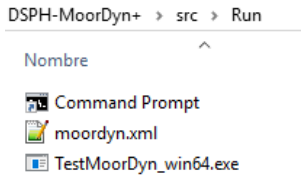


Figura 11-4: Ejecutable de MoorDyn+ para Windows

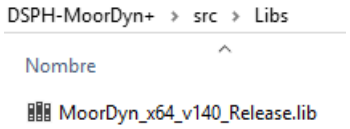


Figura 11-5: Librería estática de MoorDyn+ para Windows

11.1.2. Compilación en Linux

1. Comprobar la versión de GCC que se tiene instalada en el equipo.

```
ivan@ubuntu:~$ gcc --version
gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Figura 11-6: Versión de GCC

2. Si se dispone de una versión igual o superior a GCC 5.0, es necesario editar los archivos "makefile" y poner a *USE_GCC5=YES*. En caso contrario, *USE_GCC5=NO*.

```
#TestMoorDyn (11-04-2019)

#=============== Compilation Options
USE_GCC5=YES
USE_DEBUG=NO
USE_FAST_MATH=YES
USE_NATIVE_CPU_OPTIMIZATIONS=NO
COMPILE_CHRONO=YES
COMPILE_RZ_MLPISTONS=YES
```

Figura 11-7: Versión de GCC para la compilación de los makefile

3. Ejecutar el comando **make** en el directorio *src/source/Source_MoorDyn* introduciendo la opción *–f* ***Makefile_MoorDyn*** para compilar el código fuente de MoorDyn+.

```
ivan@ubuntu:~/DSPH-MoorDyn+/src/source/Source_MoorDyn$ make -f Makefile_MoorDyn
```

Figura 11-8: Ejecución del Makefile de MoorDyn+

4. Una vez compilado, se genera en el directorio *src/Run* un ejecutable llamado ***TestMoorDyn_linux64*** (para utilizarlo como un módulo independiente) y una librería llamada ***libmoordyn_64*.a** en el directorio *src/source* (para poder acoplarlo a otro software).

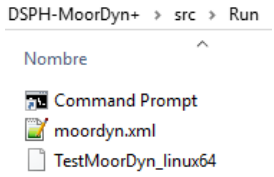


Figura 11-9: Ejecutable de MoorDyn+ para Linux

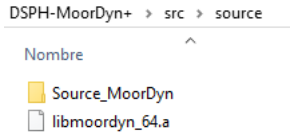


Figura 11-10: Librería estática de MoorDyn+ para Linux

5. Ejecutar el comando **make** para compilar DualSPHysics con la librería estática generada por MoorDyn+. Para ello es necesario acceder desde un terminal al directorio *src/source.*

   5.1. Para compilar la versión de CPU de DualSPHysics, ejecutar el comando para el fichero ***Makefile_cpu***.

```
ivan@ubuntu:~/DSPH-MoorDyn+/src/source$ make -f Makefile_cpu
```

Figura 11-11: Ejecución del Makefile de DualSPHysics para CPU

   5.2. Para compilar la versión de GPU de DualSPHysics, ejecutar el comando para el fichero ***Makefile***.

```
ivan@ubuntu:~/DSPH-MoorDyn+/src/source$ make -f Makefile
```

Figura 11-12: Ejecución del Makefile de DualSPHysics para GPU

11.2. Configuración de DualSPHysics

La configuración de DualSPHysics utiliza un fichero de entrada **Case*NombreCaso*_Def.xml**. En este fichero se configuran una gran cantidad de parámetros que no se van a explicar en este trabajo, pero se pueden ver en detalle en el siguiente enlace:

https://github.com/DualSPHysics/DualSPHysics/wiki/9.-Pre%E2%80%90processing.

Únicamente se explicará en detalle la creación de objetos flotantes y sus amarres. En este fichero XML se definen los objetos flotantes que posteriormente se desean amarrar con MoorDyn+. Primero, es necesario crear su geometría. En la Figura 11-13 se muestra la creación de dichos objetos mediante *drawbox* y se le asigna un identificador *(mk)*. La etiqueta *point* de *drawbox* especifica la posición inicial y la etiqueta *size* el tamaño del objeto. En la sección *floatings* se indica que ambos objetos son flotantes, como se muestra en la Figura 11-14. Finalmente, hay que indicar los objetos flotantes que dispondrán de amarres simulados por MoorDyn+ (ver la Figura 11-15).

```
<setmkbound mk="45" />
<drawbox>
    <!-- Box with moorings mk=45 -->
    <boxfill>solid</boxfill>
    <point x="0.5" y="-2" z="-2" />
    <size x="0.2" y="4" z="0.2" />
</drawbox>
    <!-- Box with moorings mk=50 -->
<setmkbound mk="50" />
<drawbox>
    <boxfill>solid</boxfill>
    <point x="-0.75" y="-2" z="-2" />
    <size x="0.2" y="4" z="0.2" />
</drawbox>
```

Figura 11-13: Definición de objetos flotantes

```
<floatings>
    <floating mkbound="45" relativeweight="0.5" />
    <floating mkbound="50" relativeweight="0.5" />
</floatings>
```

Figura 11-14: Creación de objetos con drawbox

```
<mooredfloatings>
    <floating mkbound="45" comment="Mkbound of the Floating body the mooring is linked to" />
    <floating mkbound="50" comment="Mkbound of the Floating body the mooring is linked to" />
</mooredfloatings>
```

Figura 11-15: Definición de objetos flotantes amarrados

Una vez creados los objetos flotantes, se elegirá dónde estará ubicada la configuración de MoorDyn+. Hay dos formas: integrándola en el fichero de entrada de DualSPHysics o como un fichero independiente. Con la opción de integrarla con el fichero de DualSPHysics, la configuración estará dentro del bloque que se muestra en la Figura 11-16. Para la opción de usar un fichero independiente (también XML), se especifica su nombre con el atributo *file*, como se puede ver en la Figura 11-17.

```
<moordyn comment="MoorDyn configuration">

</moordyn>
```

Figura 11-16: Configuración integrada en el fichero de DualSPHysics

```
<moordyn file="moordyn.xml" />
```

Figura 11-17: Configuración en un fichero independiente

11.3. Configuración de MoorDyn+

La configuración de MoorDyn+ siempre tiene que estar contenida dentro de un bloque *moordyn* como el de la Figura 11-16, ya sea integrada con DualSPHysics, o como en un fichero independiente. En MoorDyn+ se establece la definición de los amarres de cada uno de los objetos creados anteriormente. Cada amarre se define mediante el bloque *mooring*, el cual tendrá un identificador *(ref)*. En la Figura 11-18 se definen dos amarres con identificadores 45 y 50 que coinciden con los de los objetos flotantes.

