Facultad de Informática. Departamento de Informática y Sistemas
Universidad de Las Palmas de Gran Canaria

Proyecto Final de Carrera

Entorno de Simulación para el Desarrollo de Misiones con Vehículos Submarinos Autónomos

Rodrigo Heredero Robayna

a013762@yahoo.es

Entorno de Simulación para el Desarrollo de Misiones con Vehículos Submarinos Autónomos

Tutor: Dr. D. Antonio Carlos Domínguez Brito

Cotutor: Dr. D. Jorge Cabrera Gámez

Cotutor: Dr. D. Antonio Falcón Martel

Las Palmas de Gran Canaria, 21 de Enero de 2010

A mis padres, hermanos y pareja, por su apoyo incondicional.

A mis tutores, por su infinita paciencia e inestimable orientación en todo momento.

A mis compañeros de carrera Enrique Fernández Perdomo y Eliezer Ramírez Cabrera por su colaboración, sin la cual no hubiera sido posible superar el presente proyecto.

A todos, mi más sincero agradecimiento...

Índice

CAPÍTULO 1.- *Introducción*

Desde la Oceanografía, ciencia para el estudio de procesos biológicos, físicos, geológicos y químicos que se dan en los mares y océanos, surge la necesidad de disponer de robots capaces de realizar el muestreo in situ, en lugares que resultan inaccesibles al ser humano.

En la actualidad, la robótica al servicio de la oceanografía podríamos clasificarla en tres tipos de sistemas, a los que podemos denominaremos de forma genérica "*Vehículos Oceanográficos"*:

- Boyas: No disponen de motores ni sistemas de navegación, por lo que se mantienen en la superficie del océano, bien a la deriva o bien anclada, con la finalidad de capturar y almacenar medidas tomadas con los sensores abordo y comunicarlas vía satélite, GPRS, radio enlace, etc.

Figura 1: Ejemplo de boya oceánica fondeada.

- ROV: de las siglas en inglés "Remote Operated Vehicle", es un robot conectado a una plataforma en superficie a través de un cable que permite el trasiego de datos, la comunicación de órdenes por parte del científico al vehículo y la alimentación energética. Así, los datos pueden ser de cualquier índole, dependiendo de los sensores de los que esté dotado el vehículo. Por tanto, tiene capacidad de navegación en el entorno del vehículo en superficie, si bien siempre es comandado, no presentando autonomía energética, ni a nivel de misión. El cable es a su vez una ventaja (alimentación y control del ROV) y un inconveniente (límite de

profundidad a alcanzar, límites de movilidad, etc.). Recolecta datos a través de sensores, y presenta también por lo general actuadores para interaccionar con el medio marino, como puede observarse en la Figura 2.

Figura 2: ROVs con brazo robótico y con cámaras submarinas.

- AUV: El término AUV viene de sus siglas en inglés "Autonomous Underwater Vehicle". También llamados "Unmanned Underwater Vehicles" (UUV), son robots capaces de navegar inmersos en cualquier medio subacuático y realizar diversas misiones, siendo controlados por un ordenador abordo, pudiéndose considerar autónomos en diversos aspectos:
    - No necesitan conexión física a agente externo, y por ende sin necesidad "a priori" de supervisión o envío de comandos por parte de agente externo al mismo, aunque el tipo de comandos que se puedan ejecutar dependerá del diseño de cada AUV.
    - Es auto-guiado, en función de su lazo de control.
    - Es auto-alimentado a través de baterías propias.

Como es objeto de este proyecto realizar un simulador para AUVs, seguiremos describiendo las características de un AUV. Las partes que lo integran típicamente son:

- Sensores: reciben información del medio, tanto para instrumentación como para navegación.
- Actuadores: normalmente motores y aletas destinados a la navegación.
- Sistema de alimentación autónomo: mediante el uso de baterías.
- Dispositivos de Comunicación: Permiten interaccionar con agentes externos, bien con otros AUVs, robots, o con operadores y oceanógrafos, tanto para labores de configuración del vehículo como para envío de datos.

- Sistema de procesamiento: Ejecuta la misión tomando como entrada los datos sensoriales y actuando sobre actuadores, alimentación y dispositivos de comunicación, cerrando así el bucle de control.

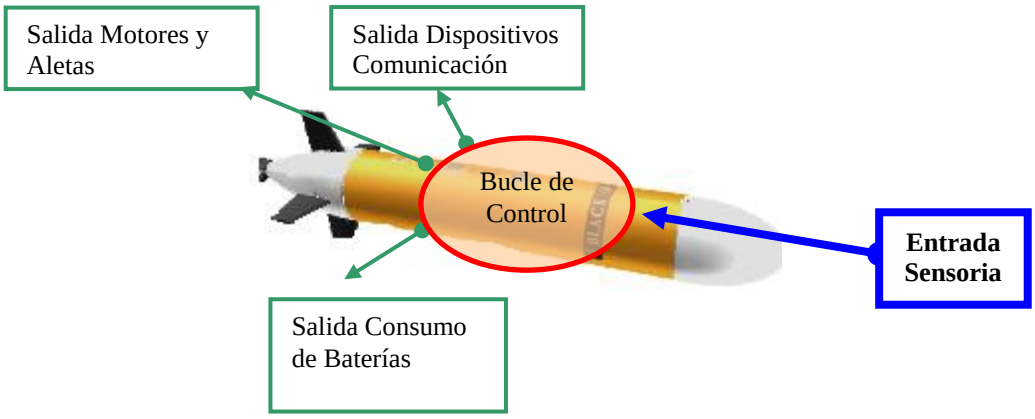


Figura 3: Subsistemas típicos de un AUV

Estas características hacen a los AUVs indicados para misiones en medios submarinos hostiles o donde el ser humano no puede llegar, ni buceando ni con dispositivos conectados con la superficie.

Además, son flexibles, ya que un AUV puede ser equipado a medida para la misión que se desea realizar, desde monitorización medioambiental, a estudio de recursos submarinos, creación de cartas batimétricas, misiones relevantes para la oceanografía en general, fines militares, o fines industriales.

Los primeros AUVs dependían únicamente del software embarcado, sin posibilidad de re-configuración, dado que la conexión sin cables era imposible. Una vez lanzado el AUV no había manera de cambiar la misión. Esto impulsó el desarrollo de sistemas para mejorar la autonomía del AUV, si bien tenía el problema de monitorización de la actividad del AUV: a lo sumo podían conocer la posición mediante ciertos sistemas de posicionamiento.

La aparición de sistemas de comunicación acústicos supuso un importante avance en este campo, con lo que la monitorización de la operación del AUV era mayor, pudiendo el operador incluso pedir datos durante el curso de la misión. Aparte, se introdujo la posibilidad de enviar comandos al AUV por parte de un agente externo, normalmente operador. El nivel de comandos que se pueden ejecutar en el AUV influirá directamente sobre el nivel de autonomía del AUV.

Para equipar un AUV se utilizan normalmente dispositivos muy sofisticados de elevado precio, lo cual hace imprescindible someter al AUV a un banco de pruebas antes de ser lanzado al mar, puesto que cualquier error puede provocar que el AUV sea irrecuperable y con ello se pierda gran cantidad de dinero y datos de misión.

Es en este punto donde se hace fundamental la simulación (objetivo de este proyecto), como medio de probar el AUV en un entorno virtual controlado, donde no corra riesgo. Todo simulador genera un entorno virtual que trata de imitar las condiciones y parámetros del medio submarino. La simulación se puede centrar en muy diversos aspectos:

- Nivel Bajo: permite probar la reacción de cada uno de los dispositivos, pudiendo éstos conectarse al simulador en modo "hardware-in-the-loop"[1]. Estas pruebas van dirigidas, por lo general, a comprobar el correcto funcionamiento de los mismos de forma individual.
- Simulación de dispositivos: permite usar dispositivos que el AUV realmente no tiene equipados, siendo emulados por el simulador. Así, se pueden comprobar las necesidades o no de incorporar cierto equipamiento al AUV, o hacer pruebas con el mismo cuando todavía no se dispone de todo el material para comprobar su validez para la misión.
- Simulación de AUVs: Permite disponer de un modelo informático de un AUV real, lo que posibilita el hacer pruebas del lazo de control del AUV, sin necesidad de disponer del mismo. Igualmente permite monitorizar y comprobar el correcto diseño y ejecución de las misiones individuales de los AUVs. Un científico puede probar las misiones y prever el fracaso o éxito de la misión antes de lanzar el AUV al mar.
- Simulación Multi-AUV: extendiendo el caso anterior, un simulador permite también probar misiones donde interaccionan varios AUVs, sin necesidad de disponer de ellos físicamente. Incluso se pueden combinar, y coexistir en el mismo instante AUVs reales conectados para simular dispositivos o para pruebas hardware-in-the-loop, con AUVs completamente simulados.

[1] Hardware-in-the-loop: Dispositivo real se conecta al simulador para simular ciertos aspectos, como pueden ser sensores de los que no disponga, o aspectos dinámicos, etc.

CAPÍTULO 2.- *AUVs: Aplicaciones y Simuladores*

El objetivo es situar al lector del contexto en el que se desarrolla este proyecto. Para ello comenzaremos describiendo los distintos ámbitos en los que se utilizan los AUVs o vehículos submarinos similares. Acto seguido, nos centraremos en el uso de simuladores culminando con la descripción de los trabajos relacionados con los AUVs que se llevan a cabo en la Universidad de Las Palmas de Gran Canaria.

2.1.- Uso de Vehículos Autónomos Submarinos

En la actualidad, los AUVs son utilizados en gran cantidad de tareas que se desarrollan en el medio acuático. Los siguientes son típicos ámbitos de utilización:

- Militares: fue una de sus primeras utilidades, como arma, elemento de espionaje, e incluso para la desactivación de minas submarinas, sin exponer vidas humanas.
- Reconocimiento de costas: equipados con diferentes tipos de sensores pueden servir para detectar y delimitar la extensión de ciertos fenómenos antes de que lleguen a la costa, como mareas de algas tóxicas o la extensión de cualquier otro tipo de contaminación medioambiental.
- Mapas batimétricos: equipados normalmente con sónar de barrido lateral, permiten recolectar datos para la generación de mapas precisos de los fondos marinos, tanto a nivel costero como oceánico, o lo que es lo mismo, mapas batimétricos (de gran utilidad en la navegación).
- Filmación de fondos marinos y estructuras sumergidas: de interés en ingeniería y en arqueología marina.
- Exploración de los océanos: su autonomía y reducido tamaño, en general hace que sean idóneos para misiones en entornos potencialmente peligrosos, donde las condiciones hacen impensable una navegación tripulada, como puede ser la exploración bajo el hielo en las zonas polares o a gran profundidad.
- Estudios oceanográficos y meteorológicos: Muestrear magnitudes submarinas, como conductividad, corrientes, oxígeno disuelto, salinidad o temperatura es relevante, por ejemplo, para ajustar modelos de interacción entre la atmósfera y el océano. Esta información permite desarrollar mejores predicciones sobre el tiempo, y ya se usan para la previsión o monitorización de la llegada de tormentas y

tsunamis a costas. Además, modelos resultan fundamentales para el estudio del cambio climático, ya que éste también se refleja en el mar.

- Observación de zonas naturales, incluso protegidas, como formaciones de coral o praderas de fanerógamas, interfiriendo lo menos posible con el medio.
- Investigación en la industria pesquera: la medición de ciertos parámetros como temperatura, salinidad, presencia de oxígeno disuelto, clorofila e incluso concentración de plancton permite predecir zonas de pesca adecuadas.
- También se están usando en la actualidad para la industria del petróleo o el despliegue y mantenimiento de cables submarinos de comunicación. En estos casos, un AUV puede servir para la identificación de averías o puntos de mantenimiento.

En función del uso, el equipamiento y diseño del AUV puede variar. Es el ejemplo del AUV AUSI, mostrado en la Figura 4, un innovador AUV que incorpora paneles solares para la recarga de sus baterías, dado que en sus misiones tiene la posibilidad de emerger con cierta frecuencia, y cuyo objetivo es aumentar la tiempo de autonomía del dispositivo en alta mar.

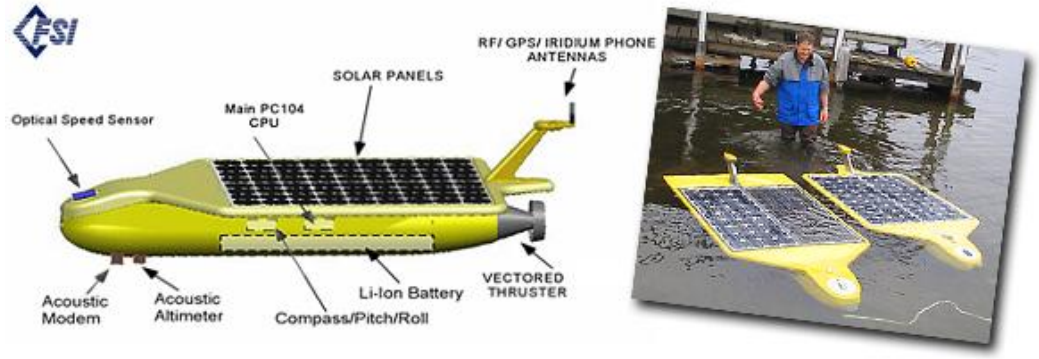


Figura 4: AUSI.- AUV con paneles solares.

Otros AUVs presentan grandes aletas laterales que permiten "planear" para poder sumergirse, evitando el uso de motores. Son los denominados "gliders". En estos AUVs prima la eficiencia energética, y actualmente son los vehículos de mayor autonomía, como lo demuestra el reciente hito del glider "Scarlet Knight" que consiguió atravesar el Atlántico Norte tras una singladura de casi ocho meses (referencia **[ICOOL]**).

La forma típica de un AUV es la de un torpedo, permitiendo que la resistencia sea mínima durante la navegación, como se muestra en la Figura 5. Como vemos, las posibilidades de diseño son infinitas, según la utilidad que se le dé.

Figura 5: Forma de torpedo de los AUVs.

2.2.- Simuladores Actuales y Utilidad

En cuanto a simuladores, existe una amplia variedad de simuladores. Los más relevantes serán descritos durante el análisis del presente proyecto. Todos tienen la misión de someter a pruebas dispositivos, sistemas, algoritmos de control o misión de los AUVs. Dependiendo de la fase del proyecto en el que se enmarca el simulador, nos encontramos con simuladores de mayor o menor grado de resolución.

Así, en fases tempranas, se opta por simuladores muy ligados a la arquitectura del AUV, como es el caso del simulador OEX (referencia **[SON03])**, que permite depurar los distintos sistemas que lo conforman de forma realista. Este tipo de simuladores es muy dado a hardware-in-the-loop.

Sin embargo, en fases más avanzadas de los proyectos, nos encontramos con simuladores más destinados a comprobar el correcto desarrollo de la misión de uno o varios AUVs. En este caso, si bien se puede conectar el AUV real al sistema, suele trabajarse con AUVs completamente simulados, que desarrollan el algoritmo de control embebido en los AUVs reales, pero en un entorno virtual creado por el simulador, incluso simulando sus dispositivos.

En estos simuladores el realismo de la simulación, tanto a nivel de representación del entorno (realismo gráfico) como de simulación del movimiento (realismo físico), se vuelve mucho más importante. En simuladores basados en la monitorización de la misión de un único AUV, como es el proyecto NEPTUNE (ver referencia **[RID042]**) desarrollado en la universidad de Girona, el simulador gráfico es muy realista y fidedigno, usando modelos matemáticos reales y precisos para la simulación de sensores y actuadores hasta el punto de permitir hardware-in-the-loop . De hecho la vista es centrada en el vehículo.

Surge este proyecto del desarrollo del AUV llamado URIS y, si bien inicialmente el simulador tenía como misión la depuración de dispositivos con hardware-in-the-loop, éste ha evolucionado hasta incluir una animación detallada del movimiento uno o varios AUVs simulados. Este simulador permite también monitorizar el desarrollo de la misión del AUV. Puede observarse el nivel de realismo gráfico alcanzado por NEPTUNE en la Figura 6 y Figura 7.

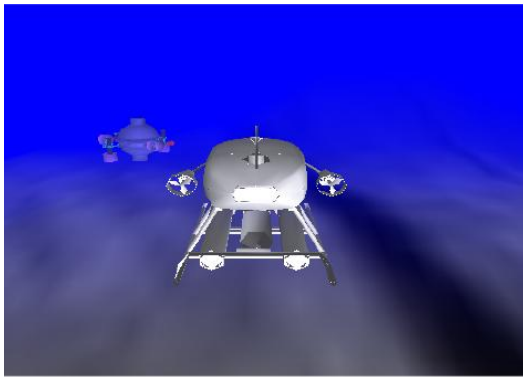


Figura 6: Simulado del entorno y del propio AUV URIS.

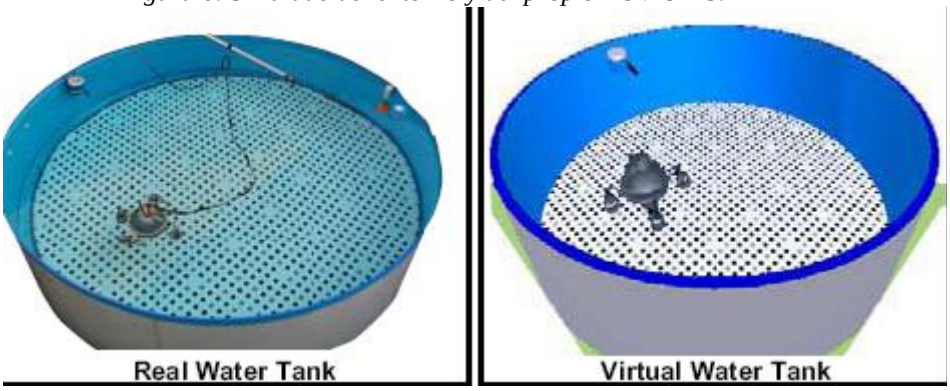


Figura 7: Comparación de entorno real hardware-in-the-loop con entorno virtualizado de URIS.

En proyectos mucho más avanzados, suele sacrificarse la precisión y el uso de modelos físicamente precisos, y con ello la posibilidad de hardware-in-the-loop, en pos de facilitar la simulación de un grupo de AUVs que podrían realizar varias misiones individuales o una misión en equipo. En estos simuladores, la representación gráfica pierde realismo, normalmente representando al AUV como un punto en el espacio cartesiano, potenciándose otros aspectos como cambios de vista y de cámaras que permitan al científico centrarse en los AUVs que desee.

Es el caso del proyecto CODA (referencia **[ALB03]**): entorno de simulación para monitorización de misiones multi-AUV, con técnicas avanzadas de control para misiones conjuntas, pero un nivel de realismo bastante bajo. Sin embargo, el proyecto CADCON (referenciaS **[ALB03]** y **[CAD]**) permite simulaciones mucho más realistas, perdiendo en

capacidad de control de misiones conjuntas. En la actualidad, estos dos simuladores trabajan conjuntamente, como se muestra en la Figura 8, consiguiendo un robusto sistema de simulación con modelos realistas, combinados con un perfecto control y monitorización de misiones multi-AUV.

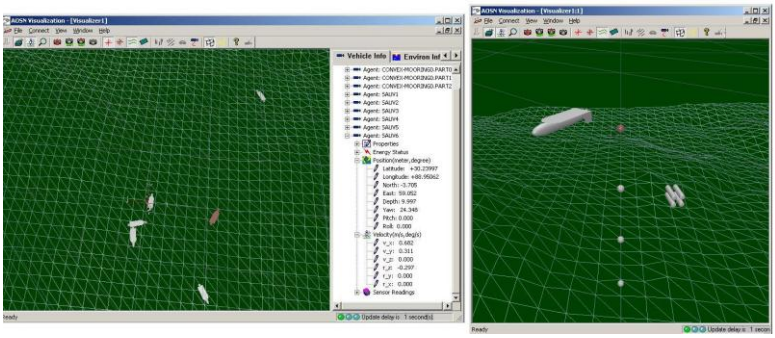


Figura 8: Simulador CODA/CADCON.

2.3.- Marco de Trabajo

Entre los proyectos finales de carrera de la Facultad de Informática de la Universidad de Las Palmas de Gran Canaria, este proyecto se enmarca en un grupo de proyectos destinados al estudio e implementación de software para control, monitorización y simulación de AUVs. En concreto, en la actualidad existen dos proyectos, que son:

- Sistema Integrado de Control para un Vehículo Submarino Autónomo: destinado a implementar el lazo de control embebido en el AUV, gestionando todos los dispositivos (sensores, actuadores, dispositivos de comunicación, etc.) y ejecutando los planes de misión.

  La conexión con el presente proyecto es a nivel de simular dispositivos del AUV del que no se dispongan en el laboratorio a través del simulador, además de simular el entorno submarino antes de ser construido y lanzado al mar.

- Sistema Integrado de Planificación y Control de Misiones con Vehículos Submarinos Autónomos: destinado a monitorizar la actividad de un AUV, planificar sus misiones, y enviar comandos de control que permita hacer modificaciones durante la misión en curso.

  La conexión será con los AUV simulados que se generen en este proyecto para probar el simulador, ya que serán monitorizados y comandados por el mismo planificador.

CAPÍTULO 3.- *Planificación y Objetivos*

Este capítulo comienza definiendo los objetivos que serán la base de todo el proyecto, siguiendo con la metodología de trabajo utilizada, los recursos empleados en el proyecto, la planificación del mismo y, finalmente, las horas de dedicación al mismo.

3.1.- Objetivos

El proyecto se enmarca en un conjunto de proyectos de final de carrera de la Universidad de Las Palmas de Gran Canaria (ULPGC), más específicamente del Instituto de Ingeniería Computacional SIANI, destinados a la aplicación de la Ingeniería Informática a estudios oceanográficos a través de Vehículos Autónomos Submarinos (AUV, de las siglas en inglés).

Objetivo Principal: *"Análisis, diseño e implementación de un entorno de simulación que sirva para dar soporte y analizar el desarrollo de una misión por parte de uno o varios AUVs.*"

Esto conlleva, como objetivo implícito previo, el realizar un **Análisis en profundidad** del contexto en el que se ubica este proyecto**.** Puesto que dentro de la ULPGC no se encuentran otros proyectos o estudios anteriores, el objetivo previo es hacer un análisis exhaustivo de los aspectos más comunes relacionados con los AUVs. Se comenzará con generalidades (estado del arte, estructura básica de un AUV, misiones típicas).

El análisis debe profundizar en aquellos aspectos más relacionados con la simulación de estos vehículos: por ejemplo otros simuladores, formatos para representación de batimetría y manejo de volúmenes de datos, sensores más importantes de un AUV y modelos para su simulación, parámetros fisicoquímicos más relevantes y modelos, sincronización de clientes de un simulador, etc. Este análisis debe proporcionar la definición de los siguientes diseños:

- Diseño de un formato de representación de misiones individuales de AUVs.
- Diseño de un formato de especificación de la instrumentación y dispositivos abordo del AUV.