```
<mooring ref="50">
  <linedefault>
    <ea value="2.9e3" comment="stiffness(N)" />
    <diameter value="3.656E-3" comment="(m)" />
    <massDenInAir value="0.0607" comment="massDenInAirDenInAir (kg/m)" />
    <ouputsFlags value="p" comment="Node output properties.(default=-) [-=None, p=positions, t=tension]" />
  </linedefault>
  <line>
    <length value="1" comment="(m)"/>
    <segments value="20" />
    <fixconnection x="0.5" y="0.0" z="-2.50" />
    <vesselconnection x="-0.55" y="0" z="-2" />
  </line>
  <output>
    <time startTime="0" endTime="6" dtOut="0.2" comment="Default [startTime= 0; endTime= 0; dtOut=0.01]"/>
    <tension type="all" comment="type=[fixed|vessel|all].(default: type=all) Stores the tensions of connections types selected for each line. "/>
    <position type="all" comment="type=[fixed|vessel|all]. Stores the positions of connections types selected for each line" />
  </output>
</mooring>
<mooring ref="45">
  <linedefault>
    <ea value="2.9e3" comment="stiffness(N)" />
    <diameter value="3.656E-3" comment="(m)" />
    <massDenInAir value="0.0607" comment="massDenInAirDenInAir (kg/m)" />
    <ouputsFlags value="p" comment="Node output properties.(default=-) [-=None, p=positions, t=tension]" />
  </linedefault>
  <line>
    <length value="1" comment="(m)"/>
    <segments value="20" />
    <fixconnection x="-0.5" y="0.0" z="-2.50" />
    <vesselconnection x="0.5" y="0.0" z="-2.0" />
  </line>
  <output>
    <time startTime="0" endTime="6" dtOut="0.2" comment="Default [startTime= 0; endTime= 0; dtOut=0.01]"/>
    <tension type="all" comment="type=[fixed|vessel|all].(default: type=all) Stores the tensions of connections types selected for each line. "/>
    <position type="all" comment="type=[fixed|vessel|all]. Stores the positions of connections types selected for each line" />
  </output>
</mooring>
</moordyn>
```

Figura 11-18: Definición de amarres para MoorDyn+

11.4. Ejecución

Para poder ejecutar los casos, se suministran dos ejecutables para Windows y otros dos ejecutables para Linux. Uno de ellos, sólo puede ejecutar DualSPHysics en CPU mientras que el otro ejecutable puede usar CPU o GPU. Se incluyen 4 ejemplos para poder ejecutar DualSPHysics con MoorDyn+. Estos se encuentran en el directorio *examples*. Todos los ejemplos presentan los mismos elementos, pero con distintas configuraciones y nombres. Cada ejemplo contiene los ficheros de la Figura 11-19.

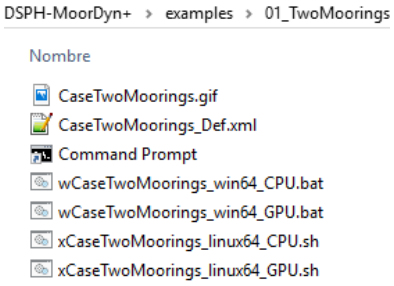


Figura 11-19: Contenido del directorio 01_TwoMoorings

- **CaseTwoMoorings.gif**: Animación con el resultado de la ejecución.
- **CaseTwoMoorings_Def.xml**: Fichero de configuración de DualSPHysics y MoorDyn+.
- **Command Prompt**: Símbolo del sistema utilizado en Windows para ejecutar comandos.
- **wCaseTwoMoorings_CPU.bat:** Script para ejecutar el caso en CPU en Windows.
- **wCaseTwoMoorings_GPU.bat:** Script para ejecutar el caso en GPU en Windows.
- **xCaseTwoMoorings_CPU.sh:** Script para ejecutar el caso en CPU en Linux.
- **xCaseTwoMoorings_GPU.sh:** Script para ejecutar el caso en GPU en Linux.

El contenido de los scripts (*.bat* y *.sh*) se puede ver en detalle en el anexo *16.6. Contenido de los Scripts de ejecución*.

11.4.1. Ejecución en Windows

1. Abrir **Command Prompt** en el directorio del ejemplo a ejecutar.

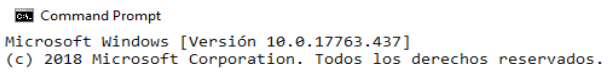


Figura 11-20: Command Promt de Windows

2. Escribir el nombre del script *.bat* a ejecutar (si no se dispone de una GPU compatible con CUDA o no se instalaron los drivers de **Cuda 9.2** o posterior, ejecutar el script de CPU).

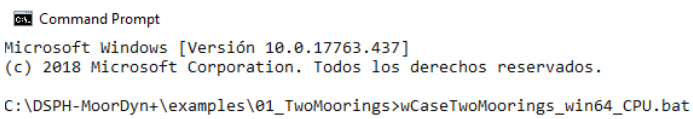


Figura 11-21: Ejecución del Script .bat

11.4.2. Ejecución en Linux

1. Abrir un **Terminal** de Linux en el directorio del ejemplo a ejecutar**.**

```
ivan@ubuntu:~/DSPH-MoorDyn+/examples/01_TwoMoorings$
```

Figura 11-22: Terminal de Linux

2. Dar permisos de ejecución a los scripts de Linux.

```
ivan@ubuntu:~/DSPH-MoorDyn+/examples/01_TwoMoorings$ chmod u+x xCaseTwoMoorings_linux64_*
```

Figura 11-23: Permiso de ejecución para los scripts de Linux

3. Lanzar el script *.sh* a ejecutar (si no se dispone de una GPU compatible con CUDA o no se instalaron los drivers de **Cuda 9.2** o posterior, ejecutar el script de CPU).

```
ivan@ubuntu:~/DSPH-MoorDyn+/examples/01_TwoMoorings$ ./xCaseTwoMoorings_linux64_CPU.sh
```

Figura 11-24: Ejecución del Script .sh

Para ambas plataformas, se ejecutará DualSPHysics acoplado a MoorDyn+. Se muestran los objetos flotantes amarrados (*mkbound*=45 y *mkbound*=50), el número de partículas creadas (50551) y el tiempo que llevará la simulación (*Finish time*).

```
ForcePoints configuration:
  Point_0(0.5,0,-2) in Floating_0 with mkbound=45. Distance to particle: 0 (0 x Dp)
  Point_1(-0.55,0,-2) in Floating_1 with mkbound=50. Distance to particle: 1.11022e-16 (1.11022e-14 x Dp)

Allocated memory in CPU: 8586520 (8.19 MB)
Part_0000        50551 particles successfully stored