- Diseño de protocolos de acceso a ficheros de mapas batimétricos y otros volúmenes de datos.

Tras éste, el objetivo fundamental del proyecto se puede desglosar en el análisis, diseño e implementación de dos aspectos.

Entorno de Simulación

Aplicación que permite la incorporación de clientes AUV para simular el desarrollo de diversos tipos de misiones individuales en un entorno completamente virtual. El objetivo es una **aplicación gráfica,** de uso sencillo para los usuarios (siendo la mayoría oceanógrafos e ingenieros), que oculte la complejidad del sistema.

La aplicación debe estar enfocada, por una parte, a la representación y gestión de la evolución temporal del **medio submarino** (corrientes, distribuciones de parámetros fisicoquímicos como temperatura o salinidad, carga de mapas batimétricos etc.) donde se desarrollarán las misiones. Con esto se habilita el poder testear misiones reales sin el coste y riesgo que conlleva realizar ensayos previos en el medio real.

No se buscan modelos complejos ni fidedignos en esa fase, que se podrán incorporar en futuros proyectos fundamentos en los resultados de este proyecto. Es mucho más interesante en el marco del presente proyecto poder disponer de un prototipo integral del sistema que resulte operativo, al menos, en los aspectos fundamentales.

Por otra parte, para la simulación propiamente dicha del desarrollo de la misión de un AUV, el entorno debe poder simular aspectos del mismo como son los **sensores principales** y **la hidrodinámica** del mismo.

- En el caso de los sensores, el objetivo es analizar e implementar modelos sencillos de los sensores más relevantes del AUV (altímetro, sensor de profundidad, sónar, brújula, inclinómetros,...) y de comunicación (modem acústico) de forma sencilla que permita experimentar con ellos sin necesidad de disponer de ellos en el laboratorio.
- Para la hidrodinámica, no se busca un nivel alto de detalle en la misma, que sería objetivo de un proyecto futuro fundamentado en los resultados del presente.

El elemento común a los objetivos anteriores es que un cliente AUV pueda simular misiones individuales, si bien en la medida en que sea posible el entorno debe permitir en el futuro la incorporación de misiones colaborativas que impliquen compartición de datos

entre AUVs, formaciones, etc. Este sería objetivo de un proyecto fundamentado en los resultados del presente. El resultado final debe ofrecer las siguientes características:

- **Integración:** un objetivo intrínseco es que el resultado de otros proyectos final de carrera, como puede ser el diseño del sistema de control de un AUV, se puedan verificar en el presente simulador. Por ello, tanto los tres diseños resultados del objetivo de **Análisis Previos** como un diseño del formato de intercambio de paquetes se han de realizar en estrecha colaboración con dichos proyectos.

  Revisar el **ANEXO I de Proyectos Complementarios**, para ver los dos proyectos finales de carrera del entorno de la ULPGC con los que se ha colaborado para estos diseños comunes.
- **Configurabilidad y Mejora**: debe ser lo suficientemente versátil y configurable para adaptarse a cualquier situación o entorno submarino a la vez que a nuevos dispositivos, mediante la fácil incorporación de nuevos modelos que sustituyan los modelos más modestos y menos fidedignos que son objetivos del presente proyecto en el futuro.
- **Escalabilidad**: el entorno de simulación debe permitir la incorporación de clientes de muy diversa índole, desde AUVs completamente simulados, a AUVs hardware-in-the-loop o incluso la fácil incorporación de otro tipo de vehículos submarinos, como ROVs o boyas.

Clientes que simulen comportamiento de AUV

Aplicación cliente que pueda conectarse de forma sencilla al **entorno de simulación** y usar los recursos del mismo para simular una misión completa.

El objetivo no es el desarrollo de misiones muy complejas ni cooperativas, sino misiones simples e individuales que muestren las capacidades del **Entorno de Simulación** para servir al AUV datos sensoriales e información del entorno, así como para simular su navegación mediante un modelo hidrodinámico sencillo. El resultado debe poder validarse contra el **Entorno de Simulación**, comunicándose con el mismo para poder simular toda su misión.

También, si el desarrollo del proyecto así lo permitiese o para proyectos futuros, se plantea un objetivo secundario, que es estudiar arquitecturas de misiones colaborativas, donde los AUVs puedan comunicarse entre sí y adoptar formaciones para completar misiones complejas.

3.2.- Metodología

Como metodología para alcanzar los objetivos planteados, se ha empleado un modelo de **ciclo de vida del software** propio de la **Ingeniería del Software**.

En ciclo de vida de un desarrollo o proyecto software consiste en el conjunto de fases consecutivas que completan el desarrollo de una aplicación. Estas fases son, en el ciclo de vida en cascada original, **Análisis**, **Diseño**, **Implementación, Pruebas** y **Mantenimiento**, como se muestran en la Figura 9.

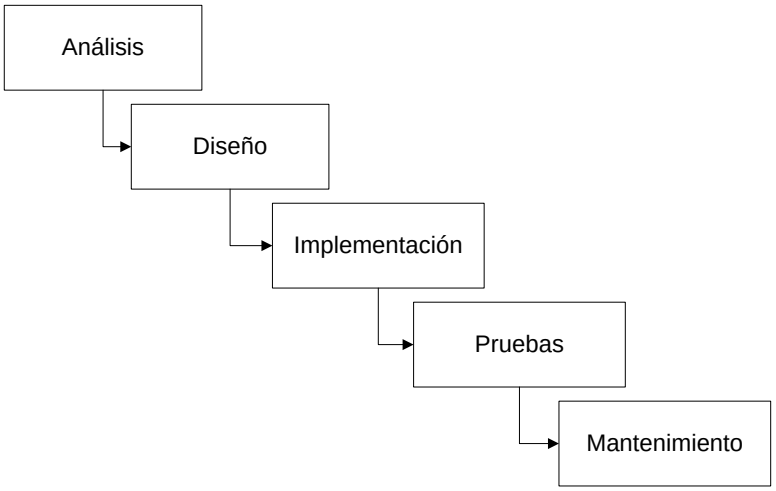


Figura 9: Ciclo de Vida del Software en cascada.

En función del tipo de software y contexto en el que se desarrolla, este ciclo de vida básico adopta diversas modificaciones o "paradigmas". En concreto el paradigma utilizado principalmente es **Ciclo de Vida Iterativo**. Consiste en la iteración de varios ciclos de vida en cascada consecutivos, de forma que en cada iteración se obtiene una nueva versión con nuevas funcionalidades incorporadas al sistema. Puede observarse en la Figura 10.

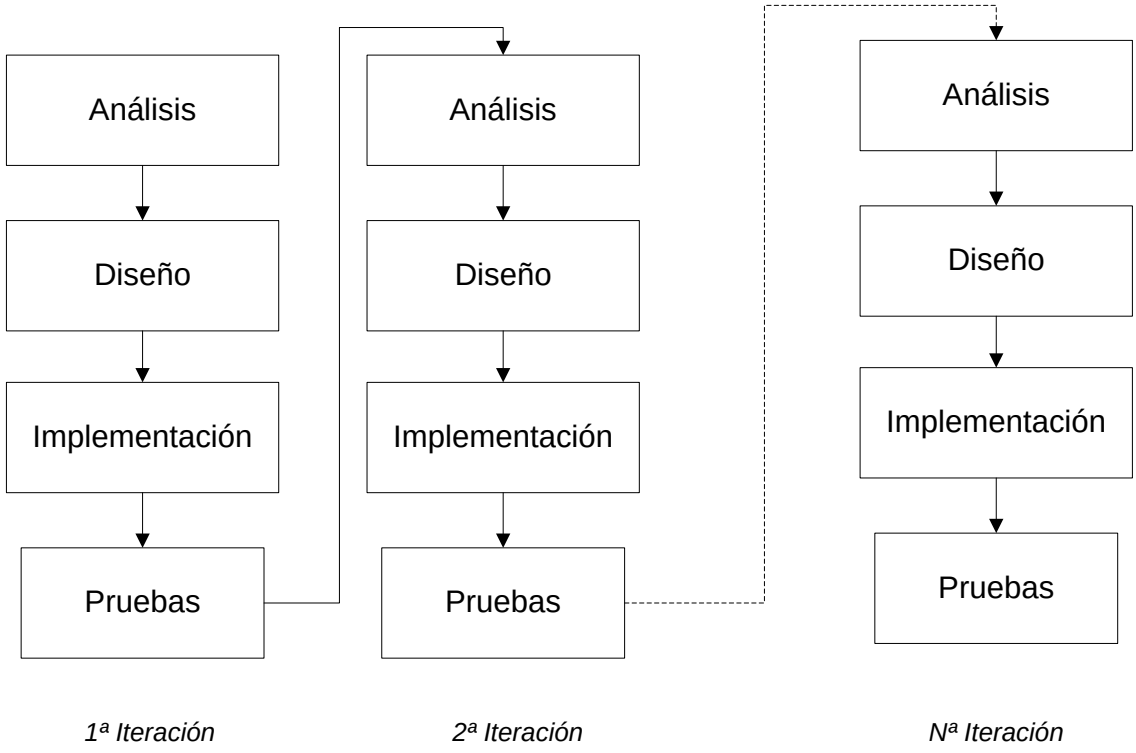


Figura 10: Ciclo de vida Iterativo.

La elección este ciclo de vida es adecuada para el presente proyecto, ya que permite hacer el desarrollo del proyecto por pasos, de manera que en cada paso quedan probadas las nuevas funcionalidades incorporadas al sistema.

Dadas las dimensiones del proyecto, se hace adecuado ir cumpliendo objetivos paso a paso, comenzando por los más básicos sobre los que se soportarán los más generales (*metodología ascendente*). El hecho de que en cada iteración se prueben los resultados, hace que se garanticen resultados en las iteraciones finales de mayor grado de generalidad. Por otra parte, es adecuado para proyectos de un solo desarrollador, al no ser las iteraciones solapadas sino consecutivas.

Cada iteración va a constar de dos **Subproyectos** que se desarrollarán en paralelo: el **Entorno de Simulación** y los **Clientes AUV.**

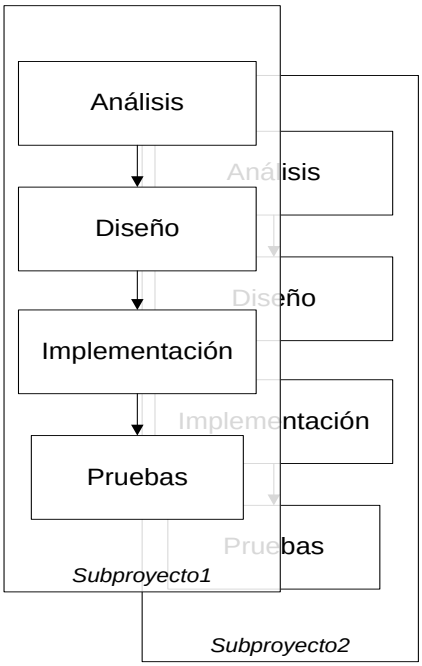


Figura 11: Estructura en Subproyectos de cada iteración del ciclo de vida.

Como se muestra en la Figura 11, a medida que se incorporan funcionalidades al Entorno de Simulación y a su entorno gráfico, se incorporan funcionalidades paralelas en los Clientes AUV.

Descripción de iteraciones.

En la primera iteración, la fase de análisis ha consumido la mayor parte del tiempo dedicado a esta iteración, cumpliendo con los objetivos de **Análisis en Profundidad** previamente establecidos. En la fase de diseño e implementación se han diseñado los formatos de especificación de misión, AUV, batimetría y volúmenes de datos. También se ha tratado el formato de intercambio de paquetes, a la vez que se ha diseñado, implementado y probado con pequeños programas. En primera iteración analítica se ha colaborado con otros proyectos finales de carrera, tal como se ha descrito en los objetivos, para sentar las bases comunes de estos proyectos.

En el resto de iteraciones, se ha ido construyendo el **Entorno de Simulación**, concurrentemente con su **interfaz gráfica** y el **cliente AUV**, comenzando en las primeras iteraciones con objetivos más básicos para el funcionamiento de las aplicaciones, y terminando con iteraciones donde se incorporan funciones generales del sistema. En cada iteración, el análisis dio como resultado un diseño e implementación en los distintos subproyectos.

Para la última fase de pruebas de cada iteración, además de pruebas básicas de funcionamiento individual, también se han ido planteando pruebas de integración y trabajo conjunto de los subproyectos, de forma que en cada iteración y a medida que se ha avanzado en el proyecto han ido ganando más peso este tipo de pruebas conjuntas frente a pruebas individuales.

Fases de Ciclo de Vida en Cascada de cada iteración.

Análisis

Gran parte del análisis se ha basado en la documentación citada en la bibliografía del presente proyecto. Esta fase ha sido mucho más relevante en la primera iteración, y se desarrolló en estrecha colaboración con otros proyectos final de carrera de la ULPGC relacionados con AUVs.

En el resto de iteraciones, el análisis se ha focalizado más en los requisitos de usuario de las distintas aplicaciones del presente proyecto. En éstas, se han utilizado **Diagramas UML de Caso de Uso** para representar los requisitos de usuario, de forma que incrementalmente se han ido incorporando casos de usos a medida que ha avanzado el proyecto.

Diseño

El diseño ha sido una fase mucho más relevante en iteraciones intermedias del proyecto. Para el diseño se han utilizado **Diagramas UML de clases**, **Diagramas de Flujo de Datos**. También se han utilizado patrones de diseño durante el diseño de la aplicación, como puede consultarse en los anexos.

Implementación

En la primera iteración más analítica, la implementación se ha basado en la generación de ficheros **XML** y **XDS** para representar los formatos de especificación de la misión y del AUV. Para los formatos de mapas batimétricos y de volumen de datos de ha utilizado **netCDF**.

En el resto de iteraciones del proyecto se ha utilizado **Java** para el desarrollo del proyecto, si bien ciertos trabajos complementarios han utilizado **Matlab** (ver en documento anexos, el **Estudio del Sónar de Barrido Lateral**). Dentro de **Java**, también se ha utilizado el formato estándar **XDR** para comunicación e intercambio de paquetes.

También se ha utilizado **JavaScript** para la incorporación de *Google Maps* en el entorno gráfico.

Pruebas

En las primeras iteraciones, las pruebas se han centrado en versiones individuales del **Entorno de Simulación**, su representación **Gráfica** y los **Clientes AUV**. A medida que se ha avanzado en el proyecto, además de estas pruebas, se han incorporado cada vez más pruebas de integración y funcionamiento conjunto de los subproyectos desarrollados en paralelo.

Mantenimiento

La fase de mantenimiento está implícita: en cada iteración, la siguiente iteración es un mantenimiento del anterior, que da como resultado una nueva versión, no sólo con nuevas funcionalidades sino con actualizaciones de las funciones anteriores y enmienda de bugs del sistema. Es por ello que no se ha mostrado dentro del ciclo de vida en cascada de cada iteración.

3.3.- Recursos Necesarios

Hardware

- *Portátil Principal*: **ACER Aspire 5920**. Intel(R) Core(TM) 2 Duo CPU T7300 @ 2.00GHz 200GHz 2038 MB RAM .**Microsoft Windows Vista (TM) Home Premium (32 bits)**, con pantalla secundaria **ACER X243H**.
- *Portátil Secundario*: **ASUS X50Rseries**. Intel(R) Core (TM) Duo CPU T2250 @ 1.73GHz, 795MHz, 1,87GB de RAM. **Microsoft Windows XP Profesional Version 2002 (SP2).**

Aplicaciones en General.

- *Acrobat Reader*: Se han usado varias versiones, la última **Acrobat Reader 9.0** versión **9.1.1**. Lectura de ficheros PDFs.
- *PDFCreator*: herramienta para convertir documentos a formato PDF.
- *Internet Explorer:* Se han usado varias versiones, la última **Internet Explorer 7**. Visualización de páginas web

- *Mozilla Firefox:* Se han usado varias versiones, la última **Mozilla Firefox 3.0.11**. Visualización de páginas web.

Aplicaciones para Análisis y Diseño

- *Microsoft Office Visio*. Se han utilizado varias versiones, la última **2007**. Diagramas de UML y de flujo de datos y procesos.
- *Magic UML 10*: Diagramas UML de Casos de Uso.

Aplicaciones para Desarrollo

- *Java JDK 1.5 (principal) y 1.6 (ciertos módulos)*: máquina virtual de Java para compilar y ejecutar código creado en Java.
- *Netbeans IDE 6.1*: framework para programar y compilar todas las aplicaciones en Java. También usado para compilar los ficheros XML usando los ficheros XSD.
- *Librerías Accesorias en NetBeans para el desarrollo Java*:
    - **JDOM**: Librería para lectura y escritura de ficheros XML.
    - **JDIC 0.9.3-bin-windows**: librería para incrustación de explorador web y otras aplicaciones en paneles de Java.
    - **FreeHEP XDR Library 2.0.3 API**: librería para serialización / deserialización de paquetes de datos en XDR.
    - **nc2.2**: Librería para lectura / escritura de ficheros en NetCDF.
- *NotePad++*: procesador de textos con marcado de distintos lenguajes de programación. Usado para HTML, y JavaScript (en la incorporación de *Google Maps* al proyecto) y para el desarrollo de ficheros XML.
- *API de Google Maps:* se ha utilizado esta API para el manejo e incorporación de representaciones en un mapa Google incrustado en la aplicación, desde código Javascript.

Formatos de Datos

- *NetCDF*: Formato para contener volúmenes de datos (o mapas) generalmente batimétricos, pero también de otras magnitudes.
- *XML y XSD*: Formato para especificación de características (de AUV, misiones) y de configuración de aplicaciones.
- *HTML*: para incorporación de *Google Maps* al proyecto.

- *XDR*: para el diseño de los paquetes de conexión de interconexión con otros proyectos y con subsistemas remotos del presente proyecto.

Aplicaciones para Documentación.

- *Microsoft Office Profesional*: Se han utilizado varias versiones, la última **Microsoft Office Profesional 2007**. En especial las aplicaciones MS Word, MS Excel, MS PowerPoint, como procesador de textos, de presentaciones y de cálculos y gráficos analíticos.
- *MathType*: módulo incorporado a *Microsoft Office Word* para la generación de ecuaciones.
- *Adobe Photoshop cs2 9.0:* para la composición de imágenes y gráficos.

Otros Recursos

- SIANI, Instituto Universitario de *Sistemas Inteligentes y Aplicaciones Numéricas de Ingeniería:* se han usado las dependencias del laboratorio de este instituto en el **Edificio Polivalente del Campus de Tafira de la ULPGC** para reuniones con otros alumnos con proyectos final de carrera del marco al que pertenece el presente proyecto, centro de trabajo y para reuniones con tutores del proyecto.
- *Matlab 7.0*: software matemático con entorno de desarrollo integrado, utilizado para el desarrollo del estudio del Sónar de Barrido Lateral.
- *MOODLE*: se ha utilizado en fases tempranas del proyecto, instalado en servidores de dependencias del SIANI, para la gestión documental y gestión de relación con tutores y alumnos con proyectos de final de carrera en el marco en el que se sitúa el presente.

3.4.- Plan de trabajo y Temporización

3.4.1.- Planificación del trabajo

Para la planificación del proyecto se ha utilizado la metodología expuesta en el **Apartado 3.2**.

Primera iteración: análisis en profundidad

En el siguiente Figura 12 se muestra la dedicación estimada a cada fase de la iteración.

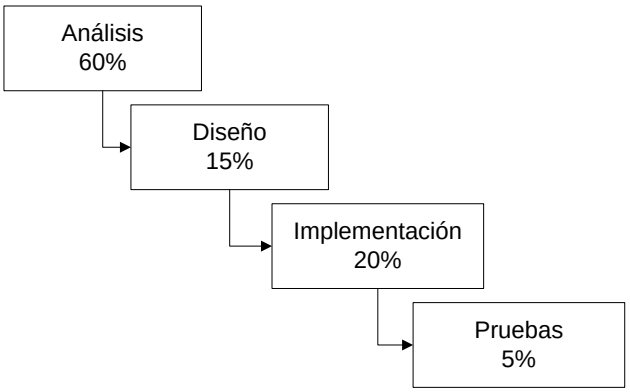


Figura 12: Porcentaje de la primera iteración dedicada a cada fase.

- Análisis.- estudio general para enfocar el proyecto. Se han abordado los siguientes temas:
    - Estado del Arte (**Capítulo 2**).
    - Entornos de programación y de formatos de intercambio de paquetes (**Capítulo 4**).
    - Arquitectura de otros simuladores de AUVs y modelos de sincronización (**Capítulo 5**).
    - Estudio de batimetría y formatos para representación de volúmenes de datos (**Capítulo 6**).
    - Estudio de modelos de parámetros fisicoquímicos y magnitudes relevantes del entorno submarino (**Apartado 7.5**).
    - Estudio de modelos de sensores e hidrodinámica de interés para los AUVs (**Apartado 7.2, 7.3, y 7.4**).
    - Estudio de misiones individuales y colaborativas entre AUVs (**Apartado 11.1**)
- Diseño.- en esta fase se han abordado las siguientes tareas:
    - Diseño del paquetes de datos y protocolo de intercambio de paquetes del simulador con clientes externos. (**Capitulo 8 y 9**).
    - Diseño de ficheros de datos batimétricos (**Capítulo 10.2.6**).
    - Diseño de especificación de misiones (**Capítulo 11**).
    - Diseño de especificación del AUV (**Capítulo 11**).
- Implementación.- Como resultado de análisis y diseño, se han hecho las siguientes implementaciones:
    - Ficheros XML para representación de misiones (**Capítulo 11**).
    - Ficheros XML para representación de AUV (**Capítulo 11**).

- Implementación de librería para comunicación e intercambio de paquetes (**Capítulo 8**)

- Pruebas.- Para probar la bondad de la fase de implementación, el conjunto de pruebas son:
  - Test de librería para intercambio de paquetes (**Capítulo 13**).
  - Pruebas de validación y completitud de ficheros para representación de misiones y AUV (**Capítulo 13**).

Resto de iteraciones: desarrollo incremental del simulador

Aquí se engloban el conjunto de iteraciones dedicadas al desarrollo del simulador en sí. Se nombran las actividades de forma general, se bien se desarrollan en sucesivas iteraciones. En la Figura 13 se puede observar la dedicación a cada fase.