[Initialising simulation (jqqoyosc)  17-04-2019 11:03:58]
PART       PartTime      TotalSteps    Steps    Time/Sec   Finish time
=========  ============  ============  =======  =========  ==================
Part_0001      0.020007           700      700     738.81  17-04-2019 11:53:13
Part_0002      0.040015          1400      700     771.72  17-04-2019 11:54:20
Part_0003      0.060022          2100      700     743.88  17-04-2019 11:54:06
```

Figura 11-25: Simulación de DualSPHysics y MoorDyn+

Se creará un directorio *examples/01_TwoMoorings/**CaseTwoMoorings_out*** al arrancar el script de ejecución. En este directorio se almacenarán los archivos de salida de DualSPHysics y de MoorDyn+. Concretamente en ***moordyn_data/*** se guardarán los ficheros generados por MoorDyn+ relativos a las fuerzas calculadas por los amarres con sus identificadores definidos anteriormente (45 y 50).

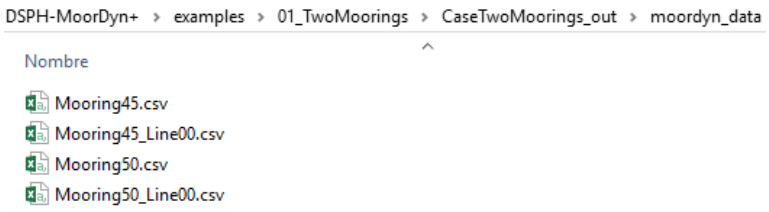


Figura 11-26: Archivos de salida de MoorDyn+

12. Principales aportaciones

Se ha re-implementado una librería que permite resolver dinámicamente el efecto de amarres (formados por una o varias líneas o cadenas) sobre varios objetos flotantes. Los resultados obtenidos, reflejados en el apartado *10.7. Iteración 9: Pruebas globales*, han demostrado que la librería resuelve de una forma precisa el efecto de los amarres sobre objetos flotantes, acercándose bastante a los datos obtenidos experimentalmente. Esta aportación cumple el *Objetivo 1*.

La librería ha sido escrita en C++, siguiendo el paradigma de la programación orientada a objetos, *Objetivo 5*. Esta técnica ha favorecido a cumplir uno de los objetivos fundamentales de este trabajo, conseguir la definición de varios amarres para múltiples objetos flotantes, *Objetivo 6*.

La librería es multiplataforma. Se puede utilizar tanto para Linux como para Windows, cumpliendo el *Objetivo 2*. Para poder obtener la versión de Linux, se hace uso de un *makefile* que compila los archivos del código fuente y genera el ejecutable y una librería estática para utilizar con otro software externo. Para Windows, se hace uso de una *solución* de Visual Studio 2015 con la que también se compilan los archivos y también genera el ejecutable y la librería estática. De esta forma se consiguen dos objetivos más:

1. La librería funciona como un módulo donde simula un movimiento predefinido de las conexiones móviles de cada amarre, esto supone el *Objetivo 3*. Al generar el ejecutable, se puede utilizar la librería como una aplicación que resuelve los cálculos de fuerzas sobre un punto que se desplaza por el plano.
2. La librería funciona acoplada a DualSPHysics, cumple así el *Objetivo 4*. Para realizar el acoplamiento se ofrece una interfaz de comunicación, ver la Figura 7-7, además de la librería estática resultante del proceso de compilación de MoorDyn+.

MoorDyn+ presenta un control de excepciones que avisa al usuario en el caso de producirse un error en el cálculo de los amarres o en la lectura del fichero de configuración XML, *Objetivo 8*. Se puede ver en detalle en el anexo *16.4. Listado del control de excepciones*.

La librería se configura a partir de un fichero de formato XML, *Objetivo 7*. Además, la salida de datos se guarda en ficheros de tipo CSV, lo que cumpliría el *Objetivo 9*.

13. Conclusiones

Cuando se me propuso la realización de este proyecto como TFG, pensaba que me costaría sacarlo adelante. El proyecto requería ciertos conocimientos de física de los que al principio carecía. Por esta razón, fue necesario invertir tiempo en comprender aspectos básicos de la física encargada de resolver la dinámica de los amarres, sin embargo, la ejecución del trabajo ha ido mejor de lo esperado y se han conseguido alcanzar los objetivos.

A nivel técnico, he ampliado mis conocimientos en los siguientes aspectos:

- En el lenguaje de programación C++. Al iniciar este proyecto no tenía muchos conocimientos sobre este lenguaje, pero al hacer uso de él para el desarrollo multiplataforma de una aplicación real, he mejorado en gran medida mis conocimientos en dicho lenguaje.

- También he podido adquirir conocimientos nuevos sobre física, relacionados con la dinámica de fluidos. Estos conocimientos, me han permitido llevar a cabo la realización del trabajo de una forma satisfactoria. Por ello, este proyecto me ha ayudado a aprender, tanto en aspectos de informática, como de un campo que anteriormente desconocía como es la física aplicada en investigación.
- He podido llevar a la práctica los conceptos teóricos de la aplicación de una metodología de desarrollo software, lo que me ha ayudado a mejorar en la toma de requisitos y en el diseño del software al llevarlo a cabo en un proyecto real.

Como conclusión personal, a pesar de los posibles inconvenientes, estoy orgulloso de haber realizado con éxito este trabajo y ofrecer una librería de amarres acoplada con DualSPHysics. Se ha demostrado que ambos sistemas software funcionan correctamente, lo que permitirá su uso en multitud de proyectos de investigación. Esto es un estímulo positivo al conseguir que el trabajo realizado sea un aporte para el campo de la investigación, facilitando la simulación de experimentos con objetos flotantes amarrados. Finalmente, la realización de este proyecto me ha permitido iniciarme en el mundo de la investigación, formando parte del grupo EPhysLab. Esto supone que he podido trabajar con compañeros especializados en otras ramas distintas a la informática y he podido aprender de todos ellos.

14. Vías de trabajo futuro

Una de las posibles ampliaciones, sería la de implementar amarres con varias líneas, dónde algunas de ellas, no estén conectadas directamente al objeto flotante. Se puede ver en la Figura 14-1.

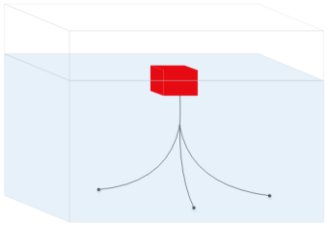


Figura 14-1: Líneas de amarre con conexiones intermedias

También sería interesante poder configurar objetos flotantes amarrados a distintas profundidades. Actualmente se realizan los cálculos sobre una profundidad global para todos los amarres de la simulación.

Una de las mejoras en codificación, sería aplicar polimorfismo en los objetos *Connection* en función del tipo que sea cada conexión (fija, móvil o intermedia). De esta forma se reemplazaría el identificador numérico que diferencia el tipo de conexión.

Implementar un sistema de *logs*. Este sistema sería muy útil para generar un fichero que almacene los *warnings* o errores que se puedan producir a lo largo de la ejecución además de mostrarlos por consola. Lo cual permitiría al usuario ver en detalle cualquier problema detectado que pudiera pasar desapercibido durante la ejecución de la librería.

Realizar la compilación de MoorDyn+ como una librería dinámica. Al estar compilada de forma estática es necesario volver a compilar DualSPHysics cada vez que se hace un cambio en MoorDyn+. Esto se debe a que las librerías estáticas se “copian” en el programa en el proceso de compilación. Con las librerías dinámicas el ejecutable accede a ellas en la ruta en la donde se encuentre.

15. Referencias

[1] A.J.C. Crespo, J.M. Domínguez, B.D. Rogers, M. Gómez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro, O. García-Feal. (2015). "DualSPHysics: Open-source parallel CFD solver based on Smoothed Particle Hydrodynamics (SPH)". Computer Physics Communications 187, 204-216.

[2] M. Hall, MoorDyn User's Guide, 2019. [Online]: www.matt-hall.ca/moordyn.

[3] J. Palm and C. Eskilsson, MooDy User's Guide, 2019. [Online]: https://github.com/johannep/moodyAPI

[4] I. Sommerville, *Software engineering*, 10th Ed. Boston: Pearson Education, 2016.

[5] G. Booch, J. Rumbaugh and I. Jacobson, *El Lenguaje unificado de modelado*, 2nd Ed. Madrid, España: Addison-Wesley, D.L. 2006.

[6] B. Stroustrup, *The C++ Programing Language*, 4th Ed. NJ: Addison-Wesley, 2013.

[7] R. M. Stallman, *Using Gcc: The Gnu Compiler Collection Reference*. Boston: GNU Press., 2003.

[8] R. Mecklenburg, *Managing Projects with GNU Make* 3rd Ed. Sebastopol, California: O'Reilly Media, 2004.

[9] R. M. Stallman, R. Pesch, S. Shebs, et al., *Debugging with GDB*. Boston: GNU Press., 2003.

[10] M. Cañete-Güeto, M. A. Fernández-Ruiz, E. Hernández-Montes, "Catenary aproximation method in the antifunicular structures design" *Informes de la Construcción*, vol. 66, pp. 1-11, 2014

Anexos

Iván Martínez Estévez

16. Anexos

16.1. Descripción de MoorDyn

MoorDyn es un software de código abierto que resuelve la dinámica de un sistema de amarres para objetos flotantes. Se ha desarrollado para ser acoplado con otros sistemas software (M. Hall, 2017).

Las líneas de amarre se discretizan como masas puntuales conectadas por modelos lineales de amortiguación para proporcionar elasticidad en la dirección axial. El modelo también resuelve la fricción del amarre con el fondo y permite conectar múltiples líneas de amarre. En la Figura 16-1 se muestran estas funcionalidades. Con una formulación simple, MoorDyn ha demostrado ser computacionalmente eficiente y preciso para simular captadores de energía que usan amarres (Hall y Goupee, 2015; Vissio et al., 2015).

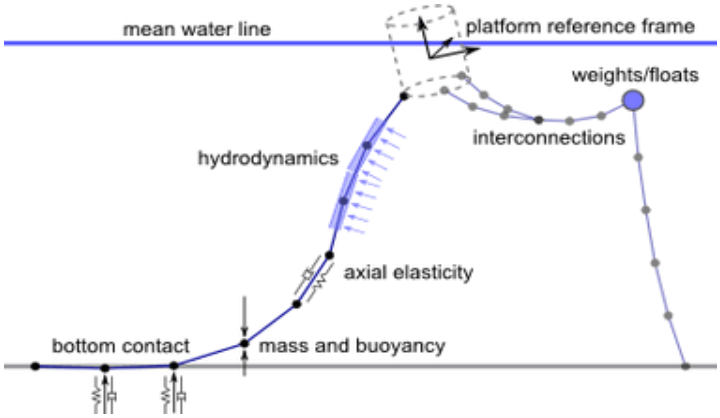


Figura 16-1: Modelo de MoorDyn

Fuente: http://www.matt-hall.ca/moordyn.html

El fichero de entrada utilizado para la configuración es del tipo ***Mooring/lines.txt***. En este fichero de entrada se le indican los parámetros que se deseen para llevar a cabo la simulación: Condiciones del entorno (gravedad, profundidad del fluido, densidad del fluido, etc.), características de los amarres (número de líneas, posiciones de las conexiones, longitud de la línea, etc.), el tipo de magnitudes deseadas para la salida de datos, o tiempo de simulación entre otros. Un ejemplo de fichero de entrada se puede observar en la Figura 16-2.