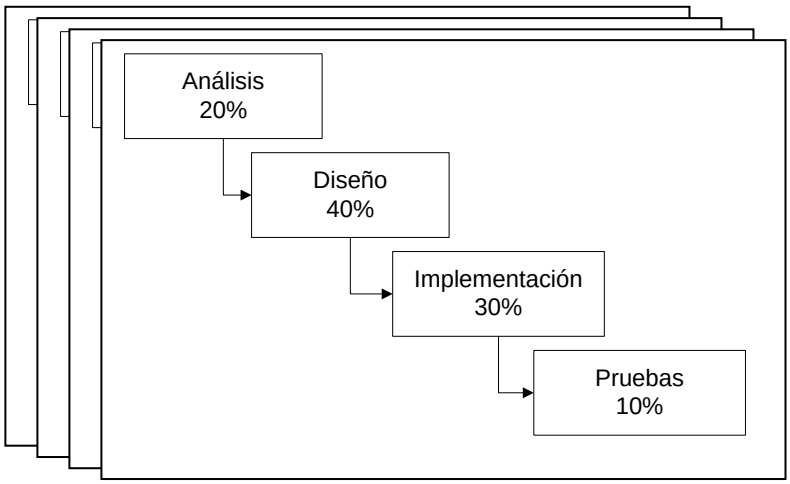


Figura 13: Porcentaje de tiempo dedicado a cada fase en el resto de iteraciones.

Análisis.- En cada iteración, se han realizado las siguientes tareas analíticas:

- Requisitos de usuario, tanto del motor de la simulación (o entorno de simulación), como de la representación gráfica como del cliente AUV simulado.
- Catálogo de servicios y comandos que debe ofrecer el motor de simulación.

En función de la focalización de estos dos aspectos de análisis, se ha diseñado, implementado y probado una nueva versión de los distintos subproyectos, el

entorno de simulación o servidor del entorno con su interfaz gráfica (**Capítulos 10** y **12**) y el cliente AUV (**Capítulo11**).

Diseño.- como resultado del análisis, se han abordado las siguientes tareas de diseño:

- Diseño de la arquitectura general del Simulador (**Capítulo 8 y 9**)
- Diseño de protocolo de comunicación entre Subsistemas del Simulador. (**Capítulo 8 y 9**)
- Diseño del Entorno de simulación o Servidor del Entorno (**Capítulo 10**).
    - Componentes
    - Protocolos de comunicación.
- Prototipo de la interfaz gráfica de usuario (**Capítulo 12**).
- Diseño lógico de la interfaz gráfica. (**Capítulo 12**).
- Prototipo de cliente AUV (**Capítulo 11**).
- Diseño lógico del AUV (**Capítulo 11**).

Implementación.- como resultado del análisis y diseño de cada iteración, se ha realizado:

- Implementación del Servidor del Entorno: incorporación de servicios y comandos (**Capítulo 10**).
- Implementación de la interfaz gráfica: visualización y configuración de nuevos servicios y comandos (**Capítulo 12**).
- Implementación del AUV simulado: uso de servicios y comandos (**Capítulo 11**).

Pruebas.- para probar la nueva versión de la iteración, se han realizado:

- Pruebas individuales de funcionamiento (**Capítulo 13**). Se dan en distintos niveles:
    - A nivel del desarrollo del Servidor del Entorno.
    - A nivel del desarrollo de la interfaz gráfica.
    - A nivel del desarrollo del AUV simulado.
- Pruebas de integración entre los subproyectos (**Capítulo 13**).

Planificación global del proyecto:

Al ciclo de vida iterativo anterior, hay que añadir dos últimas fases de Documentación y Conclusiones. En cuanto a la Documentación, incluye tareas de:

- Realización de la presente memoria.
- Realización de manuales.
- Documentación de la implementación hecha.
- Documentación de documentos comunes con otros proyectos final de carrera:
    - Especificación XML de Misiones.

- Especificación XML de AUV.
- Especificación de protocolo de comunicaciones y formato de intercambio de paquetes.

De manera global, haciendo media, podemos concluir la estimación temporal de la planificación del proyecto con la Figura 14

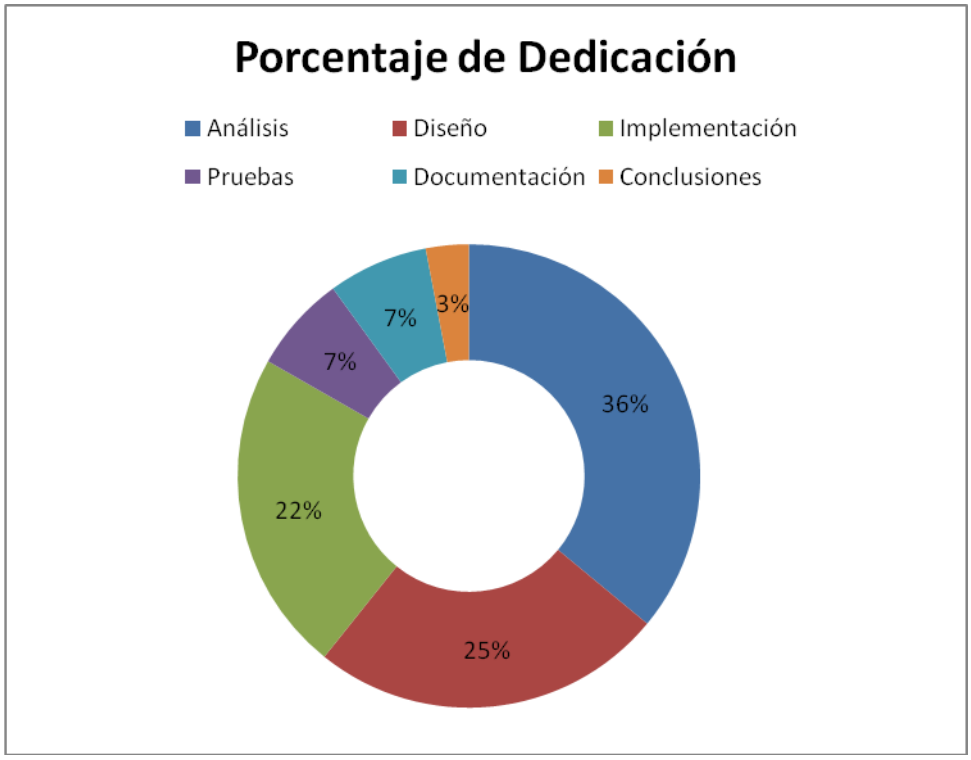


Figura 14: Porcentaje de dedicación a las distintas fases en todo el proyecto.

3.4.2.- Temporización

La realización del presente proyecto ha llevado 3 años (2006-2009). Si bien es un periodo de tiempo extenso, es justificable atendiendo a que durante la realización del mismo, se han simultaneado otras actividades.

La principal, relacionada con el contexto del proyecto, consiste en la realización de una beca con el objetivo del estudio del cambio climático a través de la información del mar recogida desde de boyas oceanográficas. El proyecto, que fue apodado como el **Proyecto Midas**, fue propuesto por la Escuela Superior de Ingenieros de Telecomunicaciones de la ULPGC y se desarrolló durante un año.

El proyecto cumplió con dos sub-objetivos claves: la programación del lazo de control de la boya (con la programación del CTD incluido a bordo) y la monitorización de la actividad a través de comunicación por satélite.

En él también han participado **Enrique Fernández Perdomo** y **Eliezer Ramírez Cabrera**, ambos citados en el **ANEXO I** como colaboraciones en el marco del proyecto.

Figura 15: Izquierda, Enrique y Eliezer con la Boya programada. Derecha, interfaz de monitorización desarrollada.

El resto de actividades simultaneadas con la realización del presente proyecto no están dentro del contexto del mismo y han sido básicamente tareas laborales, centradas en el ámbito de la Informática Médica, que han exigido una gran dedicación.

La dedicación total al proyecto se puede estimar en **800 horas** de trabajo real y efectivo dedicados plenamente al mismo. Sin embargo, a lo largo de este largo periodo y dado el carácter exploratorio en el que se enmarca el proyecto dentro de la ULPGC se ha complementado la realización del proyecto con la dedicación en tiempo al estudio y la profundización en ciertos temas colaterales que sin duda han contribuida al resultado final. Parte de estos estudios se han querido también reflejar en el presente documento, integrándolos en los anexos que se incluyen en el CD que se incluye en la documentación de este proyecto.

PARTE I

Análisis previos

CAPÍTULO 4.- *Entornos de programación*

Este capítulo de análisis hace un estudio comparativo de dos entornos de programación, Matlab y Java, para decidir cuál es el que más se adapta a las necesidades y objetivos del proyecto.

Seguidamente, se estudian los formatos para intercambio de paquetes más relevantes y que pueden ser de mayor interés durante las fases de diseño e implementación del proyecto.

4.1- Entorno de Programación: Matlab / Java

Para implementar este simulador, se plantean dos entornos de programación Matlab y Java. Si bien inicialmente en la propuesta del proyecto se planteó solamente Matlab, distintos prototipos que se desarrollaron resaltaron deficiencias en este entorno, lo que hizo necesario el siguiente estudio que señalará las ventajas y deficiencias de usar cada uno, para finalmente poder tomar una decisión sobre el entorno de programación adecuado.

4.1.1.- Generalidades

Java es multiplataforma a través de la máquina virtual de Java orientado a objetos. Esta independencia de la máquina en la que se va a ejecutar, permite aplicaciones fácilmente transportables e independiente. Es un lenguaje normalmente interpretado, aunque también puede ser compilado para su ejecución. Además, Java es muy similar a C++, pero con un modelo de objetos simplificado y eliminando herramientas de bajo nivel que inducen a errores, como son punteros a memoria.

En cambio Matlab es un software matemático con un entorno de desarrollo integrado y lenguaje de programación propio, disponible para Windows, Unix y Mac Os. Posee gran cantidad de toolboxes que resuelven problemas de bajo nivel, permitiendo al usuario trabajar a algo nivel. Presenta dos módulos importantes adicionales que aumentan su potencial. Uno es el **GUIDE**, un editor de interfaces gráficas; el otro **Simulink,** una plataforma de simulación multidominio.

En cuanto a licencias, Matlab requiere licencia, mientras que Java tiene una licencia GNU GPL, si bien en función de las librerías que se usen puede requerir librerías de pago.

4.1.2.- Orientación a Objetos.

Java es un entorno de programación orientado a objetos en su origen, mientras que Matlab no lo es, al menos no originariamente. Si bien las últimas versiones sí que van dando soporte a la orientación a objetos, es todavía laborioso, frente a la gran cantidad de entornos de desarrollo que facilitan la programación en Java.

La programación orientada a objetos tiene muchas ventajas, como pueden ser:

- **Modularidad** y fácil **Escalabilidad**.
- **Mantenimiento más sencillo**.
- **Reutilización y extensión del código, agilizando el desarrollo del Software.**
- **Facilita la creación de programas visuales**, de interés para el presente proyecto.
- **Facilita el trabajo en equipo**, ideal para futuras ampliaciones de los resultados del presente proyecto.

4.1.3.- Herramientas de representación gráfica de matrices de datos

Para el proyecto que se desarrolla, es importante disponer de herramientas que faciliten la representación gráfica de datos matriciales, tales como batimetrías, salinidad, corrientes, gráficas de temperaturas muestreadas por los AUVs, etc. En este caso, Matlab contiene gran cantidad de toolboxes que facilitan la representación gráfica, tanto en dos como en tres dimensiones. Éstas permiten al usuario configurar la visualización, decidiendo forma, color, etc., como se ejemplifica en la Figura 16.

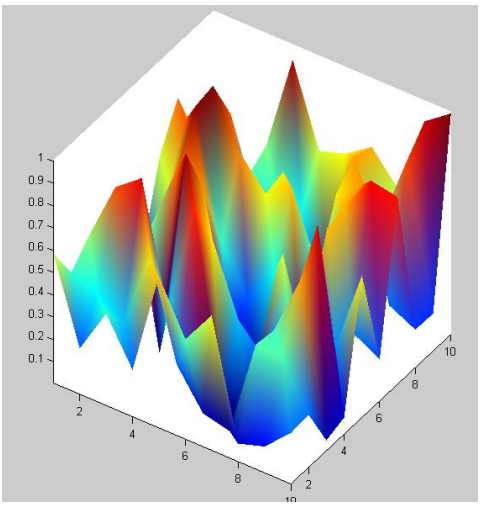


Figura 16: Ejemplo de Gráficos en Matlab.

También se presentan alternativas mucho más logradas como el **Virtual Reality Toolbox**, que permite conectar y controlar un mundo virtual generado en VRML Java 3d o X3D a Simulink o Matlab. En el caso tridimensional, el renderizado es bastante lento.

En cambio, en Java las funciones disponibles no son de tan alto nivel, lo que hace la tarea bastante más compleja al programador a la hora de representación de datos. De cara a representar datos matriciales bidimensionales, presenta la clase **Graphic**, que es una especie de lienzo que ofrece funciones básicas para dibujo de elementos de bajo nivel (líneas, rectángulos, etc) sobre el mismo. Además, presenta el conjunto de clases y funciones **Java 3D**, muy similar al anterior, pero para el dibujo de elementos tridimensionales.

También existen otro tipo de librerías, algunas con licencia, para fines gráficos específicos. Es el caso del **jfreechart**, que permite la representación gráfica de datos, de muy diversas maneras (curvas, diagrama de barras, sectores, etc). Se muestra un ejemplo en la Figura 17.

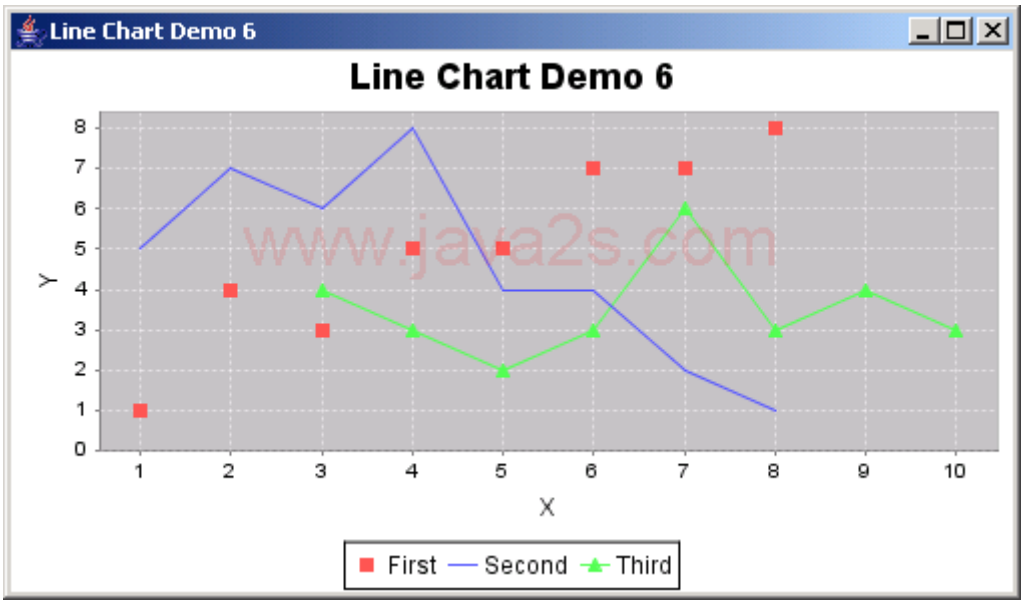


Figura 17: Ejemplo de librería jFreeChart para Java.

4.1.4.- Comunicación vía red.

En Java se da facilidad para realizar aplicaciones cliente servidor con posibilidad de comunicación mediante protocolo TCP / IP. Así, un conjunto de clases ofrecen al programador una gran versatilidad sobre la vía de comunicación. Estas clases incluyen:

- Streams o canales de comunicación.
- Sockets que es un punto final para comunicación, bien un Server Socket, o un Socket normal de cliente. Cada Socket tiene asociados streams de entrada y de salida para la comunicación.

En Matlab, existe una toolbox, llamada **Matlab Web Server** que permite al desarrollador crear aplicaciones que aprovechan la capacidad de comunicación de Internet. De esta forma se pueden enviar peticiones con formularios HTML enviados desde el cliente a través de navegador web a la entrada de Matlab Web Server para que realice cálculos u operaciones definidas y que posteriormente devuelva los resultados y se muestren en pantalla mediante un navegador Web. Se observa este protocolo en Figura 18.

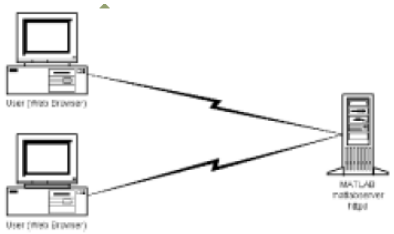


Figura 18: Protocolo de comunicación de Matlab Web Server.

Se observa por tanto una comunicación mucho menos versátil y controlable para el desarrollador que el caso de Java.

4.1.5.- Cálculo matricial.

Matlab contiene multitud de funciones preprogramadas o toolboxes para tareas complicadas o tediosas, como la multiplicación matricial o resolución de ecuaciones diferenciales. En general, Matlab surge inicialmente como una herramienta de cálculo, de ahí las ventajas del uso de Matlab para estos fines, por rapidez y optimización de recursos del sistema.

Java, por el contrario, no es una herramienta de cálculo, por lo que no está optimizado para el cálculo matricial, y en muchos casos es el programador el que debe hacer desarrollos desde cero.

4.1.6.- Programación Multihilos

Java tiene la posibilidad de programar hilos, cosa que no posee Matlab. Esto es una importante ventaja de Java sobre Matlab. Así, una aplicación Java puede modularizarse en distintas aplicaciones o la misma corriendo en distintos hilos, con lo que en entornos multiprocesador, se pueden realizar tareas a la vez, y con ello ejecutar varios módulos al mismo tiempo. Esto es muy importante en interfaces gráficas de usuario, además de para la optimización de procesos y resolver cuellos de botella.

4.1.7.- Interfaz Gráfica

En Java, con las herramientas de programación **IDE / GUI**, como NetBeans o Eclipse, las interfaces gráficas son muy sencillas de desarrollar, y el desarrollador tiene

control total sobre los ítems mostrados. Existen principalmente dos librerías que proporcionan elementos a agregar en una interfaz gráfica, si bien hay muchas otras. Son las librerías **SWING** y **AWT**, que proporcionan gran variedad de objetos incorporables a una interfaz gráfica, como se muestra en la Figura 19.

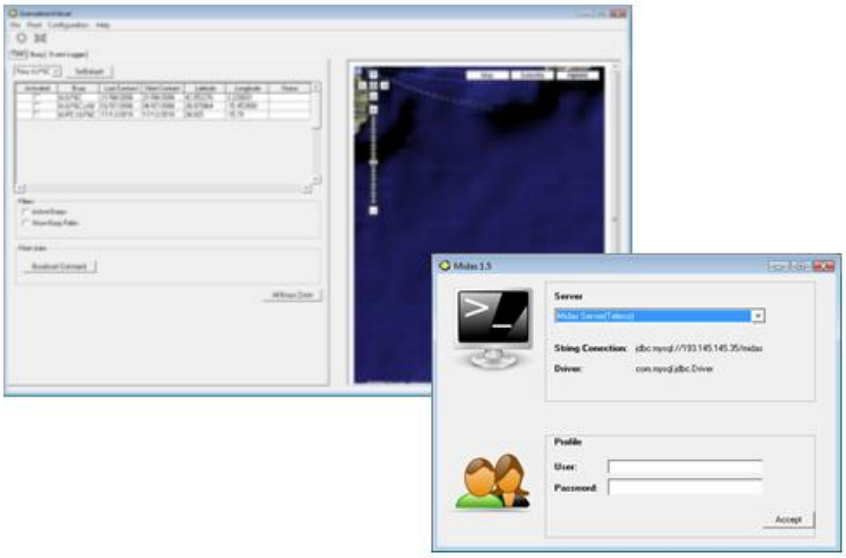


Figura 19: Ejemplo de Interfaz Gráfica de Usuario en Java.

En cambio, el **GUIDE** de **Matlab** está mucho más limitado, dando muchas menos posibilidades al programador y limitando mucho la interfaz. No existen escritorios, y el número de distintos componentes o ítems gráficos a añadir son muchos menos, aparte de que el desarrollador no puede hacer modificación alguna sobre los mismos. Se muestra un ejemplo en la Figura 20.

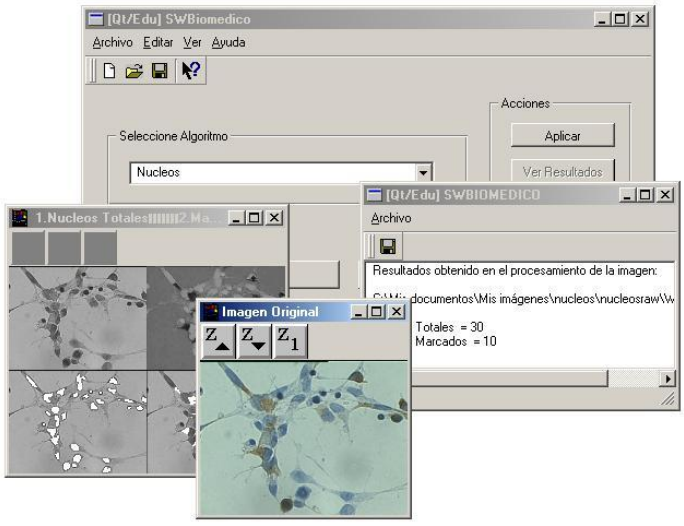


Figura 20: Ejemplo de Interfaz Gráfica de Usuario en Matlab.

4.1.8.- Conexión con dispositivos reales.

Matlab, mediante Simulink, permite la simulación y conexión con dispositivos reales en hardware-in-the-loop. Esto es una importante característica. Sin embargo, en Java esta programación tendría que hacerse a bajo nivel. Por otra parte, el Simulink es un módulo de Matlab con una sencilla interfaz gráfica que permite simulaciones basadas en modelos para sistemas dinámicos y embebidos. Así, sirve para el análisis, diseño, implementación y simulación de sistemas dependientes del tiempo, incluyendo señales, comunicaciones, controles, procesamiento de vídeo, procesamiento de imágenes, etc.

En Java todas estas funcionalidades tendrían que desarrollarse desde cero.

4.1.9.- Interfaz Java Matlab

Finalmente, en ambos entornos de programación existen librerías específicas para poder interactuar con otros entornos. Es decir, que desde Java se pueden ejecutar funciones de Matlab, y viceversa. Esto da mucha versatilidad al sistema, permitiendo, por ejemplo, desarrollar una interfaz gráfica en Java por tener más posibilidades y realizar los cálculos matriciales en Matlab por ser más rápido y optimizar recursos. Para invocar desde Java a funciones de Matlab, existen diversas alternativas, como son: JLab, JMathLink, J-Integra, y Engine.

4.1.10.- Conclusiones sobre programación.

Según lo analizado, se toma como decisión el uso de Java de forma generalizada como entorno de programación de cada elemento del simulador. En la Tabla 1 se muestra un resumen:

Característica	Java	Matlab
Multi-plataforma		
Licencia		

Orientación a Objetos		
Herramientas representación gráfica de matrices de datos		
Comunicación en Red		
Cálculo Matricial		
Programación multihilos.		
Herramientas para Desarrollo de Interfaces Gráficas de forma sencilla.		
Conexión con dispositivos reales (hardware-in-the-loop).		

Tabla 1: Resumen del estudio de entornos de programación.

4.2.-Tecnología para intercambio de paquetes.

Dado que va a proponerse para este proyecto un simulador cliente / servidor, es importarte estudiar un formato para el intercambio de paquetes a través de TCP/IP. Se han considerado las siguientes características:

- **Flexibilidad**: Capacidad para representar todos los tipos de datos que van a comunicarse entre clientes y servidor.
- **Multiplataforma**: Se pretende que la comunicación entre Servidor y Clientes sea independiente de la tecnología de programación utilizada. Es por ello que el formato tiene que poder interpretarse con independencia del lenguaje de programación que se use.
- **Eficiente**: Representar los datos en el menor espacio posible, para que la transferencia a través de la red sea lo más rápida posible.

- **Robusto**: La lectura de los datos que se envían a través del protocolo elegido debe ser unívoca, no dejando lugar a errores de interpretación, ni por el servidor ni por los clientes.
- **Estándar**: Es deseable que sea estándar, y no un formato a medida, para que este proyecto tenga proyección. Esto facilita que esas entidades externas sean capaces de comunicar con el simulador y así poder escalarse el proyecto.

4.2.1.- Estudio de XDR: "External Data Representation"

Es un estándar para descripción y serialización de datos, ideal para intercambio de datos entre máquinas don distintas arquitecturas. A colación del **Apartado 4.1**, a las características de Java se suma el hecho de que los datos en Java ya están representados en su mayoría en XDR, con lo que la serialización e interpretación de los paquetes en XDR es casi inmediata. XDR también usa un lenguaje muy parecido a C que sirve para describir formatos de datos. Tanto el formato en sí como el lenguaje vienen descritos en [**RFC1832]**.

En el marco de este proyecto, se plantea el uso de este estándar para comunicación de datos, tanto entre entidades remotas del mismo proyecto, como para la comunicación con otras aplicaciones que tengan algún tipo de interacción con los módulos que integren el simulador.

Para una descripción más completa, ver el documento "**Descripción Técnica de Formatos de Datos Utilizados** anexo al proyecto**.**

4.2.2.- Estudio de XML: "Extensible Markup Language"

Se pueden consultar las referencias [**XML11**], [**XMLSCH**] y [**LABXML**] para más información. XML es un metalenguaje extensible de etiquetas que ofrece un formato para la descripción de datos estructurados. Esto facilita unas declaraciones de contenido más precisas y unos resultados de búsquedas más significativos en varias plataformas. Es una simplificación de GSML. Además permite definir la gramática de lenguajes específicos. XML habilita una nueva generación de aplicaciones para ver y manipular datos basadas en el Web.

Aparte de su evidente aplicación en Internet, es un estándar para intercambio de información estructurada multiplataforma de forma fiable, segura y fácil. Algunas ventajas de utilizar XML para intercambio de información son:

- Los autores y proveedores pueden diseñar sus propios tipos de documentos usando XML, en vez de limitarse a HTML. Los tipos de documentos pueden ser explícitamente 'hechos a la medida de una audiencia', por lo que las difíciles manipulaciones que exige el uso de HTML para conseguir efectos especiales serán cosa del pasado: autores y diseñadores serán libres de inventar sus propias etiquetas;
- La información contenida puede ser más 'rica' y fácil de usar, porque las habilidades hipertextuales de XML son mayores que las de HTML.
- XML puede dar más y mejores facilidades para la representación en los visualizadores.
- Elimina muchas de las complejidades de SGML, en favor de la flexibilidad del modelo, con lo que la escritura de programas para manejar XML será más sencilla que haciendo el mismo trabajo en SGML.
- La información será más accesible y reutilizable, porque la flexibilidad de las etiquetas de XML pueden utilizarse sin tener que amoldarse a reglas específicas de un fabricante, como es el caso de HTML.
- Los archivos XML válidos son válidos también en SGML, luego pueden utilizarse también fuera de la Web, en un entorno SGML, una vez la especificación sea estable y el software SGML la adopte.

Sin embargo, no es un formato pensado para contenedor de datos. Más bien es un formato para generar ficheros descriptivos.Algunos ejemplos de aplicaciones donde se ha empleado XML son:

- CDF (Channel Definition Format): Los canales creados por Microsoft en el explorador IE4 con tecnología push.
- RDF (Resource Description Framework): Esquema de descripción de recursos. Una de las aplicaciones más importantes que permitirá describir los datos de cada documento y definir las relaciones que hay entre los datos XML. Tratará de los metadatos (metadata). Se les podría considerar como "los META del XML". Muchas compañías en Internet se están adhiriendo a esta aplicación. RDF Posee las siguientes virtudes:

  - Mejores motores de búsqueda. Se han adherido a esta especificación Yahoo!, Altavista, Excite, Lycos, WebCrawler, Amazon, etc.
  - La capacidad de describir los contenidos y sus relaciones en una biblioteca digital o sede Web. Permitirá el acceso a una parte concreta del documento y se facilitará el intercambio de los datos.
  - Se pueden calificar los contenidos para establecer la protección infantil y de la propia intimidad, desarrollado a través de las marcas (tags) de PICS (Platform for Internet Content Selection).
  - Establece los derechos de propiedad intelectual en las propias páginas Web.

- OSD (Open Software Description Format): Formato abierto de descripción de software para el desarrollo de software en múltiples plataformas. Describe el reparto de software a través de la Red. Las etiquetas XML con las que está descrito definen los componentes, la versión, la plataforma en la que ha sido creado, la relación con otros componentes, etc. Esto hará que se simplifique el proceso de instalación para el usuario y permitir también un fácil uso de las actualizaciones.
- CML (Chemical Markup Language): Lenguaje de marcas para química. Permite describir fórmulas, estructuras moleculares y cristalinas, los análisis de espectros y otros objetos de interés para los químicos.
- MathML (Mathematical Markup Language): Lenguaje de marcas para matemáticas. Apto para codificar signos matemáticos, símbolos científicos, etc. El MathML es un lenguaje de bajo nivel que tiene en cuenta la comunicación maquina a maquina de datos estructurados como información de bases de datos. El lenguaje MathML utiliza dos series de códigos progresivos: el primero presenta los signos matemáticos en series crecientes, y el segundo transmite el significado semántico de las expresiones matemáticas, lo que posibilita la codificación de símbolos y signos tanto matemáticos como científicos.

En el marco del proyecto que se presenta, XML tendrá utilidad para:

- representar la especificación del AUV
- escribir misiones

- generar ficheros de configuración de las aplicaciones que se diseñen e implementen. La carga de las aplicaciones irá antecedida por la lectura e interpretación del correspondiente fichero de configuración.

Para una descripción más completa, ver el apartado correspondiente en el documento anexo al proyecto **Descripción Técnica de Formatos de Datos Utilizados.**

4.2.3.- Estudio de RMI: "Remote Method Invocation"

Se ha seguido la referencia [**JAVATUT**] para este apartado. RMI es una API de Java que permite a un objeto que se está ejecutando en una máquina virtual de un equipo invocar métodos de otro objeto que se encuentre en otra máquina virtual distinta. Así, esta API de Java facilita la creación de aplicaciones con objetos distribuidos, proveyendo:

- Mecanismos de localización: registrar objetos remotos y publicación de los mismos. Pasar y devolver referencias a objetos.
- Facilidades de comunicación: llamada igual a llamada a un método de un objeto local.
- Una semántica para permitir invocar métodos de objetos remotos.
- Mecanismos para carga dinámica de clases: en caso de que el cliente no posea una clase que use el objeto remoto.
- Preserva en lo posible la semántica Java, con lo que objetos remotos y locales interaccionan de una manera muy similar.
- Facilita el desarrollo de aplicaciones distribuidas Java.
- Resuelve cuestiones de seguridad en la comunicación de componentes.

Aunque es una solución pensada sólo para comunicar componentes Java entre sí, se considera importante analizar esta API, pues si bien no va a poder resolver el intercambio de paquetes de Servidor / Cliente (dado que Cliente puede no ser Java), sí que es importante si se quiere diseñar el Servidor o un Cliente Java de forma distribuida en la red, con módulos remotos. Esta API resuelve de forma sencilla la comunicación entre componentes internos de Servidor y Cliente distribuidos en distintas máquinas.

Existen dos roles en una comunicación RMI, como se muestran en Tabla 2:

Servidor	Cliente
Crea objetos remotos	Obtiene una referencia de uno o más objetos remotos del Servidor
Hace accesible referencias a objetos remotos	
Espera a que los clientes invoquen a estos objetos remotos o a sus métodos	Invoca sus métodos
El servidor usa los **skeletons** para hacer accesibles los objetos remotos al cliente.	Cliente invoca a objetos remotos usando **stubs (**representante local de un objeto remoto). Éste permite la invocación de métodos del objeto remoto como si fuera uno local, encapsulando la complejidad.

Tabla 2: Comparación entre el Servidor y el Cliente en protocolo RMI.

Para una descripción más completa sobre cómo establecer una comunicación RMI en Java, ver el apartado correspondiente en el documento anexo al proyecto **Descripción Técnica de Formatos de Datos Utilizados.**

CAPÍTULO 5.- *Arquitecturas de Simuladores de AUVs*

El objetivo que se persigue en este capitulo es describir los principales simuladores existentes en la actualidad y sus arquitecturas, para finalizar haciendo un estudio analítico de los distintos métodos de sincronización utilizados en simuladores con arquitecturas distribuidas.

5.1.- Paradigmas de Simulación de Múltiples Agentes

En sistemas distribuidos con múltiples agentes, como es el caso del simulador que nos ocupa en este proyecto, donde cada AUV simulado o real es un agente, se manejan principalmente dos clases de simulaciones:

- Microsimulación o simulación basada en agentes: se modelan los agentes individualmente, y el comportamiento del sistema surge de la interacción de los individuos. Son más apropiadas en dominios con gran grado de localización y distribución, y dominado por decisiones discretas**. [PARU98]**
- Macrosimulación o modelación basada en ecuaciones: se modela matemáticamente el conjunto total de agentes. Es apropiada en sistemas que pueden ser modelados de forma centralizada, y en las que el "movimiento" de los agentes es provocado por leyes físicas más que por el procesamiento de información de cada entidad. **[PARU98]**

Algunos ejemplos de arquitecturas tradicionalmente utilizadas para simular sistemas son:

- **Simulación discreta de eventos paralela y distribuida (DES: Discrete Event Simulation) [FUJI99]**
    - La simulación se basa en los eventos que se disparan y desencadenando los efectos que estos eventos tienen asociados. .
    - Los cambios de estado del mundo suceden en distintos instantes de tiempo, y son causados por eventos.

- Se puede ver como una simulación continua donde los cambios de estado se dan continuamente en el tiempo.
- Se puede hablar de:
  - **DES guiado por eventos**: se usa una lista ordenada de eventos donde los eventos, con su timestamp son almacenados. Se progresa ejecutando de la lista los eventos con el timestamp más temprano, y eliminándolos de la lista después.
  - **DES guiado por tiempo:** el tiempo se avanza en una cantidad constante (tiempo de ciclo). En cada ciclo se ajusta tiempo y los agentes son informados del nuevo tiempo. Cada agente revisa que tareas tenía que haber realizado en ese tiempo, y las ejecuta.
- DES es muy importante en MABS, ya que sus procesos lógicos o unidades son objetos activos con su propio control de flujo, como agentes autónomos.
- Hay dos aproximaciones:
  - **Simulación síncrona de procesos**: un reloj global coordina los procesos lógicos. El tiempo de ciclo de la simulación es fijo y avanza con el tiempo. En cada ciclo, cada Proceso lógico debe ejecutar todas sus acciones.
  - **Simulación asíncrona de procesos**: permite que aparezcan eventos en distintos puntos temporales. El control es más complejo y pueden aparecer errores causales, que se pueden manejar de forma optimista - tiempo avanza aunque hayan errores causales - o conservativa - solo ejecuta eventos libres de conflictos-.

- **Simulación orientada a Objetos (OOS: Object Oriented Simulation) [PAGE91]**
  - Cada agente es un objeto, con lo que se simplifica bastante la simulación.

- Los agentes son autónomos y proactivos, que usan el paso de mensajes como paradigma de comunicación.
- Dado que objetos son puramente reactivos, usan invocación de métodos y son modelados en base a atributos y métodos. Es así como se coordinan.

- **Simulación microdinámica (DMS: Dynamic Micro Simulation) [HARD96]**
  - Simula el paso del tiempo en cada individuo o agente.
  - Para simular cómo cambiaran estas características con el tiempo, se usan funciones de probabilidad.
  - Realmente esta simulación se usa más bien para simulaciones demográficas, por lo que se señala para comprender la simulación Basada en MultiAgentes (MABS).

- **Simulación Basada en MultiAgentes (MABS: Multi Agent Based Simulation) [MABS00]**, **[BRAU04]**
  - Las entidades simuladas son modeladas e implementadas en términos de agentes. No son modelos globales, sino que se modela cada individuo.
  - El comportamiento de cada entidad es determinado por un conjunto de reglas. Al suceder eventos en el entorno, se generan mensajes que activan algunas de las reglas de los agentes, haciendo que estos agentes ejecuten una serie de acciones como consecuencia de las reglas.
  - El comportamiento del sistema global surge de la interacción de individuos (microsimulación).
  - Parte de los modelos anteriores:

- **OOS**: MABS es la continuación natural de la arquitectura orientada a objetos. Las características en la Tabla 3 diferencian a los sistemas, aunque el problema es establecer en qué límite éstas separan a las dos arquitecturas.

OOS	**MABS**
Entidades puramente reactivas	Entidades ProActivas
Sin comunicación, solo por llamadas a función.	Comunicación mediante un lenguaje
Sin noción del espacio	Cada Entidad tiene posición en espacio.
Entidades estacionarias	Entidades se mueven en entorno físico.
Entidades no se adaptan al entorno.	Entidades aprenden autónomamente.
Conceptos de modelado tradicionales.	Modelos con conceptos como creencias, deseos e intenciones.

Tabla 3: Comparación entre OOS y MABS

- **DES**: En entorno distribuidos con múltiples agentes, tiene muchas más ventajas la arquitectura MABS que la DES; esta última quizás más apropiada para macrosimuladores.

  MABS tiene la ventaja que es una arquitectura orientada a los agentes, lo que exige una estrecha relación entre las entidades reales, los modelos, y las entidades software de simulación. Esto simplifica diseño e implementación del software, frente al DES, más cercano a una macrosimulación.

  Así, permite modelar comportamientos individuales proactivos y diseñar sistemas distribuidos de forma muy natural, ya que cada agente es una pieza, agregar o remover agentes durante la simulación

  Por otro lado, una arquitectura MABS consume muchos más recursos, lo que se refleja en simulaciones más lentas. Además, siendo apropiada para simulaciones guiadas por tiempo,

no lo es tanto en simulaciones guiadas por eventos, donde un esquema centralizado es mejor.

- **DMS:** la primera limitación con respecto a la arquitectura MABS es que cada individuo se modela en función de probabilidades, no considerando planes, preferencias, etc. La segunda es que cada agente es modelado individualmente, sin considerar la interacción con otros. MABS facilita el estudio de comportamientos emergentes de la interacción individual.

Por otra parte, otro paradigma habitualmente utilizado en simulaciones es la **Simulación de Tiempo Real.** Ya no responde a la necesidad de un sistema distribuido multiagente, sino la necesidad de que el tiempo de respuesta del sistema sea igual o similar al tiempo que llevaría la respuesta en el sistema real que representa la simulación. Es decir, el tiempo que va desde que se produce la entrada al sistema, pasando por el tiempo de cálculo de respuesta y comunicación de la misma vía red, hasta que se da la respuesta al usuario, debe ser prácticamente nula, permitiendo dar al usuario la sensación de inmediatez.

Sin embargo, nunca se puede alcanzar un tiempo de respuesta nulo, con lo que el objetivo es minimizarlo, atendiendo a:

- **Hardware**: a más potente y rápido, menor tiempo de respuesta.
- **Arquitectura del Simulador**: ésta influye en los tiempos de respuesta, dependiendo del flujo de datos y diseño de los componentes del sistema.
- **Optimización de los modelos matemáticos** utilizados. Se pueden seguir los siguientes pasos:
  - Seleccionar grados de libertad del simulador, y con ello complejidad del mismo.
  - Reducir el grado de interdependencia entre los grados de libertad.
  - Limitar el rango en que opera cada grado de libertad.
  - Emplear sistemas lineales, con matrices constantes y eficaces métodos de resolución.

  - Evitar sistemas algebraico-diferenciales más complejos de resolver. Para ello intentar linealizarlos relajando las restricciones y haciendo reducciones de variables.
  - Utilizar algoritmos de paso variables, ya que los de paso fijo tienen que adaptarse al peor caso, con la consecuente pérdida de eficacia y rapidez.

- **Simulación Predictiva:** Todo sistema tiene un retraso intrínseco entre la acción o entrada al sistema, y la reacción o salida del sistema. Las razones son varias: el hardware, la velocidad de la red, el software, y la arquitectura del simulador para conseguir reducir el tiempo de respuesta. Una manera de paliarlo es mediante la **Simulación Predictiva,** consistente en predecir las entradas que se van a dar al sistema. Así, se puede calcular la respuesta para la más probable de las entradas futuras.

Conclusión

De las arquitecturas aquí expuestas, la más adecuada al proyecto es la MABS, ya que es la evolución de las otras arquitecturas, y la más adecuada para un entorno de simulación multi agente como es el que se plantea. Así, cada AUV simulado o conectados hardware-in-the-loop será diseñado e implementado como un agente autónomo, capaz de enviar peticiones a un entorno. El comportamiento del sistema surgirá de la interacción en el mundo simulado de los agentes.

5.2.- Arquitectura de Otros Simuladores de AUVs

5.2.1.- Orca [ROY00]

Es un controlador de misiones para AUVs simples, si bien como trabajo futuro, se plantea ampliar el entorno a multiAUVs. Su principal punto es el control inteligente de misiones, que establece planes, y realiza replanificaciones en función del contexto. El simulador es bastante realista, que se define como un sistema adaptativo, sensible al contexto y con un gran motor de inteligencia artificial.

Sistema de razonamiento.

Es un sistema de razonamiento basado en reglas o esquemas, que consiste en que dado el objetivo de una misión y la descripción, genera un plan general de la misión, recuperando de memoria "**esquemas procedurales**" que coincidan con el objetivo, modificándolos luego como sea necesario en función de la descripción de la misión. Los esquemas procedurales codifican el conocimiento asociado a planes, reglas y objetivos.

Definido el plan, lo ejecuta, refinando y añadiendo detalles que sean necesarios, y controla eventos que puedan suceder. Esto se puede hacer gracias al sistema de razonamiento planteado.

También usa un "context-mediated behavior" (CMB) para tener conciencia en todo momento del contexto actual, representado por "**esquemas contextuales**". Los esquemas contextuales representan el conocimiento asociado al contexto o entorno en el que se puede encontrar el programa (representación de sensores, etc.).

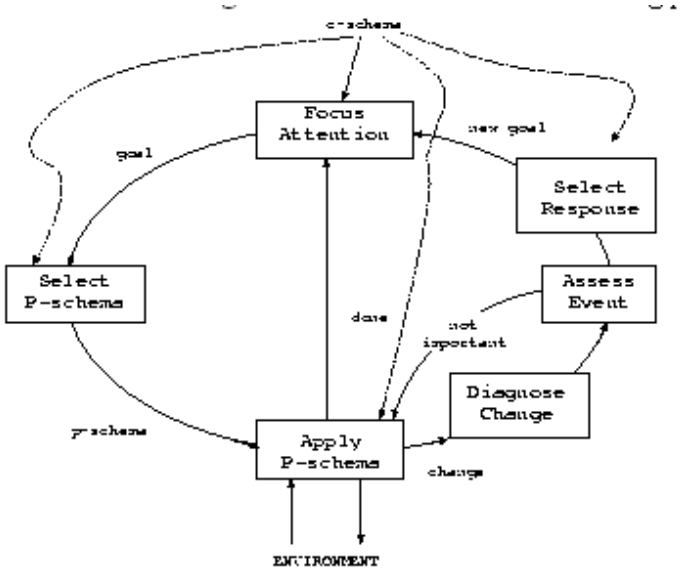


Figura 21: Lógica de razonamiento de Orca.

Como se ve en la Figura 21:

- Los nuevos objetivos son incorporados a la agenda del programa.
- El modulo que se encarga de razonar, centra su atención en un objeto concreto de esta agenda, según prioridades, etc., y sobre todo según el nuevo contexto.
- Seleccionar un esquema procedural que enlace con el objetivo, y lo aplica.

- Se diagnostica el cambio, y en caso de que hayan excepciones no muy importantes, directamente se aplican al contexto o entorno. En caso de que sean importantes, o simplemente como motivo del cambio hecho, se generan nuevos objetivos y con ellos se vuelve el bucle de control.

Arquitectura de bajo nivel

Viene definida por la necesidad de tiempo real, y por el sistema de razonamiento basado en reglas.

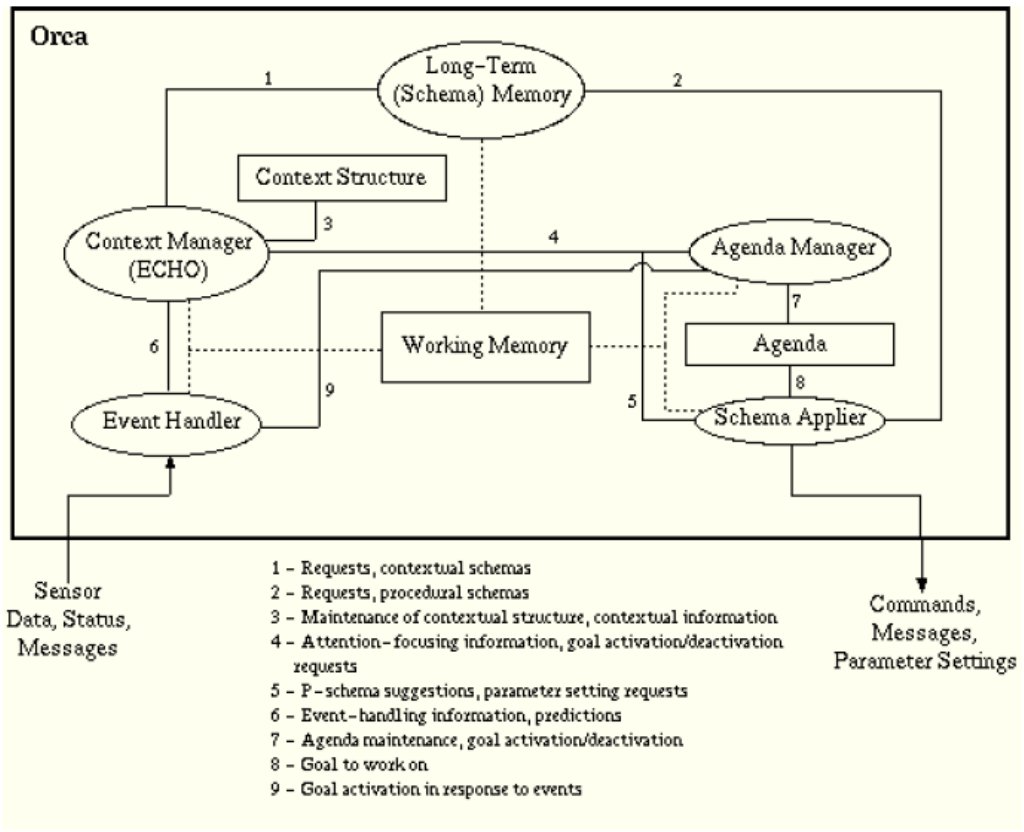


Figura 22: Arquitectura de Orca.