```
---------------------- MoorDyn Input File ------------------------------------
MoorDyn input file of the mooring system for OC3-Hywind
FALSE    Echo      - echo the input file data (flag)
---------------------- LINE TYPES --------------------------------------------
1        NTypes    - number of LineTypes
Name     Diam      MassDen      EA     BA/-zeta   Can   Cat   Cdn   Cdt
(-)       (m)      (kg/m)       (N)     (N-s/-)   (-)   (-)   (-)   (-)
main     0.09      77.7066   384.243E6   -0.8     1.0   0.0   1.6   0.1
---------------------- CONNECTION PROPERTIES ---------------------------------
6       NConnects - number of connections including anchors and fairleads
Node    Type      X         Y         Z        M     V      FX   FY   FZ   CdA  Ca
(-)     (-)      (m)       (m)       (m)      (kg)  (m^3)  (kN) (kN) (kN) (m^2) (-)
1      fixed    853.87     0.0     -320.0     0     0      0    0    0    0    0
2      fixed   -426.94    739.47   -320.0     0     0      0    0    0    0    0
3      fixed   -426.94   -739.47   -320.0     0     0      0    0    0    0    0
4      vessel     5.2       0.0     -70.0     0     0      0    0    0    0    0
5      vessel    -2.6       4.5     -70.0     0     0      0    0    0    0    0
6      vessel    -2.6      -4.5     -70.0     0     0      0    0    0    0    0
---------------------- LINE PROPERTIES ---------------------------------------
3        NLines    - number of line objects
Line    LineType  UnstrLen  NumSegs   NodeAnch  NodeFair  Flags/Outputs
(-)       (-)       (m)       (-)       (-)       (-)         (-)
1         main     902.2      20        1         4           p
2         main     902.2      20        2         5           -
3         main     902.2      20        3         6           -
---------------------- SOLVER OPTIONS ----------------------------------------
0.001    dtM       - time step to use in mooring integration (s)
3.0e6    kBot      - bottom stiffness (Pa/m)
3.0e5    cBot      - bottom damping (Pa-s/m)
320      WtrDpth   - water depth (m)
1.0      dtIC      - time interval for analyzing convergence during IC gen (s)
60.0     TmaxIC    - max time for IC gen (s)
4.0      CdScaleIC - factor by which to scale drag coefficients during dynamic relaxation (-)
0.001    threshIC  - threshold for IC convergence (-)
------------------------ OUTPUTS ----------------------------------------------
FairTen1
FairTen2
FairTen3
AnchTen3
L2N4pX
END
------------------------ need this line ----------------------------------------
```

Figura 16-2: Fichero de entrada de datos de MoorDyn

Fuente: Guía de Usuario de MoorDyn www.matt-hall.ca/moordyn

16.2. Nuevas funciones añadidas

A continuación, se muestran las nuevas funciones añadidas en la *Iteración 8*. Se han añadido a posteriori del análisis de requisitos y se pueden observar en la Figura 16-3. En la Figura 16-4 se muestra el diagrama de clases de MoorDyn+ con estas funciones incluidas.

```
//==============================================================================
///Returns the number of lines of this Mooring (floating_id)
//==============================================================================
unsigned GetnLines(const unsigned floating_id = 0);

//==============================================================================
///Returns the number of segments of a Line (LineNum) of this Mooring (floating_id)
//==============================================================================
unsigned GetSegsCount(const unsigned LineNum, const unsigned floating_id = 0);

//==============================================================================
///Returns the Tension of the Line (LineNum) of this Mooring (floating_id)
//==============================================================================
double GetFairTen(const unsigned LineNum, const unsigned floating_id = 0);

//==============================================================================
///Returns the position of node (NodeNum) of the Line (LineNum) of this Mooring
/// (floating_id)
//==============================================================================
int GetNodePos(const unsigned LineNum, const int NodeNum, double pos[3],
                                        const unsigned floating_id = 0);

//==============================================================================
///Returns the number of Moorings
//==============================================================================
unsigned GetnMoorings() { return nMoorings; };

//==============================================================================
///Returns the mooring reference of this Mooring (floating_id)
//==============================================================================
unsigned GetMooringReference(const unsigned floating_id = 0);
```

Figura 16-3: Nuevas funciones añadidas

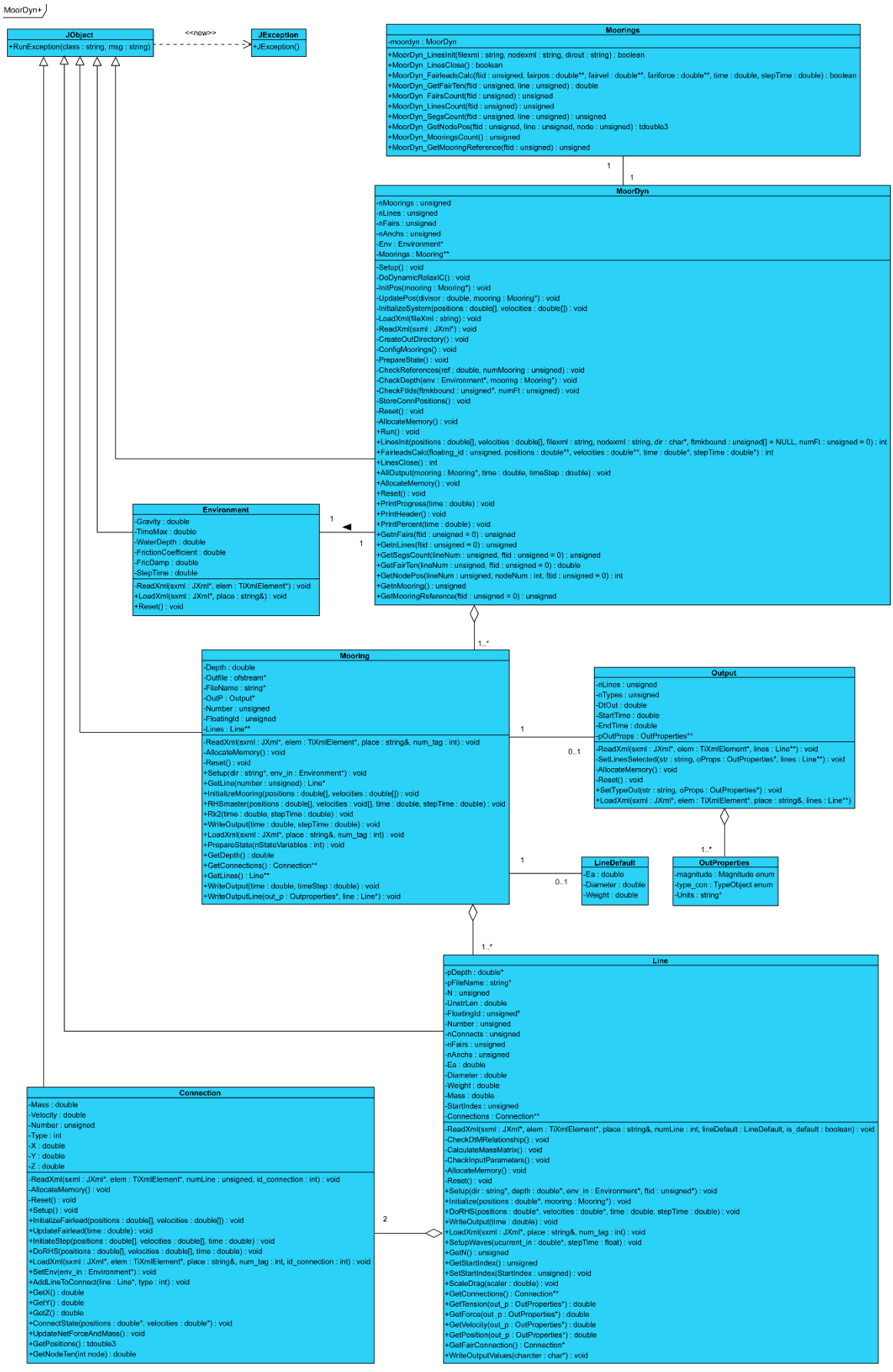


Figura 16-4: Diagrama de Clases final de MoorDyn+

16.3. Acoplamiento a nivel de proceso

En este apartado se muestra cómo interactúan DualSPHysics y MoorDyn+ a nivel de proceso. MoorDyn+ recibe el identificador de un objeto flotante, su posición, velocidad y tiempo y el intervalo de tiempo.

MoorDyn+ realiza los cálculos de fuerzas ejercidas por los amarres sobre el objeto flotante recibido y se los devuelve a DualSPHysics que actualiza las posiciones y velocidades de las partículas del objeto flotante. Este proceso se repite hasta completar la simulación.

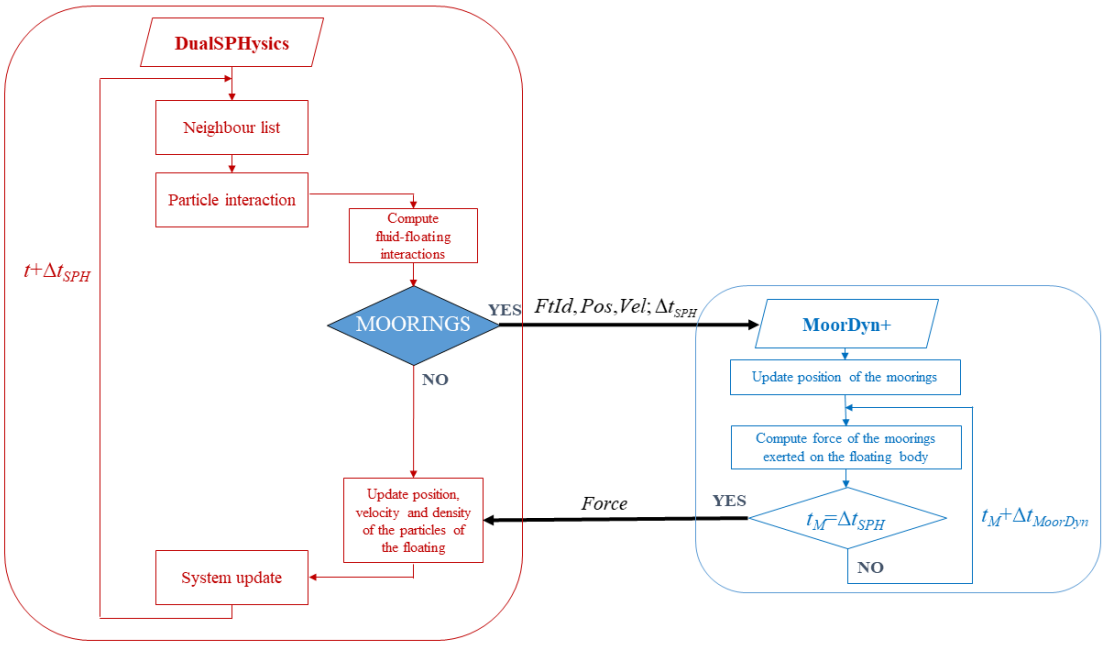


Figura 16-5: Proceso de comunicación entre DualSPHysics y MoorDyn+

16.4. Listado del control de excepciones

En este apartado se muestran todas las excepciones que se han incluido para garantizar que el programa se detiene cuando detecta un error. Se agrupan por clases (fichero *.cpp*). Algunas de las excepciones reemplazan mensajes de texto mostrados anteriormente por la librería MoorDyn.

1. Clase Connection

- **Función:** LoadXml
- **Línea:** 77
- **Descripción:** Se lanza cuando no se encuentra en el XML el nodo que se le pasa como parámetro.

```
RunException("Connection::LoadXml","Cannot find the element \'" + place + "\'.");
```

Figura 16-6: Excepción 1

- **Función:** ReadXml
- **Línea:** 95
- **Descripción:** Se lanza cuando en la etiqueta del tipo de *Connection* se inserta un tipo no válido.

```
string tex = "Cannot find the element \'"; tex = tex + sxml->ErrGetFileRow(elec) + "\'.";
RunException("Connection::ReadXml", tex);
```

Figura 16-7: Excepción 2

- **Función:** InitializeFairlead
- **Línea:** 195
- **Descripción:** Se lanza cuando se intenta ejecutar el método con un *Connection* de tipo distinto de f*airlead (type=1)*

```
string tex = "Error: wrong connection Type given. Something's not right.";
RunException("Connection::InitializeFairlead", tex);
```

Figura 16-8: Excepción 3

- **Función:** InitializeConnect
- **Línea:** 214
- **Descripción:** Se lanza cuando se intenta ejecutar el método con un *Connection* de tipo distinto de *connect (type=2)*

```
string tex = "Error: wrong connection Type given. Something's not right.";
RunException("Connection::InitializeConnect", tex);
```

Figura 16-9: Excepción 4

- **Función:** DoRHS
- **Línea:** 318
- **Descripción:** Se lanza cuando se intenta ejecutar el método con un *Connection* de tipo distinto de *connect (type=2)*

```
string tex = "Error: wrong connection Type given. Something's not right.";
RunException("Connection::DoRHS", tex);
```

Figura 16-10: Excepción 5

- **Función:** UpdateFairlead
- **Línea:** 354
- **Descripción:** Se lanza cuando se intenta ejecutar el método con un *Connection* de tipo distinto de f*airlead (type=1)*

```
string tex = "Error: wrong connection Type given. Something's not right.";
RunException("Connection::UpdateFairlead", tex);
```

Figura 16-11: Excepción 6

2. Clase Environment

- **Función:** LoadXml
- **Línea:** 46
- **Descripción:** Se lanza cuando no se encuentra en el XML el nodo que se le pasa como parámetro.

```
RunException("Environment::LoadXml", "Cannot find the element \'" + place + "\'.");
```

Figura 16-12: Excepción 7

3. Clase Line

- **Función:** AllocateMemory
- **Línea:** 114
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero.

```
try {
    Connections = new Connection*[nConnects];
    pFileName = new string;
    Env = new Environment;
    AnchConnect = new Connection;
    FairConnect = new Connection;
    pDepth = new double;
    FloatingId = new unsigned;
}
catch (const std::bad_alloc) {
    RunException("Line::AllocateMemory", "Could not allocate the requested memory.");
}
```

Figura 16-13: Excepción 8

- **Función:** LoadXml
- **Línea:** 165
- **Descripción:** Se lanza cuando no se encuentra en el XML el nodo que se le pasa como parámetro.

```
RunException("Line::LoadXml", "Cannot find the element \'" + place + "\'.");
```

Figura 16-14: Excepción 9

- **Función:** CheckChannels
- **Línea:** 252
- **Descripción:** Se lanza cuando se inserta en el XML un tipo de salida no válido para los nodos de cada línea.

```
string tex = "Value of ouputsFlags tag isn't correctly. Checks the options. \'";
tex = tex + sxml->ErrGetFileRow(elel) + "\'.";
RunException("Line::CheckChannels", tex);
```

Figura 16-15: Excepción 10

- **Función:** CheckInputParameters
- **Línea:** 270
- **Descripción:** Se lanza cuando la longitud de la línea es menor que la distancia entre los extremos.

```
string tex = "Error: The length of the line " + to_string(Number)
    + " must be at least of " + to_string(distance) + "m.";
RunException("Line::CheckInputParameters", tex);
```

Figura 16-16: Excepción 11

- **Función:** Initialize
- **Línea:** 431
- **Descripción:** Se lanza cuando no se puede abrir el fichero de salida.

```
RunException("Line::Initialize", "Cannot open file ouput \'" + *pFileName + "\'\n");
```

Figura 16-17: Excepción 12

- **Función:** Initialize
- **Línea:** 433
- **Descripción:** Se lanza cuando no se creó correctamente el fichero de salida.

```
RunException("Line::Initialize", "Cannot create file ouput \'" + *pFileName + "\'\n");
```

Figura 16-18: Excepción 13

- **Función:** Initialize
- **Línea:** 447
- **Descripción:** Se lanza cuando se crea la conexión fija de la línea a una distancia de la superficie más grande que la profundidad establecida.

```
string tex = "Error: water depth is shallower than Line " + to_string(Number)
    + " anchor in Mooring " + to_string(mooring->GetNumber());
RunException("Line::Initialize", tex);
```

Figura 16-19: Excepción 14

- **Función:** DoRHS
- **Línea:** 646
- **Descripción:** Se lanza cuando se detecta un *Not-a-Number value* al calcular el estiramiento de la línea.

```
string tex = "Error:  NaN value detected in MoorDyn state at dynamic relaxation time "
    + to_string(time) + " s. Probably, the EA value for line " + to_string(Number)
    + " of mooring " + to_string(*FloatingId) + " is very high.";
RunException("Line::DoRHS", tex);
```

Figura 16-20: Excepción 15

- **Función:** DoRHS
- **Línea:** 655
- **Descripción:** Se lanza cuando al aplicar los cálculos, la distancia entre dos nodos (longitud del segmento *i*) es igual a 0.