Se organiza en una serie de módulos, como se observa en la Figura 22, que incluye los siguientes elementos:

- **Memoria de trabajo:** estado actual del mundo.
- **Long-term memory:** memoria conceptual y dinámica, basada en el programa CYRUS. Contiene todos los esquemas procedurales y esquemas contextuales, por

tanto es a donde debe acceder cualquier módulo que requiera de estos esquemas, o para cualquier módulo que quiera usar la memoria principal.

- **Schema Applier:** aplica los esquemas procedurales para cumplir los objetivos. Cuando un objetivo pasa al **Focus Attention**, este módulo toma el esquema procedural de la Long-tem memory que mejor cumple este objetivo. En la elección del esquema procedural pueden influir esquemas contextuales del ECHO: representa el contexto actual.
- **Agenda Manager:** mantiene la agenda con el foco de atención u objetivos.
- **Event Handler:** maneja eventos que no se pudieron anticipar, y actúa como filtro de entrada del ORCA. Toda la información de esta arquitectura de bajo nivel entra en este módulo y es revisada. Usa lógica difusa.
- **ECHO:** maneja los esquemas contextuales para crear una representación del contexto actual del programa. No solo esto, sino para crear el comportamiento del contexto, y simularlo.

5.2.2.- SAMON [PHO01]

Permite generar una simulación de alta fidelidad de múltiples UUV (unmanned underwater vehicles) que se coordinan entre ellos. Estos AUVs se jerarquizan, y en función de su rango tienen una instrumentación u otra para permitir unos u otros comportamientos. Los **SUVsn** (supervisory underwater vehicles) tienen capacidad de supervisión y comunicación, repartiendo el plan entre los distintos UUVs de nivel inferior que tienen más cercanos, y ordenando la topología. Los SUVs también se conectan a un TC (tactical coordinator, **coordinador táctico**), que controlan y coordinan la misión entera. Se observa una representación de esta jerarquía en la Figura 23:

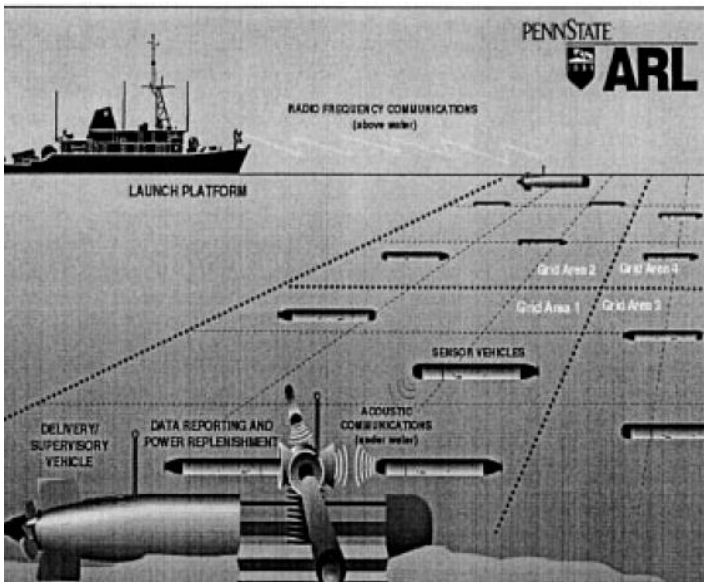


Figura 23: Jerarquía multiUUV

La misión se divide en planes entre los distintos SUVs. Cada plan es un conjunto de tareas que un SUV encomienda a los UUVs subordinados, y éstos descomponen en procesos internos para realizarlas.

Características:

- **Autonomía y escalabilidad de nodos:** cada nodo de una red multiUUV, tiene su propio comportamiento. Esto permite modificar, añadir o quitar comportamiento a nodo o grupo de nodos sin afectar al resto.
- **Composición de comportamientos:** se definen comportamientos básicos, que se combinan secuencial, paralela e iterativamente. Esto permite comportamientos de más alto nivel en la Red de UUVs.
- **Interoperabilidad y colaboración de UUVs:** el comportamiento compuesto de cada UUV puede ser combinado para formar una misión de una red de UUVs. Esto requiere el desarrollo de un lenguaje de control común (CCL) para modelar las interacciones entre UUVs, y un medio común como es el protocolo TCP/IP.
- **Flexibilidad de reconfiguración:** se pueden diseñar comportamientos de un AUV a partir de una librería de comportamientos reusables para cada nodo de la red de

AUVs. Así, el comportamiento entero de un UUV se puede reconfigurar en función de la misión.

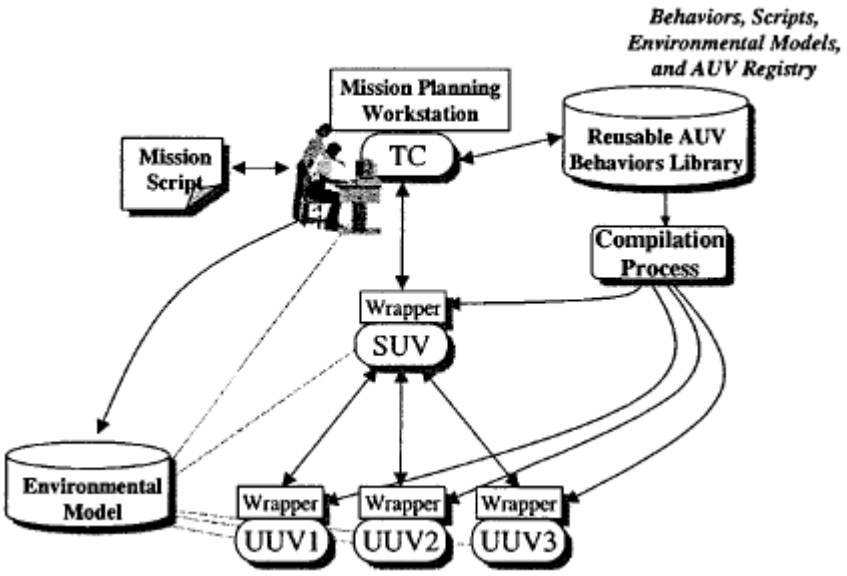


Figura 24: Interconexión de componentes de SAMON.

Arquitectura:

Es una arquitectura basada en Web, y sigue el siguiente esquema mostrado en la Figura 25, donde los distintos módulos se conectan vía Internet.

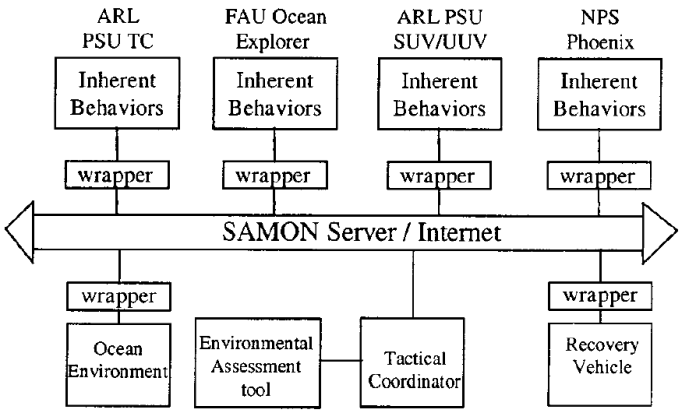


Figura 25: Arquitectura Web de SAMON.

Esta arquitectura preserva una de las premisas del proyecto: las redes de UUVs pueden haber sido desarrolladas de forma independiente, con lo cual, estas redes pueden

ser muy heterogéneas. La arquitectura facilita la cooperación entre redes muy distintas de UUVs. UUVs remotos pueden registrarse y participar en una simulación.

Así, para cada UUV desarrollado de forma independiente, se implementa el código que implementa el comportamiento y además lo que en el diagrama viene nombrado como **wrapper**. El wrapper es una interfaz, que permite que códigos muy heterogéneos puedan interactuar y cooperar en el mismo simulador, no solo códigos del comportamiento de cada UUV, sino también códigos del propio simulador como el entorno etc. Traduce comandos del simulador a comandos inteligibles por el UUV y viceversa.

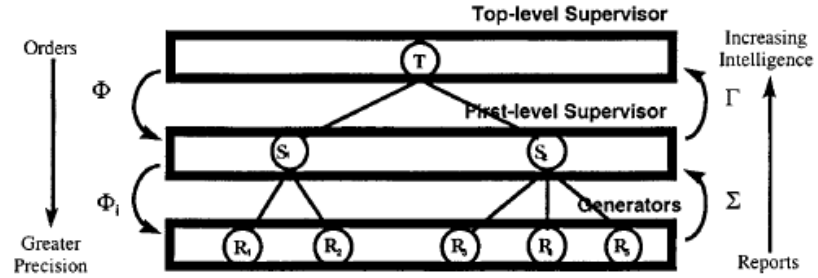
Otro componente, es el **enviromental assessment**, que archiva datos de los sensores de la red de sensores de SAMON, permite acceso a una amplia variedad de bases de datos oceanográficas, datos a diferentes resoluciones, interpola, y visualiza datos que den soporte al control automatizado y toma de decisiones de las aplicaciones para SAMON.

El último componente que merece mención es el **ocean environment**, que provee al simulador de estímulos y respuestas del entorno. Provee batimetría y valores de parámetros oceanográficos pedidos por los sensores. También simularía corrientes oceanográficas, etc.

Arquitectura: innovaciones.

Control basado en comportamientos. Permite UUVs heterogéneos, para coordinar comportamientos compuestos. Se modela como un autómata guiado por eventos: la ocurrencia de eventos determina la transición en el autómata. Cada nodo del autómata dice el comportamiento a seguir por el UUV. El uso de autómatas permite explotar el uso de lenguajes formales.

Básicamente, se usa un autómata para controlar cada nodo de cada red de UUVs, como se muestra en la Figura 26. Es un modelo jerárquico, con 3 niveles de jerarquía, supervisor de nivel superior, o sea, el controlador táctico, el supervisor de primer nivel (SUVs) y los generadores (UUVs inferiores.



Generator $R_j = (Q_j, \Sigma_j, \delta_j, q_{0j}, Q_{mj})$, where Q_j is a set of states, Σ_j is the alphabet, δ_j is a partial mapping from $Q_j \times \Sigma_j$ to Q_j, q_{0j} is the starting state, and Q_{mj} is a set of marked, or final, states.

First-level Supervisor: $S_i = (S_i, \phi_i)$, where $S_i = (X_i, \Sigma_i, \Gamma_i, \xi_i, x_{0i}, X_{mi})$ is a transducer, where X_i is a set of states, Σ_i is the input alphabet, Γ_i is the output alphabet, ξ_i is a partial mapping from $X_i \times \Sigma_i$ to $X_i \times \Gamma_i \cup \{ \varepsilon \}$, x_{0i} is the starting state, and X_{mi} is a set of marked, or final, states, and ϕ_i: $X_i \rightarrow 2^{\Sigma_i}$ is the state feedback mapping.

Top-level Supervisor $T = (T, \phi)$, where $T = (Z, \Gamma, \iota, z_0, Z_m)$ is a recognizer and $\phi : Z \rightarrow 2^{\Gamma}$, is the state feedback mapping.

Figura 26: Lógica de control sobre la jerarquía SAMON.

5.2.3.- OEX [SON03]

Simulador stand-alone.

La base de este simulador es el realismo, tanto en el modelo del entorno, como del vehículo, como de los sensores, usando modelos matemáticos y físicos muy precisos. Por tanto, tiene el simulador una arquitectura muy similar a la que pudiera tener la arquitectura real del AUV. Para este proyecto, se basaron en una arquitectura del software de control del AUV tal como se muestra en la Figura 27

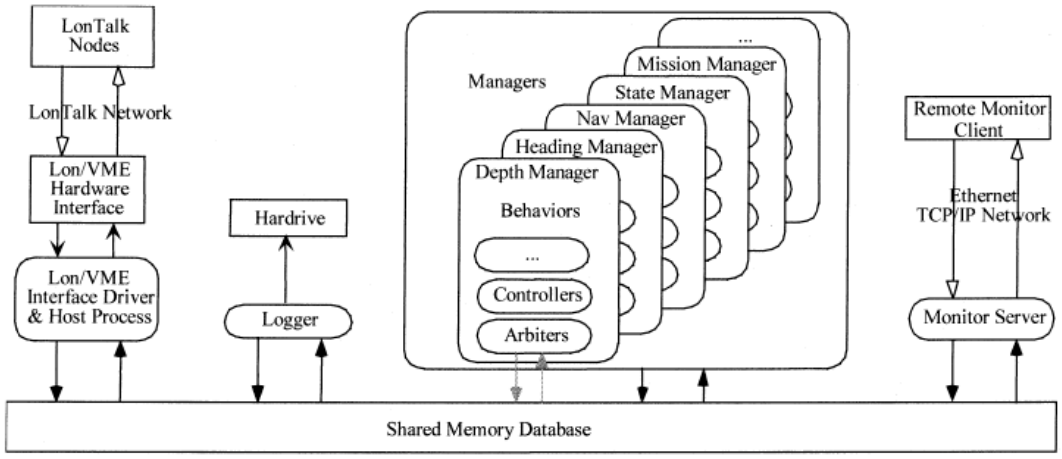


Figura 27: Arquitectura del simulador de OEX.

Cabe destacar de ella el uso de una memoria compartida, al que acceden prácticamente todos los elementos del simulador. Se destaca un grupo de **Managers** que van a ser cada uno de los controladores de tareas del AUV (el controlador de navegación, de estado, de misión, etc), un **logger** que almacena en disco duro. Los **sensores y actuadores**, son tratados aparte, de manera que sus valores son intercambiados vía *Lon Talk Nodes*. Pasan a la memoria compartida a partir del *Lon/VME gateway*. Y finalmente el **Monitor Server** que permite conectar clientes externos para monitorizar el estado del AUV.

La memoria compartida es en general considerada la forma más eficiente de implementar comunicación entre procesos en una plataforma con un único procesador, en contraste con el paso de mensajes. Pues bien, desde el punto de vista del simulador, esto se implementa y esquematiza con una arquitectura muy similar, como vemos en la figura que viene a continuación. Se ve como tanto el modelo del vehículo, como el de sensores, como el del entorno se interrelacionan a través de una memoria compartida, mostrada en la Figura 28, y que implica semáforos y esquemas de sincronización entre consumidores.

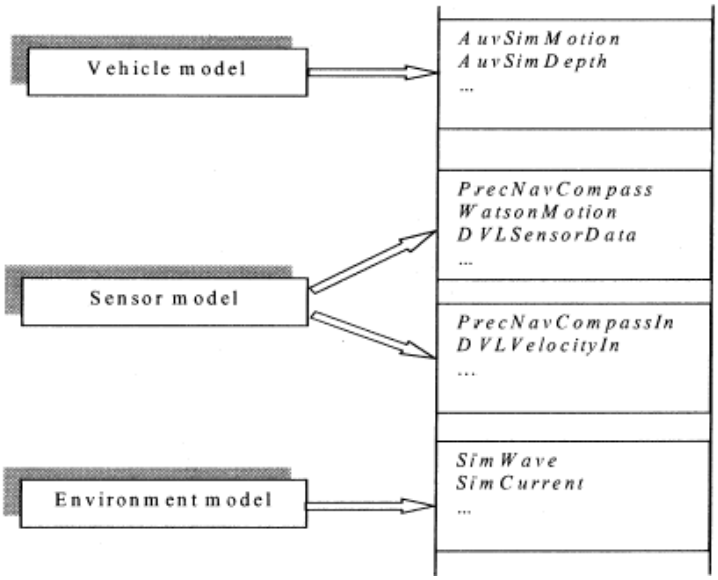


Figura 28: Interacción de modelos con memoria compartida.

El modelo del vehículo es el software de control de vehículo. Esto permite simulaciones realistas y una fácil extrapolación del software al hardware real. Implica la implementación de cada uno de los módulos o **managers** antes vistos, cada uno con un timer asociado, para el trabajo en tiempo real. Las tareas son desarrolladas por hilos independientes, muy similar al esquema de tareas de los sistemas operativos actuales. Los timers son creados como un array de timers a partir del timer real hardware del sistema. Este esquema se muestra en la Figura 29 :

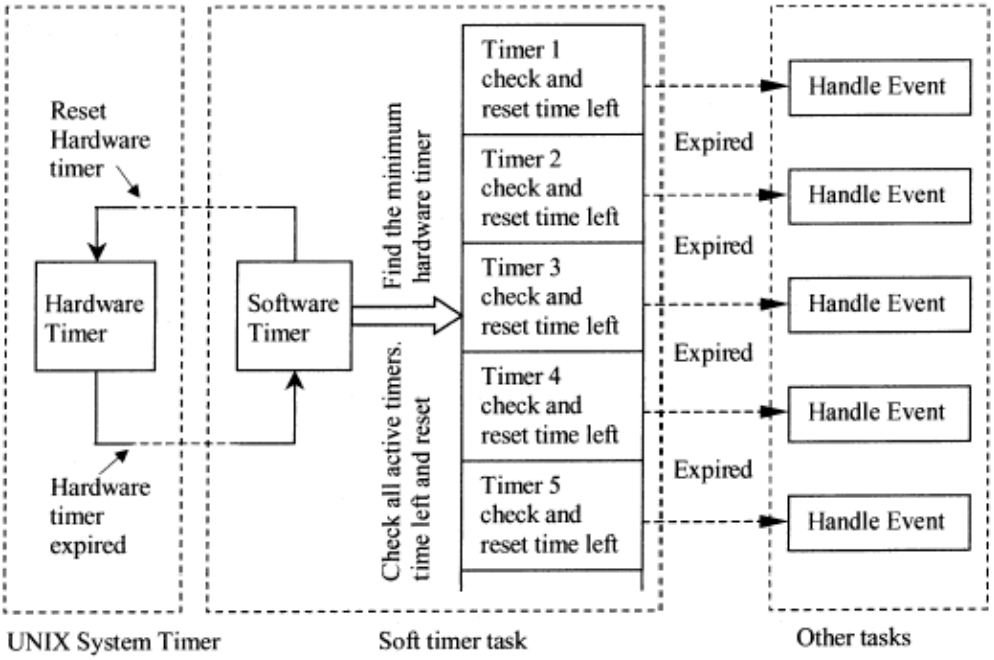


Figura 29: Esquema de tareas y timers en que se organiza OEX.

Simulador HIL:

Este grupo de trabajo también propone otro tipo de simuladores, para hardware-in-the-loop (HIL). La idea es muy similar a la anterior, con la memoria compartida como máximo exponente, pero introduciendo como nuevo elemento el hardware real para la simulación. El esquema de esta arquitectura se muestra en la Figura 30.

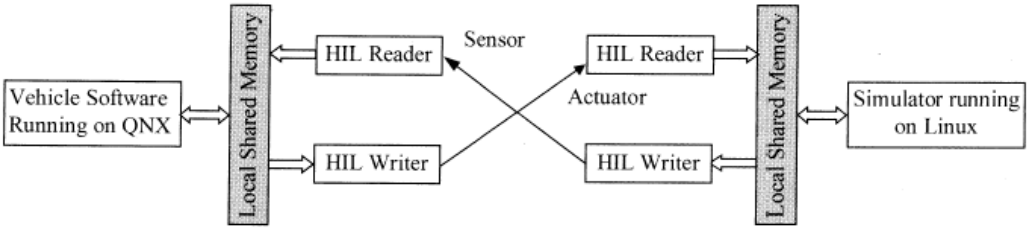


Figura 30: Implementación HIL de un sólo vehículo.

El simulador y el vehículo real residen en sitios diferentes. El simulador tiene su memoria compartida. A su vez, los dispositivos reales tienen su propia memoria compartida, y por tanto hace falta una comunicación entre ambos, que sea veloz y libre de fallos, para que los datos de sensores y actuadores estén actualizados en todo momento en ambos lados. Se usa para la conexión LAN con protocolo TCP/IP. Éste fue el esquema para un único vehículo conectado en hardware. La solución para varios vehículos en hardware se muestra en la Figura 31:

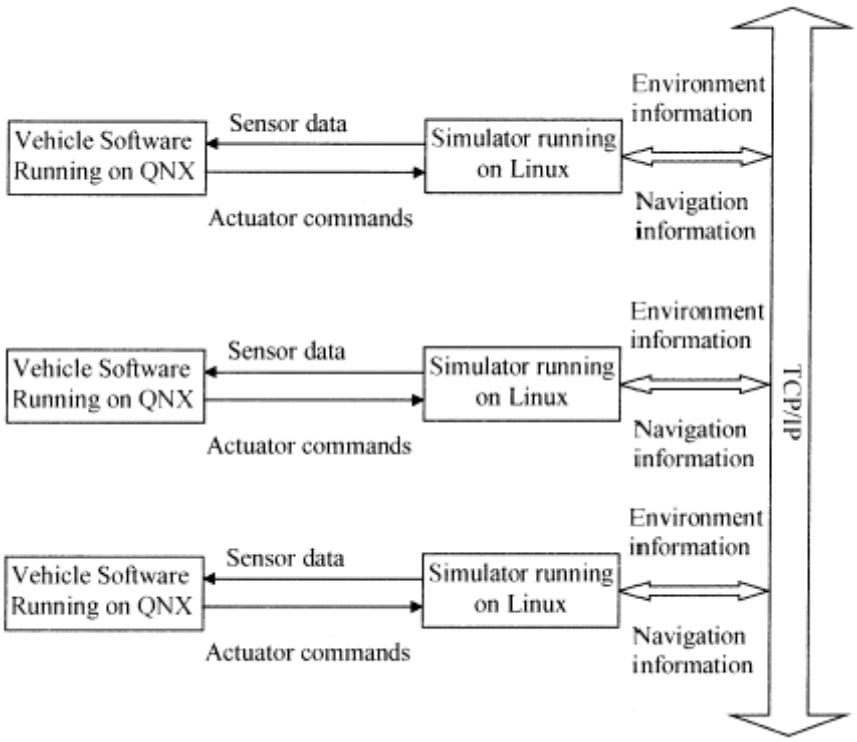


Figura 31: múltiples vehículos HIL conectados al simulador.

Es muy similar al anterior, pero la red une no solo el simulador con el vehículo, sino con varios vehículos, a modo de cliente servidor. Cada vehículo es un cliente y el simulador actúa de servidor de los mismos.

5.2.4.- CODA [ALB03]

Desarrollado por la Universidad de Maine, permite la simulación de **múltiples AUVs**, de forma distribuida. Es un simulador de **múltiple fidelidad**: presenta distintos niveles de abstracción en los que se puede desarrollar la simulación. Aún así, en ninguno de sus niveles presenta una fidedigna representación de los aspectos hidrodinámicos del AUV, ni una exacta simulación del entorno del mismo. Estos múltiples niveles permiten la edición incremental de una misión, partiendo de los niveles más bajos hasta los más altos. También en una misma simulación se pueden simular distintos elementos a distintos niveles de abstracción.

Simula algoritmos de control inteligentes para un solo AUV o redes de AUV. En particular explota este último aspecto, integrando la visión de grupo con la que cada AUV que forma parte de un colectivo con mecanismos multiagente para reorganizarse, resolver problemas y objetivos autónomamente ante distintos medios que se puedan presentar.

En el **Apartado 11.1.1** se describirá mejor cómo aborda CODA las misiones multiAUV.

Implementación

Corre en Linux, y fue implementado con dos lenguajes basados en reglas. Ambas partes se conforman como dos grandes módulos comunicados entre sí a través de PIPES de Unix bidireccionales.

- CLISP: niveles más bajos de abstracción y más fidedignos. Movimiento y comportamiento de AUVs, entorno, y el tiempo de simulación.
- Allegro Common Lisp: niveles más altos de abstracción del simulador. Controla la parte del CLISP y heurística de restricciones entre otros.

5.2.5.- CADCON [ALB03], [CAD]

Desarrollado en el Autonomous Undersea Systems Institute. Sirve para la simulación de **múltiples AUV** con un **nivel alto de fidelidad**. Destaca su detallada simulación de aspectos hidrodinámicos, distintos **modelos de AUV** e incluso distintas **clases de sensores**. Tiene una interfaz amistosa y herramienta de visualización. Permite **hardware-in-the-loop**.

Implementación

Hecho en C y OpenGL, corre sobre Linux y Windows. Se estructura de manera distribuida sobre una arquitectura **Cliente/Servidor** de la forma siguiente:

- *Servidor del Entorno*: simula en entorno, información que brindan sensores y comunicaciones.
- *Cliente de Visualización:* interfaz de usuario y representación 3d del modelo.
- *Clientes:* interaccionan con el servidor para simular un AUV concreto, o para diversas tareas, como un sistema de control de AUVS, o interfaz a otros sistemas. Se puede ver los distintos clientes de la arquitectura **CADCON** en la Figura 32.

Arquitectura:

Es una arquitectura distribuida, usando un modelo cliente / servidor. En este simulador se reconocen 4 componentes básicos:

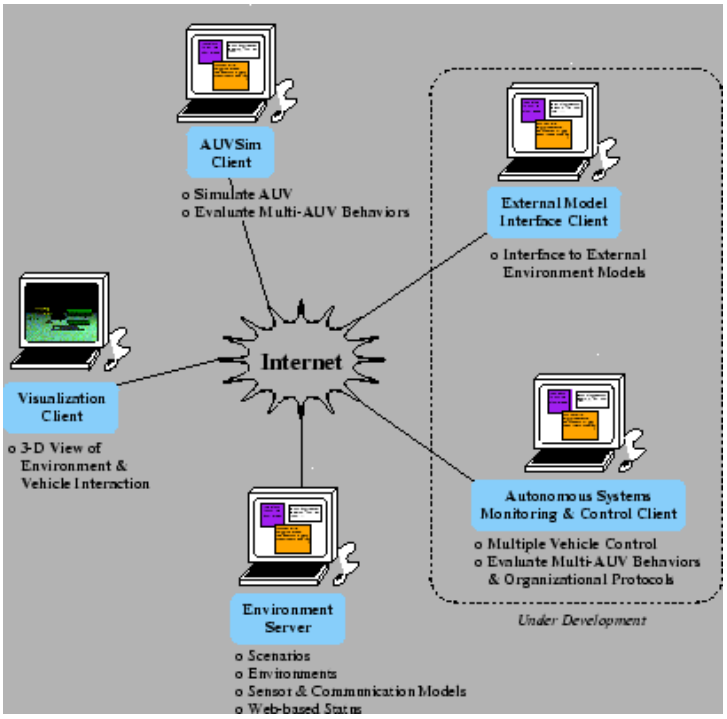


Figura 32: Componentes del simulador CADCON.

- **Servidor del entorno:** es el servidor principal, al que se conectan el resto de elementos. Es el motor que genera los escenarios, entornos, modelo de sensores y comunicaciones, y estado basado en Web.
- **Cliente Simulador del AUV:** es un cliente, que se conecta al servidor del entorno para simular el AUV, y el comportamiento multiAUV.
- **Cliente Visualizador cliente:** se conecta para recoger datos y visualizar tanto el entorno como los AUVs en modo 3D.
- **Cliente para monitorización del estado y control de clientes:** permite el control de múltiples vehículos a alto nivel, y permite evaluar organizaciones complejas de múltiples AUVs.

También puede conectarse un quinto elemento, que sería cualquier simulador externo conectándose mediante una interfaz adecuada a este entorno. Es el caso de la unión entre **CODA** y el simulador **CADCON**.

Variante:

En una universidad de Tokyo, se realizó un sistema también distribuido, pero con múltiples servidores, que desmenuzan por ejemplo, lo que CADCON ha incluido todo en un solo servidor del entorno, en un servidor para el entorno, otro para la dinámica del vehículo, otro para obstáculos, otro para corrientes, etc. A la hora de simular el control de los vehículos, se hace con un esquema que toma decisiones de 2 vías: una del entorno o información procedente de otros sensores, y otra de otros AUVs o información que proviene de comunicación entre AUVs).

5.2.6.- CODA/CADCON [ALB03]

Desarrollado por el Departamento de Informática de Maine. Reúne las mejores características de ambos simuladores. **CODA** es un simulador mucho más orientado a multi AUVs, con técnicas avanzadas de control de misiones conjuntas. Pero su nivel de realismo en las simulaciones es muy bajo. Por ello, como solución a esta representación tan poco fidedigna, se adoptó conectar el simulador **CODA** a **CADCON**, permitiendo un nivel de abstracción mucho menor y por tanto simulaciones más realistas.

Es de múltiple fidelidad, como es CODA, pero siendo más fidedigno en el nivel más bajo de abstracción como el simulador CADCON. Permite desde simulaciones rápidas obviando parámetros reales, a simulaciones en tiempo real de misiones.

5.2.7.- MVS [MVS98]

MVS (MultiVehicle Simulator) es un simulador desarrollado entre 1993 – 1998 por el MVS Developing Group, del "Underwater Robotics & Application Laboratory" en Tokio. Surge esta línea de trabajo de la necesidad de simular un AUV real, el TWIN-BURGER, desarrollado por el mismo equipo de trabajo, sin correr riesgos innecesarios en un entorno real, aparte de simular misiones con múltiples TWIN-BURGER, que evitando el costo de tener una flota real de TWIN-BURGERs. Se puede observar los resultados del proyecto en la Figura 33.

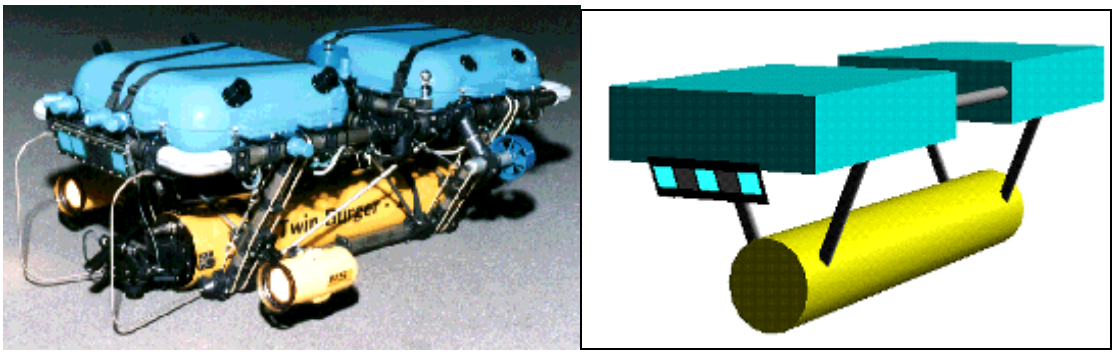


Figura 33: TWIN-BURGER real y virtual

Arquitectura

El proyecto está formado por un grupo de servidores que simulan un entorno submarino a través de servicios accesibles desde Internet. Se puede observar en la Figura 34.

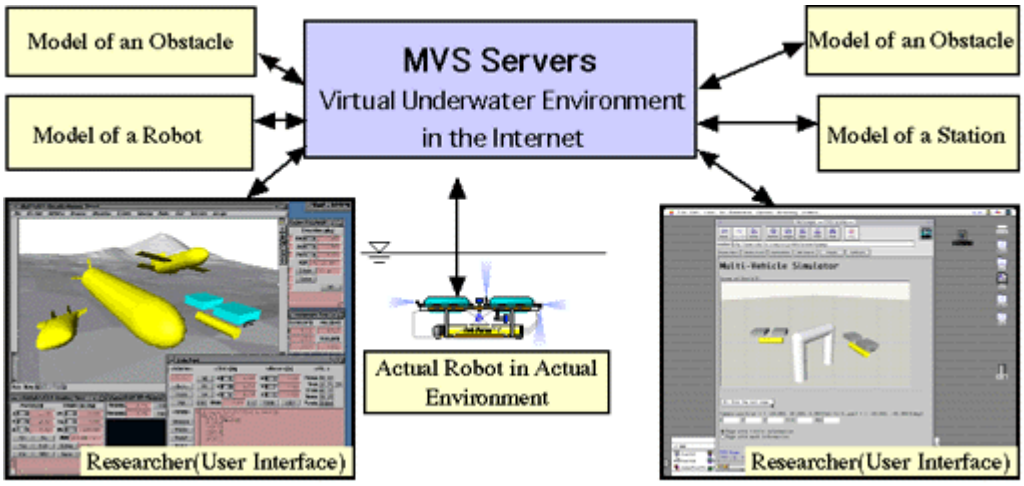


Figura 34: Interconexión de múltiples servidores y conexión con interfaz cliente.

Como se puede observar, a través de Internet sirve entornos virtuales a en los que se pueden conectar modelos de obstáculos, vehículos submarinos (AUVs, ROVs, etc), interfaz de usuario, vehículos reales en un entorno real, etc.

La arquitectura es distribuida como se muestra en la Figura 35, aprovechando Internet como vía de comunicación entre los distintos componentes.

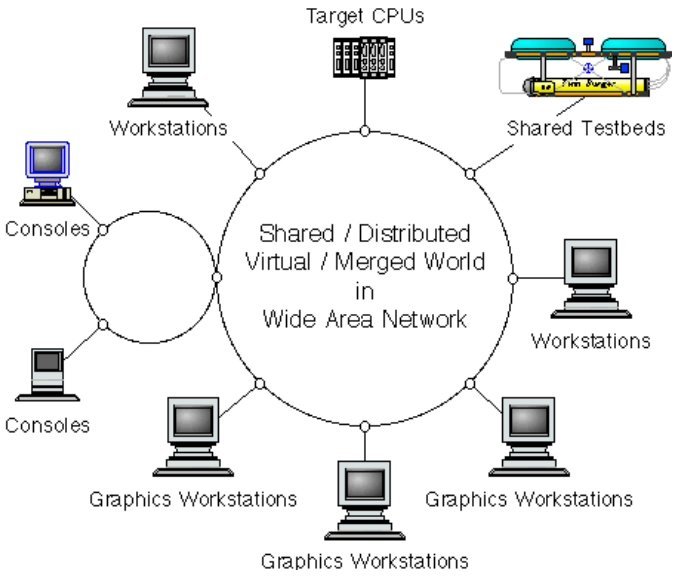


Figura 35: Arquitectura distribuida de MVS.

Permite, por tanto, trabajar con el vehículos submarinos en distintos entornos: desde la oficina, en el laboratorio, o inmerso en entorno real o controlado, como puede verse en la Figura 36 – como pudiera ser un estanque - usando en todos los ámbitos un mundo virtual creado por el MVS.

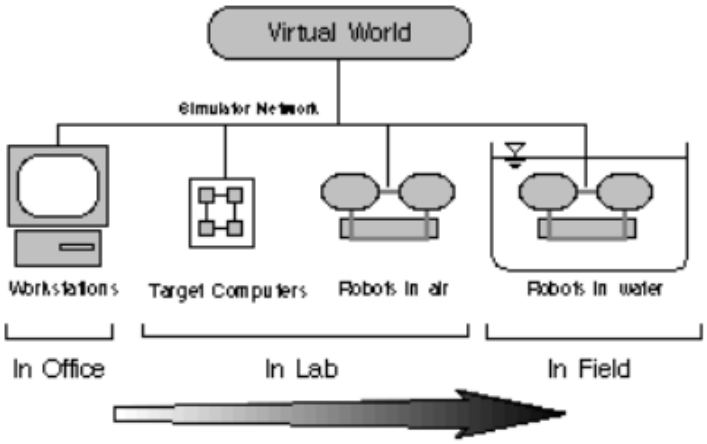


Figura 36: Distintos entornos de trabajo de MVS.

Características:

Fue diseñado con el propósito de cumplir con los siguientes objetivos:

- Observar simulaciones de entorno y de vehículos a través de Internet, mediante navegadores o las interfaces de usuario diseñadas por MVS.

- Implementar algoritmos de control de los distintos vehículos submarinos en modelos de vehículo ya implementados, usando la librería que proporciona MVS para programar.
- Definir nuevos vehículos que no existen en el sistema.
- Definir fenómenos en medios submarinos, mediante la agregación de nuevos servidores al sistema.

Tanto el simulador del entorno virtual, como los modelos de vehículos y la interfaz de usuario permiten gran detalle de simulación, con lo que tiene **un alto nivel de fidelidad**, permitiendo incluso la simulación de vehículos en hardware-in-the-loop. Aparte, es un sistema multiagente que permite simular misiones conjuntas de vehículos submarinos.

Algunos ejemplos de aplicación práctica de un simulador a experimentos reales son:

- Misiones cooperativas para un sistema multivehículo submarino. En el ejemplo, Figura 37, los conos rojos son objetivos y se simulan 5 AUVs.

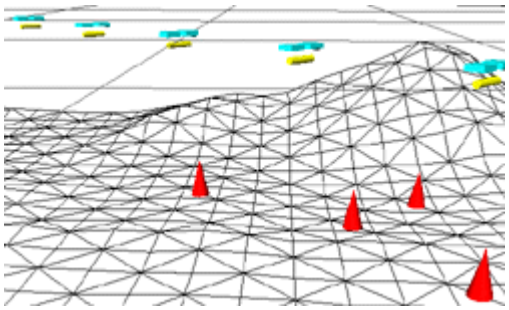


Figura 37: Ejemplo de misión cooperativa multiAUV.

- Mapeado de superficies por sensores ultrasónicos. En la imagen derecha de la Figura 38 se muestra el objetivo a detectar y en la izquierda el resultado del reconocimiento. El robot decide el plan de acción para tomar la información desde distintos puntos.

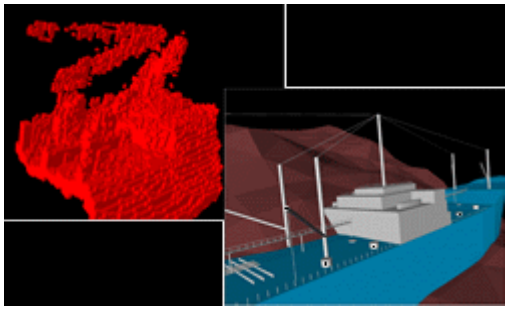


Figura 38: Ejemplo de mapeado mediante sensores ultrasónicos.

- Simulación de un vehículo real - en el ejemplo del Twin- Burger - conectado hardware-in-the-loop y en una piscina real, evitando obstáculos de un entorno virtualizado. Con esto se consigue ahorrar costos en preparar escenarios de prueba, como en la Figura 39.

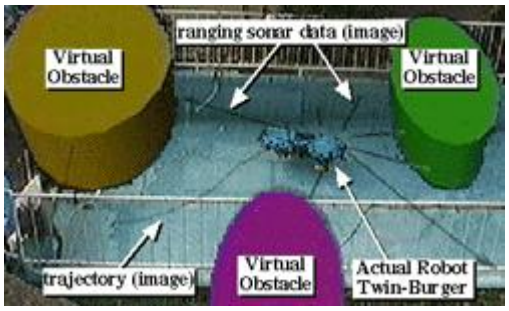


Figura 39: Simulación HIL de Twin-Burguer.

- Tests de algoritmos para evitación de obstáculos, como el mostrado en la Figura 40, primordial en las misiones reales. El simulador permite probar algoritmos, que sería muy costoso probar en la realidad.

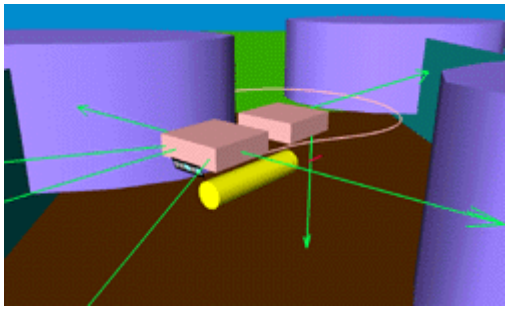


Figura 40: Ejemplo de evitación de obstáculos sobre un simulador.

- Seguimiento de altura constante en robots de tipo crucero o torpedos, como en la Figura 41.

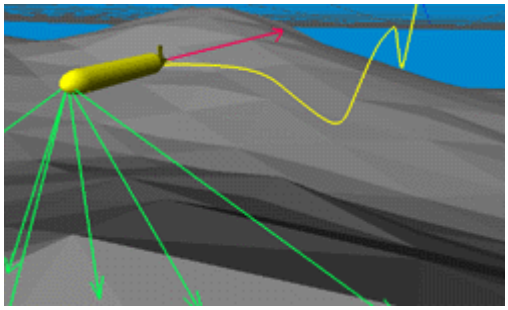


Figura 41: Mantenimiento de distancia al fondo simulado.

- Evaluación de dinámica de movimiento de los planes de los vehículos submarinos.
- Control remoto de estos vehículos submarinos.
- Revisar simulaciones anteriores usando la interfaz MVS. En la Figura 42, las líneas rojas muestran la trayectoria del Twin- Burger simulado. En verde se muestran lecturas de los sensores de ultrasonidos.

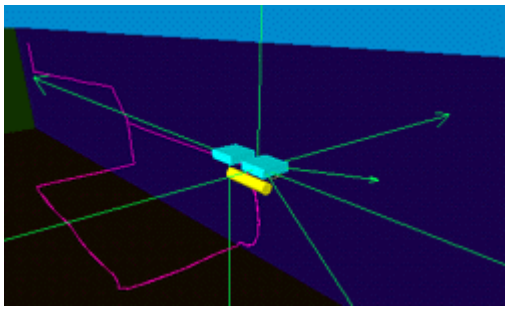


Figura 42: revisión de misión simulada en MVS.

Este simulador es, por tanto, un buen ejemplo de las posibilidades que brinda un simulador de entornos, vehículos y obstáculos submarinos en un proyecto con robots submarinos.

5.2.8.- Conclusiones

Los distintos simuladores en el mercado hacen una contribución muy valiosa al presente proyecto, sirviendo como guía y evitando partir de cero.

- ORCA: se aleja del objetivo del proyecto, al tratar con detalle el trazado de planes y misiones de los que se encargaría un planificador, y no tanto el simulador que se pretende implementar.

- SAMON: este proyecto aporta dos conceptos claves.
    - Autómatas: el uso de autómatas de estado para representar el funcionamiento de AUVs simplifica el uso de modelos matemáticos y físicos complejos.
    - Los "wrapper", elementos que permiten conectar al sistema elementos de muy diversa índole, funcionando como "middleware" o interfaz de integración. Esto aumenta notablemente la escalabilidad y potencia del sistema.
- OEX: es un simulador fidedigno que aporta una solución para simulación en tiempo real de AUVs, necesaria por ejemplo a la hora de simular hardware-in-the-loop, incluyendo modelos realistas y complejos de sensores y parámetros del entorno submarino. Sin embargo, este proyecto no es tan ambicioso, siendo una primera aproximación.
- CODA /CADCON: De todas las arquitecturas revisadas, esta es la que mejor se adapta a la filosofía y objetivos del presente proyecto. El esquema distribuido aporta escalabilidad, permitiendo realizar una simulación con resolución y nivel de abstracción adecuados a los objetivos del proyecto, sin cerrar las puertas en el futuro a la integración de módulos con modelos más realistas y a bajo nivel.

  Por otro lado, dado que el proyecto se desarrolla en un grupo de proyectos que abordan la temática de los AUV, es imprescindible no hacer un sistema cerrado y monolítico.
- MVS: aparte de aportar al proyecto el esquema distribuido y escalable, también ilustra novedosas aplicaciones prácticas del simulador.

5.3.- Sincronización de Procesos en Arquitecturas Distribuidas

La sincronización en aplicaciones no distribuidas, que se ejecutan sobre un mismo ordenador, es muy sencilla y rápida: todos pueden usar el reloj del sistema como referente. Sin embargo, a la hora de sincronizar la actividad de procesos distribuidos, hay que considerar:

- Dos ordenadores diferentes no están sincronizados. Salvo cuando se disponga de hardware específico, que permita el intercambio de una señal física entre ordenadores, en un mismo instante dos máquinas pueden tener un tiempo diferente.
- Los datos a enviar por una máquina A en el instante t_{1A}, llegan al ordenador B en un instante t_{1B}, tras un cierto lapso temporal, tiempo que tarda en viajar los datos por la red, en principio desconocido.
- Además, la velocidad o frecuencia con la que avanzan los relojes de las máquinas A y B no tienen por qué ser la misma.
- Aun resueltas las anteriores dificultades, dos aplicaciones distribuidas pueden tener tiempos característicos de ciclo diferentes: no es lo mismo el tiempo de ciclo en una aplicación de visualización que requiera gran frecuencia de refresco, que una aplicación que requiera mucho cálculo.