```
string tex = "Error: lstr[" + to_string(i) + "] == "
    + to_string(lstr[i]) + ". lstr_squared= " + to_string(lstr_squared)
    + " sqrt= " + to_string(sqrt(lstr_squared)) + " Cannot divide by 0. Check line Number "
    + to_string(Number) + " of mooring " + to_string(*FloatingId) + ".";
RunException("Line::DoRHS", tex);
```

Figura 16-21: Excepción 16

- **Función:** DoRHS
- **Línea:** 810
- **Descripción:** Se lanza cuando al calcular la velocidad del nodo a lo largo del suelo, se obtiene un valor infinito. Se calcula cuando el nodo está “apoyado” en el fondo.

```
string tex = "Error: BottomVel == +/- inf. Cannot divide by infinite. Check line Number "
    + to_string(Number) + ".";
RunException("Line::DoRHS", tex);
```

Figura 16-22: Excepción 17

- **Función:** DoRHS
- **Línea:** 849
- **Descripción:** Se lanza cuando al calcular la fuerza en cada nodo, se obtiene un *Not-a-Number value*.

```
string tex = "Error: Fnet[" + to_string(i) + "][" + to_string(J)
    + "] NaN value detected in MoorDyn state at dynamic relaxation time "
    + to_string(time) + " s.";
RunException("Line::DoRHS", tex);
```

Figura 16-23: Excepción 18

- **Función:** WriteOutput
- **Línea:** 914
- **Descripción:** Se lanza cuando no se puede abrir el fichero de salida.

```
RunException("Line::WriteOutput", "Cannot open file ouput \'" + *pFileName + "\'\n");
```

Figura 16-24: Excepción 19

- **Función:** WriteOutput
- **Línea:** 916
- **Descripción:** Se lanza cuando no se creó correctamente el fichero de salida.

```
RunException("Line::WriteOutput", "Cannot create file ouput \'" + *pFileName + "\'\n");
```

Figura 16-25: Excepción 20

4. Clase MoorDyn

- **Función:** AllocateMemory
- **Línea:** 84
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero.

```
try {
    Moorings = new Mooring*[nMoorings];
}
catch (const std::bad_alloc) {
    RunException("MoorDyn::AllocateMemory", "Could not allocate the requested memory.");
}
```

Figura 16-26: Excepción 21

- **Función:** GetPtrToPtr
- **Línea:** 100
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero.

```
try {
    pToPtr = new double*[sizex];
}
catch (const std::bad_alloc) {
    RunException("MoorDyn::GetPtrToPtr", "Could not allocate the requested memory.");
}
```

Figura 16-27: Excepción 22

- **Función:** GetPtrToInt
- **Línea:** 121
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero.

```
try {
    pToInt = new int[size];
}
catch (const std::bad_alloc) {
    RunException("MoorDyn::GetPtrToInt", "Could not allocate the requested memory.");
}
```

Figura 16-28: Excepción 23

- **Función:** GetPtrToDouble
- **Línea:** 137
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero.

```
try {
    pToDbl = new double[size];
}
catch (const std::bad_alloc) {
    RunException("MoorDyn::GetPtrToDouble", "Could not allocate the requested memory.");
}
```

Figura 16-29: Excepción 24

- **Función:** LoadXml
- **Línea:** 162
- **Descripción:** Se lanza cuando no existe el fichero de entrada que se le indica.

```
RunException("MoorDyn::LoadXml", "MoorDyn configuration was not found.", FileXml);
```

Figura 16-30: Excepción 25

- **Función:** LoadXml
- **Línea:** 165
- **Descripción:** Se lanza cuando no se encuentra en el XML el nodo raíz *moordyn*.

```
RunException("MoorDyn::LoadXml", "Cannot find the element \'" + RootNodeXml + "\'.");
```

Figura 16-31: Excepción 26

- **Función:** CheckReferences
- **Línea:** 240
- **Descripción:** Se lanza cuando el programa detecta que el usuario introdujo identificadores de amarres duplicados.

```
string tex = "Error: The ID of mooring" + to_string(ref) + " + is duplicated .";
RunException("MoorDyn::CheckReferences", tex);
```

Figura 16-32: Excepción 27

- **Función:** GetLines
- **Línea:** 275
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero. Esta función devuelve todas las líneas del programa.

```
try {
    lines = new Line*[nLines];
}
catch (const std::bad_alloc) {
    RunException("MoorDyn::GetLines", "Could not allocate the requested memory.");
}
```

Figura 16-33: Excepción 28

- **Función:** GetLines(*mooring*)
- **Línea:** 302
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero. Esta función devuelve las líneas del *mooring* pasado por parámetro.

```
try {
    lines = new Line*[nL];
}
catch (const std::bad_alloc) {
    RunException("MoorDyn::GetLines", "Could not allocate the requested memory.");
}
```

Figura 16-34: Excepción 29

- **Función:** GetConnections
- **Línea:** 323
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero. Esta función devuelve todas las conexiones del programa.

```
try {
    connections = new Connection*[nConnects];
}
catch (const std::bad_alloc) {
    RunException("MoorDyn::GetConnections", "Could not allocate the requested memory.");
}
```

Figura 16-35: Excepción 30

- **Función:** GetConnections(*mooring*)
- **Línea:** 351
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero. Esta función devuelve todas las conexiones de un *mooring* en concreto.

```
try {
    connections = new Connection*[nConn_in];
}
catch (const std::bad_alloc) {
    RunException("MoorDyn::GetConnections", "Could not allocate the requested memory.");
}
```

Figura 16-36: Excepción 31

- **Función:** GetConnections(*type*)
- **Línea:** 380
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero. Esta función devuelve todas las conexiones de un tipo en concreto.

```
try {
    connections = new Connection*[nConnects];
}
catch (const std::bad_alloc) {
    RunException("MoorDyn::GetConnections", "Could not allocate the requested memory.");
}
```

Figura 16-37: Excepción 32

- **Función:** CheckFtIds
- **Línea:** 585
- **Descripción:** Se lanza cuando el software externo envía identificadores de objetos flotantes que no coinciden con los de los amarres.

```
RunException("MoorDyn::CheckFtIds", "Error: Cannot find someone Mooring with id " + to_string(ids[i]) + ".");
```

Figura 16-38: Excepción 33

- **Función:** LinesInit
- **Línea:** 596
- **Descripción:** Se lanza cuando el software externo envía identificadores de objetos flotantes que no coinciden con los de los amarres.

```
RunException("MoorDyn::LinesInit", "Error: The number of driven-object floatings doesn't be 0.");
```

Figura 16-39: Excepción 34

- **Función:** FairleadsCalc
- **Línea:** 632
- **Descripción:** Se lanza cuando el número de iteraciones para realizar el Método de Aproximación Catenaria es menor que 1. (iteraciones = 1/ dtM).

```
string tex = "Error: dtC is less than dtM. ("
    + to_string(dtC) + " < " + to_string(Env->GetDtM0()) + ").";
RunException("MoorDyn::FairleadsCalc", tex);
```

Figura 16-40: Excepción 35

- **Función:** CheckMooring
- **Línea:** 654
- **Descripción:** Se lanza cuando el software externo envía el número de un amarre fuera de los límites establecidos.

```
string tex = "Error: The range of Moorings is ["
    + to_string(0) + "," + to_string(nMoorings - 1)
    + "]. Sent: " + to_string(ftid);
RunException("MoorDyn::CheckMooring", tex);
```

Figura 16-41: Excepción 36

- **Función:** CheckMooring
- **Línea:** 658
- **Descripción:** Se lanza cuando no existe ningún amarre con el número que se le pasó.

```
string tex = "There isn't a mooring with id " + to_string(ftid) + ".";
RunException("MoorDyn::CheckMooring", tex);
```

Figura 16-42: Excepción 37

- **Función:** CheckLine
- **Línea:** 670
- **Descripción:** Se lanza cuando no existe ninguna línea con el número que se le pasó.

```
string tex = "There isn't a line with number " + to_string(numLine) + ".";
RunException("MoorDyn::CheckLine", tex);
```

Figura 16-43: Excepción 38

- **Función:** GetNodePos
- **Línea:** 714
- **Descripción:** Se lanza cuando el nodo que se le pasa no está dentro de los límites.