Los métodos más comunes para sincronizar procesos son:

- **Sincronización Hardware**: es el más fiable. Una señal física se transmite instantáneamente entre todos los ordenadores que ejecutan su ciclo simultáneamente. Pero requiere hardware y software específico, y en el caso de un ciclado síncrono, la frecuencia puede ser como máximo la del proceso más lento.
- **Ausencia de sincronización**: Es la opción más sencilla. Si la fiabilidad de la red se supone elevada, y la exactitud temporal entre los procesos que participan no es clave para la simulación, entonces es la opción más adecuada. Consiste en considerar que todos los procesos tienen el mismo reloj, sin desfase, a la misma frecuencia y con una tardanza de comunicación de datos mínima.
- **Sincronización software**: Se envían al principio o de forma periódica, mensajes desde un gestor central o principal al resto y viceversa, con temporal, lo que permite por repetición calcular la tardanza y el desfase de las líneas temporales, permitiendo hacer los ajustes pertinentes. Esto da bastante fiabilidad a la sincronización de los procesos.

Hay dos paradigmas clásicos en la sincronización de procesos:

- **Sistema síncrono:** todos los clientes y el servidor compartieran el mismo reloj.
    - **Ventajas**: Es el sistema más sencillo de implementar.
    - **Desventajas**: En este proyecto concreto, donde un objetivo es obtener un simulador capaz de permitir clientes simulados así como reales en hardware-in-the-loop, se hace impensable que un AUV real comparta el mismo reloj que el servidor del entorno. Esto es posible, pero poco ajustado a funcionamiento real.
- **Sistema asíncrono:** No comparten reloj, por lo que tienen que existir métodos de sincronización.
    - **Ventajas:** Integra sistemas bien diferentes, lo cual se hace especialmente ventajoso en dispositivos remotos, que no tienen por qué tener la misma señal de reloj, como es el caso del simulador que se pretende diseñar.
    - **Desventajas:** Es más complicado de implementar. Requiere diseñar protocolos de sincronización para comunicar procesos implicados.

Asimismo, la aplicación se puede hacer más complicada de controlar.

5.3.1.- Terminología Inicial

Es importante distinguir una serie de términos relacionados con sistemas de simulación distribuidos, y en especial con los simuladores, que van a ser usados a continuación:

- **Ciclo de simulación**: cada uno de los bloques temporales en los que se divide la operación de cada agente en una arquitectura distribuida. Este ciclo suele contener una serie de fases que se repiten periódicamente en cada ciclo.
- **Tiempo de ciclo**: Es la duración real de cada ciclo de la simulación. La elección de este parámetro es crítico, pues en el tiempo de ciclo el simulador debe ser capaz de leer las entradas, ejecutar los cálculos y transmitir las respuestas a los agentes.

Hay dos alternativas:

- **Tiempo de ciclo fijo**: cada instante del simulador dura la misma cantidad de tiempo real. Las ventajas de este tipo de ciclado son:
  - Mayor claridad de funcionamiento.
  - Parámetros se fijan de antemano.
  - El flujo de información está más claramente regulado.

  En cambio las desventajas son:

  - Pérdida de eficacia en los ciclos donde sobra tiempo.
  - El tiempo de ciclo debe ajustarse al proceso más lento de la simulación para que pueda dar tiempo a ejecutarse. Esto empeora el punto anterior.
- **Tiempo de ciclo variable:** El tiempo de ciclo es el mejor que se pueda en cada circunstancia, pudiendo ser un tiempo diferente para distintos componentes del simulador. Cada sistema hace lo que puede dependiendo de sus circunstancias. Las ventajas son:
  - Mayor eficacia de funcionamiento.
  - Flexibilidad para afrontar situaciones complicadas.
  - Mejor adaptación en aplicaciones distribuidas.

  Las desventajas son:

  - Mayor complejidad de la arquitectura y comunicaciones.
  - Necesidad de desarrollar la interpolación.
  - Más compleja la sincronización de procesos.

- **Tiempo de simulación**: dado un determinado instante real, el tiempo simulado es el instante de la simulación que corresponde a ese instante real. Es un instante "virtual".
- **Unidad de tiempo simulado**: es el periodo simulado al que corresponde un tiempo de ciclo. Así, una hora de simulación podría corresponder a 10 horas simuladas, etc.

Puede ser:

- **Unidad de tiempo simulado constante:** cada ciclo de la simulación representa un mismo incremento del tiempo Simulado.
- **Unidad de tiempo simulado variable:** cada ciclo de la simulación dura un tiempo simulado distinto, en función de las necesidades.

- **Interpolación de datos de simulación:** Este término está íntimamente relacionado con el ciclado de la simulación: definido un tiempo de ciclo y su correspondiente unidad de tiempo simulado (fijo o variable), los datos temporales del sistema se actualizan de forma discreta de unidad de tiempo simulado en unidad de tiempo simulado. Por ello puede ser necesario representar el estado del sistema en tiempos simulados intermedios.

  Un ejemplo es cuando a un simulador se le añade una aplicación o módulo de visualización: Si por ejemplo el tiempo de ciclo es variable, se puede necesitar visualizar la simulación en tiempos discretos fijos, siendo más fácilmente interpretable por el usuario. Otro ejemplo es cuando el tiempo de ciclo es fijo, y se desea que el paso temporal del programa de visualización sea distinto al tiempo de ciclo de simulación, para analizar más en detalle la simulación - por ejemplo, como si hiciéramos un zoom.

  La solución es:

  - suponer un modelo lineal muy sencillo basado en la velocidad anterior, o cuadrático basado en la aceleración anterior.
  - Así, haciendo uso de interpolaciones lineales o cuadráticas, se pueden obtener valores de estados intermedios.

- **UTC**: “Coordianted Universal Time”: es un estándar internacional basado en TAI (International Atomic Time) pero corrigiendo ligeramente este tiempo en función del movimiento solar.

  Las señales UTC son enviadas en broadcast desde estaciones de radio y satélites. Tener un receptor UTC es caro, y es por ello que se suelen usar servidores para acceder a este tiempo.

5.3.2.- Casos prácticos: Sincronización software de sistemas distribuidos Cliente/Servidor para Entornos de Simulación.

Para los distintos casos prácticos se parte como base de la referencia **[CAB05]**.

Opción 1: Gestión Temporal Distribuida.

Roles

- **Servidor de Datos**: actualiza todos los datos que dependan del tiempo, y sirve a los distintos clientes los datos solicitados.
- **Clientes**: agentes que solicitan datos al Servidor de Datos.

Descripción

Cada cliente tiene su propio reloj, y el servidor el suyo propio también. Se desprecian los desfases temporales entre el tiempo de los clientes y servidor, desfase de la frecuencia del reloj, y la tardanza temporal en llegar un mensaje de servidor a cliente y viceversa a través de la red. Se trata por tanto de una simulación a "ritmo real".

Es la opción más sencilla de implementar, y es apta para probar hardware real (hardware-in-the-loop), que en el caso de simuladores de AUVS, serían AUVS reales.

El problema es el coste temporal: una hora de simulación es una hora real, por lo que habría que esperar una hora real para poder obtener resultados. Por otra parte, se han obviado desfases entre relojes y desfase debido a tiempo de emitir datos vía red, que en la realidad pueden ser nada despreciables, y contribuir a una descoordinación de los procesos.

Variantes

- *Ajuste de Relojes, mediante Sincronización Software*: la idea es convertir en sistema asíncrono en "síncrono", intentando que todos los relojes se sincronicen con el del servidor, bien en un momento inicial, o periódicamente. Una vez sincronizados, de forma "virtual" es como si todos los elementos del sistema distribuido compartieran la misma señal de reloj.

Si bien todo elemento se sincroniza con el servidor, éste puede estar conectado a una fuente UTC.

Desfase constante de relojes:

Supongamos dos máquinas, tal que sus relojes tengan un desfase constante: la frecuencia de sus relojes sea la misma.

Método propuesto:

El servidor tiene un tiempo inicial t_{j0} que se considera "tiempo de referencia", mientras que un cliente que queremos sincronizar tiene un tiempo inicial t_{i0}, de forma que ambos relojes tienen un desfase inicial.

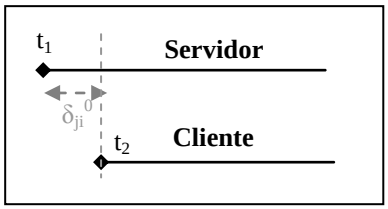


Figura 43: Desfase inicial

$$t_i = t_j + \delta_{ji}$$

Si el cliente conociese el desfase inicial (mostrado en la Figura 43), podría ajustar su reloj por software y así trabajar sincronizado con el servidor. Mediante un handshaking entre cliente y servidor, el cliente podrá calcular el desfase y así corregir el reloj. Los pasos son los siguientes:

0) El cliente envía en el instante ti_0 una solicitud para comenzar Handshaking.

1) El servidor envía un mensaje de sincronización al cliente. El contenido es el timestamp en el servidor (t_{j1}).

2) El mensaje llega al cliente con un desfase temporal total de δ_{ji1} en el instante t_{i1} según el reloj del cliente.

3) Inmediatamente el cliente envía al servidor un mensaje con el timestamp en el cliente (t_{i2})

4) El mensaje llega al servidor con un desfase temporal total de δ_{ji2} en el instante t_{j2} según el reloj del servidor.

5) Inmediatamente el servidor envía al cliente un mensaje con el timestamp t_{j2}, en el instante t_{j3} del servidor.

6) El cliente recibe el mensaje en el instante t_{i3} con el timestamp t_{j2} como información, para que el cliente pueda calcular el desfase objetivo δ_{ji0}.

Gráficamente, este esquema se corresponde con la Figura 44[2]:

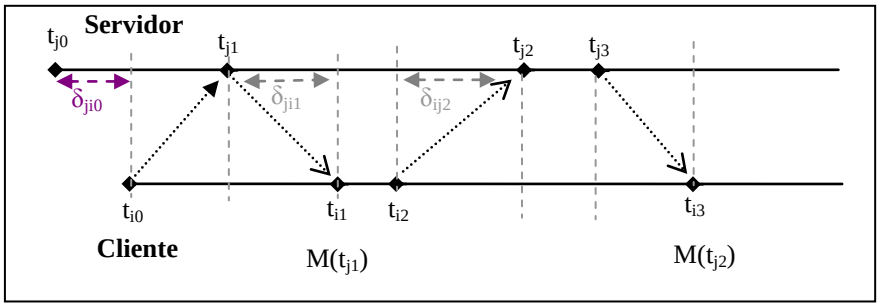


Figura 44: Handshaking para cálculo de desfase inicial.

Acabado el handshaking, el cliente i conoce t_{i1}, t_{j1}, t_{i2}, t_{j2}. δ_{ji1} y δ_{ji2} son el tiempo de demora en llegar mensaje de servidor a cliente y viceversa respectivamente. Se pueden calcular así:

$$\delta_{ji1} = t_{i1} - t_{j1} - \delta_{ji0}$$

$$\delta_{ji2} = t_{j2} - t_{i2} + \delta_{ji0}$$

Dado que el tiempo que tarda el cliente en enviar el mensaje al servidor tras δ_{ji1} es corto según lo ya descrito, puede suponerse que ambas demoras, debidas a retrasos en la red de comunicación entre las dos máquinas, son iguales. Por tanto:

$$t_{i1} - t_{j1} - \delta_{ji0} = t_{j2} - t_{i2} + \delta_{ji0}$$

$$2\,\delta_{ji0} = t_{i1} + t_{i2} - t_{j1} - t_{j2}$$

$$\delta_{ji0} = \frac{t_{i1} + t_{i2} - t_{j1} - t_{j2}}{2}$$

[2] Se marcan con M(s) los mensajes que llegan al cliente, donde s es el contenido del mensaje.

Por tanto, el cliente puede calcular el desfase inicial δ_{ji0}, y con ello ajustar su propio reloj para que esté sincronizado con el del servidor.

Algoritmo de Christian [CRIS89]

Este algoritmo consiste en los siguientes pasos (tal como se muestra en la siguiente Figura 45):

1) El cliente pide la hora al servidor.
2) Tras recibir la petición del cliente, el servidor genera la respuesta con la hora en el servidor *T*.
3) Cuando el cliente recibe la respuesta, pone su propio tiempo a *T* + *RTT/2*.

RTT es el "Round Trip Time", que es el tiempo que desde que el cliente envía el paquete, hasta que lo recibe. El algoritmo asume de forma ideal que el retardo es exactamente el mismo en llegar el paquete de cliente a servidor que viceversa.

Se puede aproximar RTT de diversas formas. La más sencilla es enviar desde el cliente un paquete tipo ping al servidor. Otras aproximaciones consisten en hacer múltiples pings y usar como el RTT el menor (*min*).

Esto es así porque el menor tiempo en el que el servidor pudo fijar el tiempo *T* tras el envío del cliente es *min.* Así, cuando el mensaje de ping del cliente se recibe en el servidor está en el rango (T-min) a (T + RTT-min). La amplitud del rango es (RTT-2*min), lo cual da una exactitud de ±(RTT/2-min).

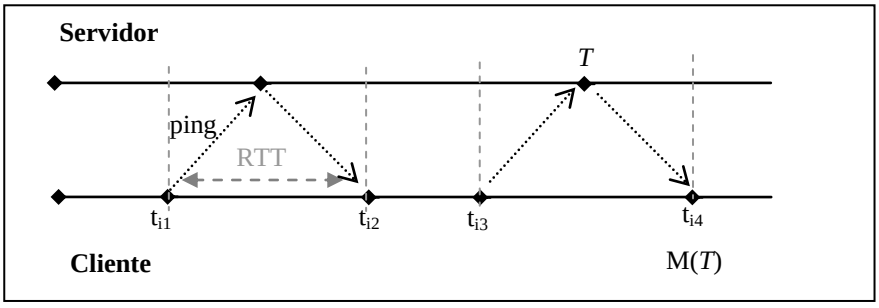


Figura 45: Algoritmo de Christian

$$RTT = t_{i2} - ti_1$$

$$t_{cliente} = T + RTT72.$$

Network Time Protocol (NTP) [AKANTP] y [MIL06]

Protocolo para distribuir el tiempo UTC a clientes que se sincronizan a través de Internet, de forma robusta ante perdidas de conectividad mediante servidores redundantes. Usa el puerto UDP 123 como capa de transporte.

Este protocolo utiliza un modelo jerárquico cliente/servidor para la sincronización. Los distintos servidores se conectan en una jerarquía lógica, tal que los servidores del nivel *n* se sincronizan directamente con los del nivel *n-1*, más precisos en la medida temporal. Así:

- El **estrato 0 (reloj de referencia)** lo forman dispositivos muy precisos, tal como relojes atómicos de cesio o rubidio, reloj GPS, u otros relojes por radio. No se conectan por red, sino localmente a ordenadores mediante RS-232, usando señal de un pulso por segundo.
- El **estrato1 (servidores temporales)** son ordenadores clientes conectados a los equipos del **estrato 0**. Responden a peticiones temporales del **estrado2,** y son los servidores de tiempo con más precisión del protocolo.
- El **estrato 2** son ordenadores, donde cada uno de ellos actúa como cliente de varios servidores del **estrato1**, y que tras obtener la muestra temporal de varios servidores - mediante paquetes NTP - usan el algoritmo del protocolo NTP para obtener la mejor muestra. Incluso se conectan entre sí los ordenadores de este nivel para obtener medidas más robustas.
- El **estrato 3** está constituido por ordenadores con las mismas funciones que los ordenadores del **estrato 2**, solo que toman el tiempo del **estrato 2** en vez del **estrato 1**, con lo que pierden precisión. Por tanto, pueden actuar como servidores de otros estratos inferiores.

Dependiendo de la versión de NTP, acepta diferente número de **estratos**. El básico acepta 16 **estratos**. En la Figura 46 se muestran en rojo las conexiones de red, y en verde la conexión directa.

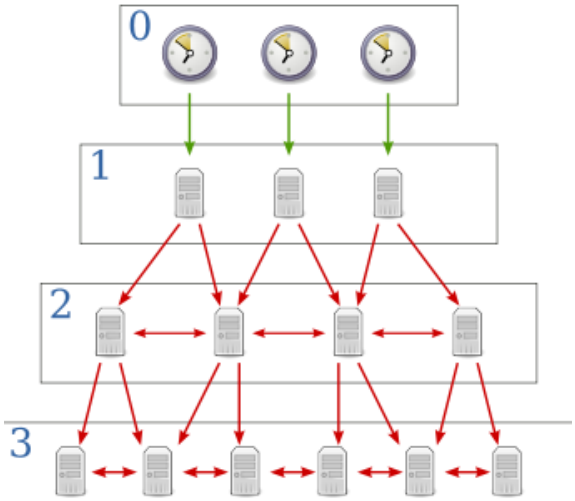


Figura 46: Protocolo NTP.

Esta jerarquía se puede reconfigurar a medida que los servidores fallan o pierden conectividad. Esta redundancia es la que permite mayor robustez en el protocolo.

Por otra parte la sincronización entre equipos de distintos estratos se realiza mediante tres sistemas, de menor a mayor necesidad de exactitud, teniendo en cuenta que entre estratos superiores, hace falta más exactitud:

- **Multicast o redes LAN locales de alta velocidad**: muchos clientes pueden sincronizarse con uno o varios servidores, reduciendo el tráfico cuando muchos clientes están implicados.
- **Modo de llamada a procedimiento**: por ejemplo el algoritmo de Christian.
- **Modo simétrico:** conseguir la máxima exactitud. Se usan algoritmos como el **método propuesto** ya citado anteriormente, donde se trata de calcular el offset entre dos servidores para corregir el que corresponda.

Algoritmo de Berkeley [GUS89]

En esta aproximación se asume que en el sistema distribuido ninguno ni siquiera un servidor tiene el tiempo más exacto, ni siquiera el servidor. Los pasos son:

1) Se elige un equipo maestro por cualquier algoritmo.
2) El maestro interroga al resto de equipos tal como se hace en el algoritmo de Christian, obteniendo de cada equipo el tiempo, y además el RTT usado para calcular el tiempo en cada equipo, según algoritmo de Christian.
3) El maestro hace la media de todos los tiempos en todas las máquinas (incluido el maestro) y calcula el tiempo del sistema *T*.
4) El maestro calcula para cada cliente la diferencia entre *T* y el tiempo que le envió la propia máquina (δ)
5) El maestro envía a cada máquina esa diferencia δ, lo que indica el tiempo que tiene que sumar o restar a su tiempo para sincronizarse con el sistema.

$$t_{cliente\_i} = t_{cliente\_i-1} \pm \delta_i$$

La Figura 47 y Figura 48 muestran el algoritmo y su resultado final.

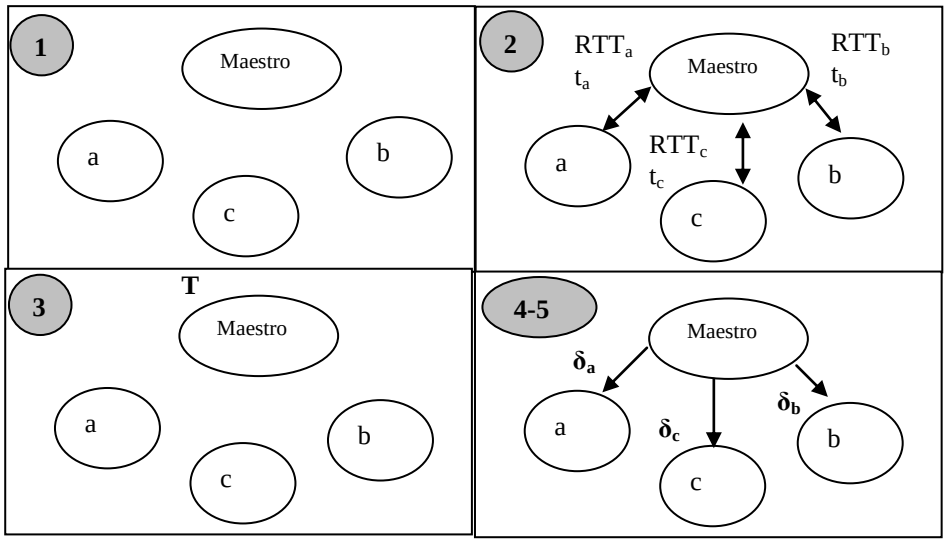


Figura 47: pasos del algoritmo de Berkeley

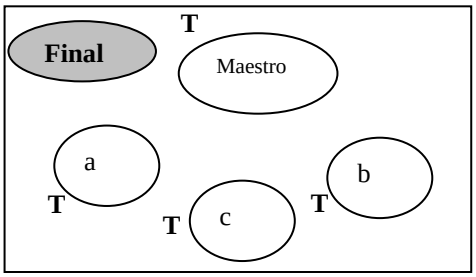


Figura 48: Resultado final: todos sincronizados.

Es importante tener en cuenta que, en el caso de que δ sea negativo, significa que la máquina tiene que ajustar su reloj restando. Esto puede tener llevar a problemas si ya se han realizado acciones. Por ello una solución es, en vez de retroceder, congelar el tiempo tanto como indica δ sin hacer nada ni avanzar tiempo, y así sincronizarse. En este caso no tiene sentido usar UTC.

Desfase en la frecuencia de los relojes

Método propuesto: Ajuste a posteriori

Esta es la aproximación más realista: el problema no solo puede residir en que inicialmente no estén sincronizados, sino que se pueden desincronizar de nuevo si las frecuencias de los relojes es diferente.

Una primera solución sería repetir el proceso anterior periódicamente: cada vez que lo ejecuta tendría que ajustar el reloj. Pero esto conlleva ciertas dificultades:

- La elección del periodo: se tiene que llegar a un periodo de compromiso:
  - Un gran periodo: podría provocar que la desincronización afectara al comportamiento y funcionamiento de clientes y Servidor.
  - Un pequeño período: podría ralentizar demasiado la operación de clientes y Servidor.

- Cuando hablamos de múltiples clientes: si las frecuencias de sus relojes son muy dispares, puede ser un problema elegir un período igual para todos: tendríamos que tomar el periodo que necesite el cliente con una frecuencia de reloj más dispar que la del Servidor, para que su funcionamiento no se vea afectado por el desfase.

La solución más razonable sería un período por cada cliente, ajustado a sus necesidades.

- Si el cliente tiene una frecuencia mayor que el servidor, el ajuste del reloj negativamente en el cliente, pues se ha adelantado al tiempo del servidor, plantea el problema de la interpolación: qué hacer con las acciones ya realizadas y que según el tiempo del servidor no debieron haberse realizado.
- En el caso contrario al anterior, que el cliente tenga una frecuencia menor que la del servidor, el ajuste del reloj a un tiempo futuro puede provocar inanición: acciones que se tenían que realizar no se realizan, debido al salto temporal para equiparar al cliente con el servidor en tiempo.