```
string tex = "Error: The range of the Nodes is ["
    + to_string(0) + "," + to_string(line->GetN())
    + "]. Sent: " + to_string(NodeNum)
    + " of Line " + to_string(LineNum) + ".";
RunException("MoorDyn::GetNodePos", tex);
```

Figura 16-44: Excepción 39

5. Clase Mooring

- **Función:** AllocateMemory
- **Línea:** 72
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero.

```
try {
    Lines = new Line*[nLines];
    Outfile = new ofstream;
    OutP = new Output;
    pFileName = new string;
    Env = new Environment;

}
catch (const std::bad_alloc) {
    RunException("Mooring::AllocateMemory", "Could not allocate the requested memory.");
}
```

Figura 16-45: Excepción 40

- **Función:** AllocateMemoryIntegration
- **Línea:** 102
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero.

```
try {
    States = new double[size];
    f0 = new double[size];
    f1 = new double[size];
    xt = new double[size];
}
catch (const std::bad_alloc) {
    RunException("Mooring::AllocateMemoryIntegration", "Could not allocate the requested memory.");
}
```

Figura 16-46: Excepción 41

- **Función:** LoadXml
- **Línea:** 117
- **Descripción:** Se lanza cuando no existe el fichero de entrada que se le indica.

```
RunException("Mooring::LoadXml", "Cannot find the element \'" + place + "\'.");
```

Figura 16-47: Excepción 42

- **Función:** ReadXml
- **Línea:** 158
- **Descripción:** Se lanza cuando no se ha definido ninguna línea para un amarre.

```
string tex = "Cannot find the element line \'"; tex = tex + sxml->ErrGetFileRow(ele) + "\'.";
RunException("Mooring::ReadXml", tex);
```

Figura 16-48: Excepción 43

- **Función:** GetConnections
- **Línea:** 224
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero. Esta función devuelve todas las conexiones del programa.

```
try {
    connections = new Connection*[nConn_in];
}
catch (const std::bad_alloc) {
    RunException("Mooring::GetConnections", "Could not allocate the requested memory.");
}
```

Figura 16-49: Excepción 44

- **Función:** GetConnections(*type*)
- **Línea:** 252
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero. Esta función devuelve todas las conexiones de un tipo en concreto.

```
try {
    connections = new Connection*[nConn_in];
}
catch (const std::bad_alloc) {
    RunException("Mooring::GetConnections", "Could not allocate the requested memory.");
}
```

Figura 16-50: Excepción 45

- **Función:** Setup
- **Línea:** 306
- **Descripción:** Se lanza cuando no se puede abrir el fichero de salida.

```
RunException("Mooring::Setup", "Cannot open file ouput \'" + *pFileName + "\'\n");
```

Figura 16-51: Excepción 46

- **Función:** Setup
- **Línea:** 308
- **Descripción:** Se lanza cuando no se creó correctamente el fichero de salida.

```
RunException("Mooring::Setup", "Cannot create file ouput \'" + *pFileName + "\'\n");
```

Figura 16-52: Excepción 47

- **Función:** Rk2
- **Línea:** 587
- **Descripción:** Se lanza cuando después de realizar la integración y los cálculos de fuerzas, se detecta un *Not-a-Number value*.

```
string tex = "Error: NaN value detected in MoorDyn state at dynamic relaxation time "
    + to_string(*t0) + " s.";
RunException("Mooring::Rk2", tex);
```

Figura 16-53: Excepción 48

6. Clase Output

- **Función:** AllocateMemory
- **Línea:** 81
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero.

```
try {
    pOutProps = new OutProperties*[nTypes];
}
catch (const std::bad_alloc) {
    RunException("Output::AllocateMemory", "Could not allocate the requested memory.");
}
```

Figura 16-54: Excepción 49

- **Función:** LoadXml
- **Línea:** 92
- **Descripción:** Se lanza cuando no existe el fichero de entrada que se le indica.

```
RunException("Output::LoadXml", "Cannot find the element \'" + place + "\'.");
```

Figura 16-55: Excepción 50

7. Clase Output

- **Función:** AllocateMemory
- **Línea:** 38
- **Descripción:** Bloque *try-catch*. Se lanza cuando no se reserva correctamente la memoria para algún puntero.

```
try {
    Lines = new Line*[nLines];
}
catch (const std::bad_alloc) {
    RunException("OutProperties::AllocateMemory", "Could not allocate the requested memory.");
}
```

Figura 16-56: Excepción 51

16.5. Estructura del directorio DSPH-MoorDyn+

En la Figura 16-57 se muestra como está estructurado el directorio que se suministra para probar la librería. El código fuente de MoorDyn+ se encuentra en la ruta *src/source/Source_MoorDyn*.

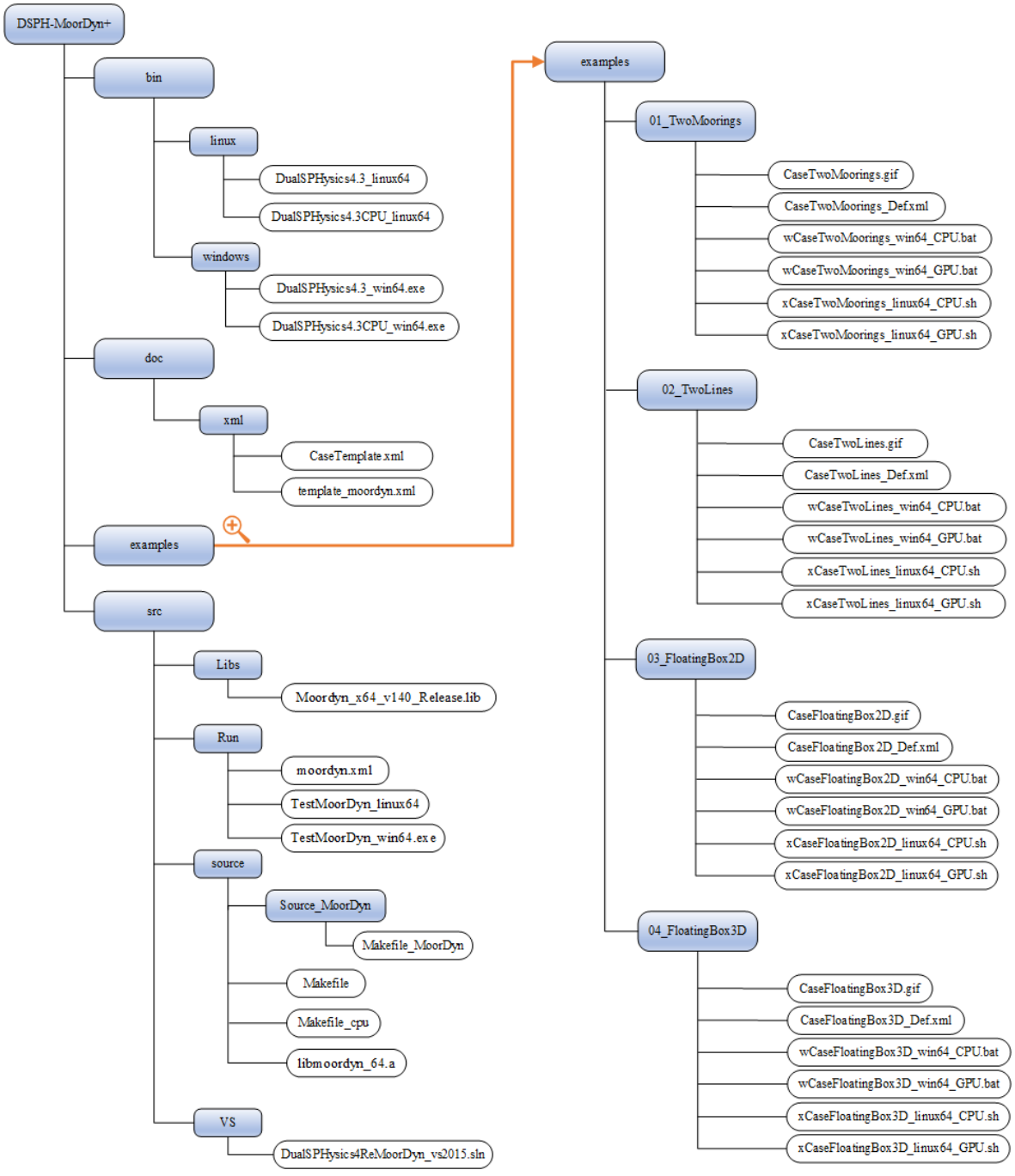


Figura 16-57: Estructura del directorio DSPH-MoorDyn+

16.6. Contenido de los Scripts de ejecución

Los scripts de Windows son los que tienen extensión *.bat* mientras que los de Linux tienen la extensión *.sh*. Se utilizan para lanzar los ejecutables de DualSPHysics con MoorDyn+ acoplado e incluyen ciertos parámetros de ejecución, como por ejemplo directorio de salida, fichero de entrada o directorio donde se encuentran los ejecutables. Ambos tipos de scripts tienen la misma información, aunque distinta sintaxis. Por esta razón solo se explicará el contenido de los scripts de Windows. Existe información detallada de las opciones que admiten los ejecutables en el directorio *DSPH-MoorDyn+/doc/help*.

wCaseTwoMoorings_win64_CPU.bat

```
@echo off

rem "name" and "dirout" are named according to the testcase

set name=CaseTwoMoorings
set dirout=%name%_out
set diroutdata=%dirout%\data

rem "executables" are renamed and called from their directory

set dirbin=../../bin/windows
set gencase="%dirbin%/GenCase4_win64.exe"
set dualsphysicscpu="%dirbin%/DualSPHysics4.3CPU_win64.exe"
set dualsphysicsgpu="%dirbin%/DualSPHysics4.3_win64.exe"
set boundaryvtk="%dirbin%/BoundaryVTK4_win64.exe"
set partvtk="%dirbin%/PartVTK4_win64.exe"
set partvtkout="%dirbin%/PartVTKOut4_win64.exe"
set measuretool="%dirbin%/MeasureTool4_win64.exe"
set computeforces="%dirbin%/ComputeForces4_win64.exe"
set isosurface="%dirbin%/IsoSurface4_win64.exe"
set flowtool="%dirbin%/FlowTool4_win64.exe"
set floatinginfo="%dirbin%/FloatingInfo4_win64.exe"
```

Figura 16-58: Contenido del script para Windows

- **name:** Indica el nombre del fichero XML de entrada
- **dirout:** Indica el nombre del directorio de salida donde se almacenarán todos los ficheros resultantes.
- **diroutdata:** Indica el directorio donde se almacenarán los archivos binarios que contienen la información de las partículas en un instante de tiempo.
- **dirbin:** Indica el directorio donde se encuentran los ejecutables.
- **dualsphysicscpu:** Indica el nombre del ejecutable de DualSPHysics para CPU.
- **dualsphysicsgpu:** Indica el nombre del ejecutable de DualSPHysics para GPU.

Hay que tener en cuenta que ambos scripts eliminan el directorio de salida cada vez que se ejecutan. Se muestran las líneas que realizan esta acción en la Figura 16-59 si se quieren conservar los archivos de una simulación anterior habrá que cambiarle el nombre al directorio de salida.

```
rem "dirout" is created to store results or it is removed if it already exists

if exist %dirout% rd /s /q %dirout%
mkdir %dirout%
if not "%ERRORLEVEL%" == "0" goto fail
mkdir %diroutdata%
```

Figura 16-59: Borrado previo del directorio de salida

El comando encargado de ejecutar DualSPHysics es el de la Figura 16-60.

```
36 %dualsphysicscpu% -cpu %dirout%/%name% %dirout% -dirdataout data -svres
37 if not "%ERRORLEVEL%" == "0" goto fail
```

Figura 16-60: Ejecución de DualSPHysics

- **-cpu**: indica que tipo de ejecución que se desea (GPU o CPU). La opción –gpu sólo es válida en el ejecutable que incluye el código para GPU.
- Nombre del fichero de entrada.
- Directorio de salida
- **-dirdataout** data: Directorio donde se almacenan los ficheros binarios.
- **-svres**: Genera un fichero con un resumen de la ejecución.

16.7. XML Schema de MoorDyn+

XML Schema es un lenguaje utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de forma muy precisa. Se ha elaborado un XML Schema que se usará para validar los ficheros de configuración admitidos por MoorDyn+ de forma automática.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="moordyn">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="solverOptions"/>
        <xs:element maxOccurs="unbounded" ref="mooring"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="solverOptions">
    <xs:complexType mixed="true">
      <xs:choice minOccurs="1" maxOccurs="unbounded">
        <xs:element ref="cBot"/>
        <xs:element ref="cdScaleIC"/>
        <xs:element ref="dtIC"/>
        <xs:element ref="dtM"/>
        <xs:element ref="fricDamp"/>
        <xs:element ref="frictionCoefficient"/>
        <xs:element ref="gravity"/>
        <xs:element ref="kBoot"/>
        <xs:element ref="statDynFricScale"/>
        <xs:element ref="threshIC"/>
        <xs:element ref="timeMax"/>
        <xs:element ref="tmaxIC"/>
        <xs:element ref="waterDepth"/>
        <xs:element ref="waveKin"/>
        <xs:element ref="writeUnits"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="cBot">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:double"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="cdScaleIC">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="dtIC">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="dtM">
```

```
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="fricDamp">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="frictionCoefficient">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="gravity">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="kBoot">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:double"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="statDynFricScale">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="threshIC">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="timeMax">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="tmaxIC">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="waterDepth">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="waveKin">
    <xs:complexType>
```

```
    <xs:attribute name="comment" use="required"/>
    <xs:attribute name="value" use="required" type="xs:integer"/>
  </xs:complexType>
</xs:element>
<xs:element name="writeUnits">
  <xs:complexType>
    <xs:attribute name="comment" use="required"/>
    <xs:attribute name="value" use="required" type="xs:NCName"/>
  </xs:complexType>
</xs:element>
<xs:element name="mooring">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="line"/>
      <xs:element ref="linedefault"/>
      <xs:element ref="output"/>
    </xs:choice>
    <xs:attribute name="ref" use="required" type="xs:integer"/>
  </xs:complexType>
</xs:element>
<xs:element name="line">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="fixconnection"/>
      <xs:element ref="length"/>
      <xs:element ref="segments"/>
      <xs:element ref="vesselconnection"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
<xs:element name="fixconnection">
  <xs:complexType>
    <xs:attribute name="x" use="required" type="xs:decimal"/>
    <xs:attribute name="y" use="required" type="xs:decimal"/>
    <xs:attribute name="z" use="required" type="xs:decimal"/>
  </xs:complexType>
</xs:element>
<xs:element name="length">
  <xs:complexType>
    <xs:attribute name="comment" use="required"/>
    <xs:attribute name="value" use="required" type="xs:decimal"/>
  </xs:complexType>
</xs:element>
<xs:element name="segments">
  <xs:complexType>
    <xs:attribute name="value" use="required" type="xs:integer"/>
  </xs:complexType>
</xs:element>
<xs:element name="vesselconnection">
  <xs:complexType>
    <xs:attribute name="x" use="required" type="xs:decimal"/>
    <xs:attribute name="y" use="required" type="xs:decimal"/>
    <xs:attribute name="z" use="required" type="xs:decimal"/>
  </xs:complexType>
</xs:element>
<xs:element name="linedefault">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="ea"/>
      <xs:element ref="diameter"/>
      <xs:element ref="massDenInAir"/>
```

```
        <xs:element ref="ba"/>
        <xs:element ref="can"/>
        <xs:element ref="cat"/>
        <xs:element ref="cdn"/>
        <xs:element ref="cdt"/>
        <xs:element ref="ouputsFlags"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ea">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:double"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="diameter">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:double"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="massDenInAir">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="ba">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="can">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="cat">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="cdn">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="cdt">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="ouputsFlags">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="value" use="required" type="xs:NCName"/>
```

```
    </xs:complexType>
  </xs:element>
  <xs:element name="output">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="time"/>
        <xs:element ref="tension"/>
        <xs:element ref="position"/>
        <xs:element ref="velocity"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="time">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="dtOut" use="required" type="xs:decimal"/>
      <xs:attribute name="endTime" use="required" type="xs:integer"/>
      <xs:attribute name="startTime" use="required" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="tension">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="type" use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="position">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="type" use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="velocity">
    <xs:complexType>
      <xs:attribute name="comment" use="required"/>
      <xs:attribute name="type" use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```


Environmental I Physics I Laboratory
EPhysLab