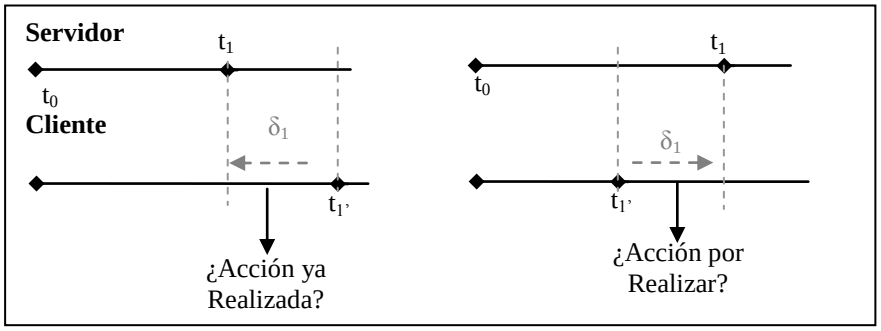


Figura 49: Ajuste a Posteriori

En la Figura 49 podemos observar lo expuesto en los dos puntos anteriores. δ1 en cada caso es la corrección de reloj que tiene que realizar el cliente, en el caso de la izquierda atrasar el reloj y en el de la derecha adelantarlo.

Método propuesto: Ajuste a priori

El mayor inconveniente de la solución anterior es que se hace la corrección a posteriori. Por tanto, la alternativa es hacer la corrección a priori: en resumen, consiste no solo en calcular el desfase inicial, sino también el desfase de las frecuencias de ambos relojes, y usar ambas en el cálculo temporal del cliente. Para ello:

1) Utilizar el ***Método propuesto: desfase constante de relojes*** descrito anteriormente.

2) Con el cálculo del desfase inicial, realizar primer ajuste del reloj en el cliente.

3) Esperar en el cliente un tiempo *p* fijado de antemano.

4) Repetir el método de ajuste propuesto en el punto 1.

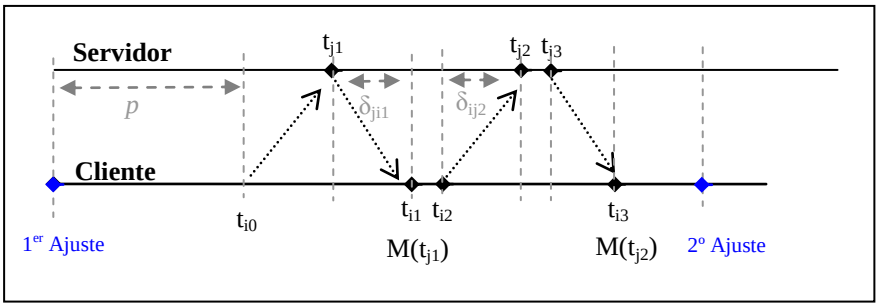


Figura 50: Ajuste a Priori

5) En este caso, obtenemos un desfase δ_{ji0}'. Dado que ya se realizó un primer ajuste, si los relojes de cliente y Servidor tuvieran la misma frecuencia, debería ser 0 o despreciable.

6) En caso contrario: si llamamos γ al tiempo transcurrido desde el primer ajuste hasta que termina el método de handshaking empleado en el punto 4 (t_{i3} en la Figura 50 anterior), tenemos, el resultado de la Figura 51.

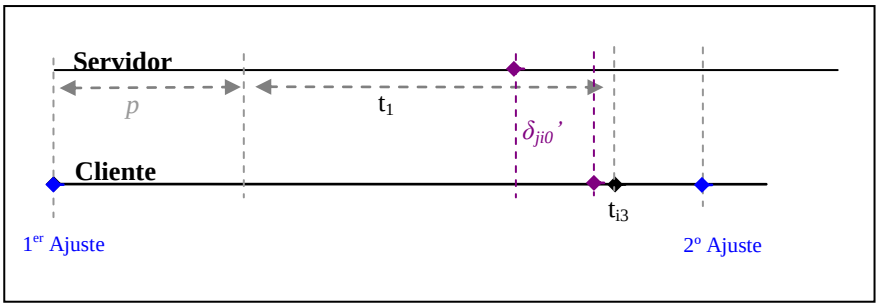


Figura 51: Resultado ajuste a Priori

Por un simple silogismo, si en un tiempo aproximado[3] de γ desde primer ajuste, el desajuste ha sido de δ_{ji0}', en el tiempo de ciclo del cliente (t_{ciclo}) el desajuste será δ_{ij0}'* t_{ciclo} / γ

7) Con esto podemos hacer un *segundo ajuste* del tiempo en el cliente respecto al tiempo en el servidor, en función del desfase δ_{ij0}'.

8) A partir de este segundo ajuste, y para evitar desajustes debido a la diferencia de frecuencias de los relojes de servidor y cliente, se puede utilizar la siguiente fórmula para incrementar el tiempo en el cliente:

$$t_{k+1} = t_k + t_{ciclo} - \frac{\delta_{ij0}' * t_{ciclo}}{\lambda}$$

Hay que considerar dos aspectos:

1) Es importante la elección del período *p*: cuanto mayor sea, con más precisión se ajustará la fórmula de ajuste temporal en el cliente. Depende de la diferencia de frecuencias entre los relojes de cliente y servidor: cuanto mayor sea ésta, menor *p* se necesita para ajustar la fórmula.

2) La fórmula de ajuste temporal en el cliente es una aproximación: esto es así porque durante el tiempo en que se produce el handshaking, la diferencia δ no es constante, sino que varía, mientras que la técnica de handshaking considera que este desfase es constante. Cuanto mayor sea la diferencia de frecuencias de relojes de cliente y servidor, peor calidad tendrá la solución aportada. En la

[3] Aproximado porque γ comprende el tiempo en que se realiza el handshaking en el que se calcula el desajuste δ_{ji0}'. Por ello durante el cálculo del desajuste a su vez se está produciendo un desajuste.

mayoría de casos esta diferencia de frecuencias afecta de forma mínima, ya que tanto clientes como servidor son ordenadores.

Variante:

- *Escalado del tiempo de simulación constante*: Se define un tiempo de ciclo real y una correspondencia en tiempo simulado, de manera que el paso de una unidad de tiempo real equivale en la simulación al paso de la unidad de tiempo de simulado.

  Por ejemplo: si se define que un ciclo equivale a 1 segundo real, y 3 horas simuladas, cada paso de 1 segundo, equivale al paso de 3 horas de simulación.

  Esta aproximación sigue siendo sencilla de implementar, aunque requiere una interpolación de los datos que se generarían durante el tiempo simulado que avanza durante el tiempo de ciclo.

Conclusión

Por todo lo anterior, consideraremos como una aproximación inicial aceptable el que cada elemento realice su propia gestión temporal. Además, no es una aproximación difícil de implementar, y da juego el hecho de hacer un escalado temporal para acelerar el proceso de simulación. Otros esquemas más refinados podrían ser incluidos como objetivos de continuación en proyectos más avanzados.

En cuanto a las variantes, parece muy interesante la sincronización del desfase constante de relojes, que siempre puede suceder dado que dos máquinas remotas pueden tener un desfase temporal muy importante, en cualquiera de las propuestas expuestas.

Sin embargo, las aproximaciones para sincronizar el desfase en la frecuencia de los relojes no parecen tan interesantes, al menos inicialmente en este proyecto, ya que en general el desfase de frecuencias entre máquinas no tiene por qué ser muy importante y se puede despreciar.

Opción 2: Servidor Temporal Central

Roles

- **Servidor Temporal**: Se encarga de coordinar a servidor de datos y clientes. Genera pulsos temporales para marcar inicio y fin de cada ciclo, y además informa del tiempo de simulación en todo momento.
- **Servidor de Datos**: Actualiza los datos que dependan del tiempo y sirve datos a los clientes. El tiempo lo marca el servidor temporal. En muchas ocasiones el servidor de datos y el temporal son la misma entidad.
- **Clientes**: realizan peticiones al servidor de datos, y el tiempo es marcado por el servidor temporal.

Descripción

A diferencia de las aproximaciones anteriores, donde, con o sin sincronización al final, tanto clientes como servidor de datos tenían su propio módulo para medir el tiempo, en esta opción *el **servidor temporal** coordina a todos los clientes, al servidor de datos y a sí mismo*, permitiendo así un tiempo común. Coordinar procesos con gestor temporal externo es más complejo que las alternativas mentadas en la Opción 1

Por otra parte, se ve afectada por los retardos en la red ya que el tiempo ya no es generado localmente, sino a través de la red. Incluso introduce nuevos posibles errores, por ejemplo, si se pierde momentáneamente la comunicación.

Sin embargo, un tiempo centralizado, como veremos más adelante, permite hacer un balance de las necesidades de todos los clientes, y con ello se introduce la posibilidad de un tiempo de ciclo "ajustable" a las necesidades globales. Esto hace que el coste temporal de las simulaciones sea menor, obviando el factor red para transmisión de señales de sincronización entre servidor central y clientes.

Variantes

A continuación se proponen diversas alternativas:

- *Simulación guiada por tiempo*: En este caso el tiempo de ciclo no está establecido de antemano. Pero en cada ciclo la unidad de tiempo simulado es la misma y constante preestablecido en servidor temporal y clientes, o se le comunica al cliente en un handshaking inicial.

  El protocolo sería:

  0) El servidor temporal resuelve el tiempo de simulación del ciclo actual *t*. Si es primera iteración, un valor inicial $t=t_0$, mientras que si no, sumar la unidad de tiempo simulado *T* al tiempo de simulación anterior, $t=t+T$.
  1) Envía un Tic Temporal al servidor de datos, incluido el tiempo de simulación.
  2) El servidor de datos actualiza los datos que dependan del tiempo, calculando los datos para el instante $t + T$, que es el final del ciclo. Para interpolaciones, deberá conservar los datos tanto en el instante *t* como en $t+T$. Tras ello hace un acknowledge al servidor Temporal. Véase que si el servidor de datos está integrado con el temporal no hace falta el acknowledge ni el paso anterior.
  3) El servidor temporal envía un mensaje que marca el inicio del ciclo (Tic Temporal) a los clientes. En este mensaje también se envía el tiempo de simulación correspondiente al inicio del ciclo *t*, que el cliente deberá tomar como fecha de comienzo del ciclo1[4].
  4) Cada cliente, cuando reciba esta "señal", sabe que comienza el ciclo en *t*, y que en ese ciclo debe simularse la unidad de tiempo simulado *T*.
  5) Dos opciones:
     a. **Ejecución en orden**: Cliente realiza ordenadamente paso a paso todas las acciones que debería realizar entre *t* y $t+T$, siguiendo sus planes de misión en el caso de los AUVs, incluido enviar peticiones al servidor de datos.

[4] El cliente puede tomar el tiempo solo 1 vez, y a partir de ahí ir incrementando la unidad de tiempo simulado por si mismo, o actualizar su propio contador temporal cada vez que el servidor le envía un nuevo Tic Temporal acompañado del tiempo de simulación en ese instante.

Al enviar una petición al servidor de datos espera por la respuesta de la misma, ya que no puede saltarse ninguna acción. Esto introduce evidentes retardos en la operación del cliente, que dependen de la congestión tanto del servidor de datos como de la red.Sin embargo es una aproximación más sencilla: el cliente solo tiene que seguir su hilo conductor.

b. **Ejecución fuera de orden**: es una optimización del anterior.

b.1. El cliente realiza una lista de todas las acciones que debe ejecutar en la unidad de tiempo simulado *T*.

b.2. Dentro de esta lista, ordena las acciones en colas según dependencia.

b.2. Todas las que sean independientes las ejecuta (primer elemento de la cola). Puede que tras la ejecución de algunas acciones surjan nuevas acciones, que son añadidas a la lista y a la cola correspondiente.

Respecto a las acciones que requieren peticiones al servidor de datos, todas las que no dependan entre sí ni de peticiones anteriores, las envía en una ráfaga al mismo.

b.3 A medida que se van resolviendo acciones y peticiones del servidor de datos, se van eliminando elementos de las colas, permitiendo que avancen las acciones o peticiones dependientes, y que se puedan ejecutar.

Esta aproximación es bastante más compleja, ya que requiere:

- Una predicción previa de todas las acciones que se tienen que realizar en la unidad de tiempo simulado T.
- Algoritmo para ordenar las acciones y peticiones según dependencia.
- Gestión de las colas de dependencia.

- Gestión de las respuestas u acciones para retroalimentar las colas con las acciones o peticiones que desencadenen.

Sin embargo se evita que las acciones independientes de los clientes esperen innecesariamente de peticiones a servidor de datos o de acciones más pesadas pero planificadas para ejecutarse antes según el plan/es del cliente.

6) El cliente, cuando termina de ejecutar todas las acciones que tiene planificadas durante la unidad de tiempo simulado *T*, envía una señal de *acknowledge* al servidor temporal, para que pueda saber que el cliente terminó de simular durante el ciclo.
7) Aparte, el cliente debe tener una cuenta del instante simulado en que se encuentra (*t*) y avanzarlo tras haber ejecutado el ciclo ($t=t+T$). El tiempo *t* será el que use en todos los timestamps, ya que es la fecha simulada.
8) Cuando el servidor temporal reciba el *acknowledge* de todos los clientes, vuelve al punto 0. Debe existir un tiempo de espera máximo para evitar el bloqueo indefinido del servidor. Al ser excedido por algún cliente, el servidor debe resolver el conflicto:
   a. Bien interrogando el cliente en cuestión y haciendo chequeos de conectividad. Si no persevera el cliente en no responder será excluido y dejará de tomarse en consideración.
   b. Bien directamente excluir a cualquier cliente que haya excedido el tiempo.

Nótese que el hecho de enviar el acknowledge por parte del cliente, y que el servidor temporal espere a todos los acknowledges introduce un retardo en el protocolo, pero es la manera más "fiable" y óptima.

Otra alternativa sería que el servidor temporal espere un tiempo prefijado *X* a que los clientes terminen sus operaciones, pero ajustar esta *X* es muy complejo: demasiada pequeña hace que los clientes no tengan tiempo de ejecutar sus acciones en el ciclo, y demasiado grande hace que el tiempo de ciclo se prolongue innecesariamente.

Respecto al punto 6, puede existir un problema: un cliente envía un acknowledge, pero acto seguido, y antes de que acabe el ciclo recibe una "acción inesperada". Si el agente simula un robot comandado, la llegada de un comando, o por ejemplo en el caso de AUVs, una situación de emergencia. Esto se puede resolver:

- O bien cualquier eventualidad tiene que esperar a la llegada del nuevo Tic Temporal del Servidor.
- O bien se da a los clientes la posibilidad de anular un acknowledge realizado. En esta opción el cliente tiene que mandar la anulación, y el servidor devolverle un mensaje como que acepta la anulación, y así ya tratar la eventualidad.

Para ambas opciones es importante que el envío del acknowledge no se realice inmediatamente tras realizar todos los eventos de la cola, sino dar un retardo prudencial para dar tiempo a la cabida de eventualidades. Durante el ciclo.

En general, la desventaja de éste método es que requiere interpolación en el servidor de datos para obtener datos que dependan del paso temporal entre los periodos t y $t+T$.

Por ejemplo, si nos encontramos ante un servidor de datos que simula las corrientes marinas en cada instante, éste solo puede calcular las corrientes en instantes discretos (en t, y después en $t+T$). De forma que si un cliente pide un dato en el instante α, entre t y $t+T$, el servidor de datos tendrá que interpolarlo.

Sólo si las peticiones se enviaran ordenadas temporalmente, el servidor de datos podría avanzar el tiempo en vez de de t a $t+T$, de t a α, y luego de α a $t+T$, y así no haría falta interpolar. Pero no es el caso que nos ocupa, pues aunque los clientes enviaran peticiones ordenadamente, no hay orden entre clientes, por lo que, como muestra la Figura 52, un cliente *C1* puede primero pedir un dato para el instante *t2*, y luego el cliente *C2* pedir un dato para el instante anterior *t1*, porque el cliente C2 necesita resolver datos antes del instante *t2*.

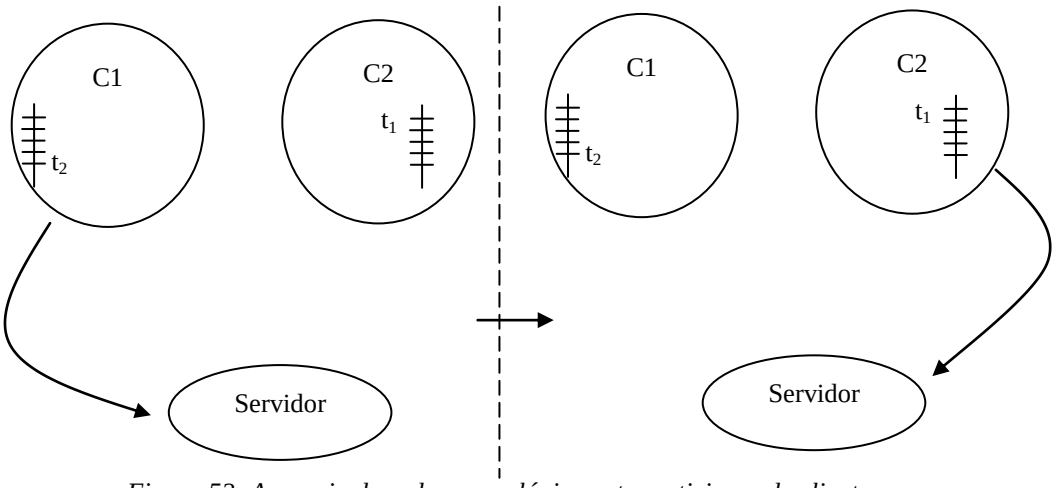


Figura 52: Ausencia de orden cronológico entre peticiones de clientes.

Una ventaja de esta aproximación es que se hace más sencillo detener la simulación: basta con que el servidor deje de enviar mensajes de Tic Temporal.

Por otra parte, a la hora de hacer una simulación gráfica también se simplifica: con cada nuevo Tic Temporal se recalculan datos en servidor de datos y se refresca la vista de la simulación, con lo que se consigue que el incremento del tiempo de simulación sea constante. Incluso es fácil de configurar otras escalas de tiempo que sean múltiplo de la unidad de tiempo simulado: en vez de cada Tic, cada *n* Tics.

- *Simulación guiada por eventos*: En este caso el tiempo de ciclo no está establecido de antemano, ni el tiempo de simulación. La unidad de tiempo simulado en cada ciclo lo marca el siguiente evento a ejecutarse. El proceso es el siguiente:

  0) El servidor temporal resuelve el tiempo de simulación del ciclo actual *t*. Si es primera iteración, un valor inicial $t=t_0$, si no, sumar la unidad de tiempo simulado *T* que se calculó en el bucle anterior, al tiempo de simulación anterior, $t=t+T$.

  1) El servidor temporal envía un mensaje que marca el inicio del ciclo (Tic Temporal) acompañado del tiempo de simulación según el servidor temporal.

2) Cada cliente, toma este tiempo de simulación como propio [5] y como respuesta al mismo, envía al servidor temporal el tiempo del próximo evento que tiene que simular.

3) Cuando el servidor haya recibido el tiempo del próximo evento de todos y cada uno de los clientes, calcula como unidad de tiempo simulado la diferencia entre el tiempo actual *t* y el tiempo del evento que se tiene que ejecutar antes de todos los clientes (*t*+*T*), y por tanto calcula *T*.

   Debe existir un tiempo de espera máximo para evitar el bloqueo indefinido del servidor. Al ser excedido por algún cliente, haga que el servidor debe resolver el conflicto de las formas especificadas en el punto 8 del protocolo anterior.

4) Envía un Tic Temporal al servidor de datos, incluido el tiempo de simulación.

5) Con esta unidad de tiempo simulada, ya puede el servidor de datos calcular los datos que dependan del tiempo en el instante *t* + *T* al final del ciclo, aunque para la interpolación deberá conservar los datos también en el instante *t*. Véase que si el servidor de datos está integrado con el temporal no hace falta el acknowledge ni el paso anterior.

6) Envía el servidor temporal la unidad de tiempo simulado a cada cliente. Ya todo cliente conoce tanto el comienzo del ciclo, como cuanto dura el mismo.

7) Ejecutar el protocolo de *Simulación guiada por tiempo* en cada cliente, desde el paso 5 al 8, y tras el 8, volver al paso 0 del protocolo que aquí se explica.

Las ventajas este método es que evita ciclos innecesarios, ya que en cada ciclo siempre se va a realizar alguna acción por parte de algún cliente. Además, se reducen notablemente el número de interpolaciones de datos dentro de un ciclo, ya que el ciclo está ajustado para un evento. Sólo harían falta interpolaciones en el caso de que alguno de los clientes de forma imprevista tenga que realizar un evento, bien porque es comandado o por una situación de emergencia.

[5] El cliente puede tomar el tiempo solo una vez, y a partir de ahí ir incrementando la unidad de tiempo simulado por si mismo, o actualizar su propio contador temporal cada vez que el servidor le envía un nuevo Tic Temporal acompañado del tiempo de simulación en ese instante.

El primer problema es que puede ser ineficiente, ya que "en el peor caso", en cada ciclo se ejecuta un único evento de un solo cliente, por lo que hay que repetir para cada evento todo el protocolo. Y "en el peor caso" porque puede darse eventos imprevistos por parte de algún cliente.

De igual forma, la simulación gráfica es otro problema, ya que si se quiere que la unidad de tiempo simulado sea constante - fijar un paso temporal de simulación - habría a su vez que hacer interpolaciones, o lo que es lo mismo, que el simulador gráfico sea otro cliente que pida todos los datos en los periodos señalados.

Conclusión

En general esta alternativa es más compleja que la primera expuesta, si bien más robusta, pues controla que en cada ciclo todos los subsistemas de la simulación terminen sus tareas a realizar en el ciclo correspondiente, con lo que ninguno se queda atrás.

Por otra parte, también es mejor en términos de coste temporal que la opción 1, puesto que da pie a que cada ciclo dure justo el tiempo que tarden los subsistemas en terminar sus tareas. Terminadas todas, el gestor temporal puede avanzar el ciclo *aunque no haya terminado el tiempo de ciclo*. En la opción 1, como no hay retroalimentación de los clientes, el ciclo siempre tiene que durar slices constantes. Por todo esto, esta opción tiene gran valor para un proyecto de simulación como el que se expone.

CAPÍTULO 6-. *Batimetría y Formatos para Representar Volúmenes de Datos*

El capítulo aborda en profundidad la batimetría, definiéndola y definiendo su proceso de obtención, almacenamiento y representación. Acto seguido, se citan las bases de datos batimétricas actuales más relevantes, para finalmente presentar los formatos para almacenamiento de datos batimétricos y otros volúmenes de datos en ficheros.

6.1.- Definición de Batimetría

La batimetría se define como la profundidad bajo el nivel del mar. Topografía es la elevación por encima del nivel del mar. La definición del relieve requiere la especificación de la batimetría y de la topografía. El término incluye también el conjunto de técnicas asociadas a la obtención de estas profundidades, tanto a nivel marino, fluvial (relativo a ríos) como lacustre (relativo a lagos). La representación de la misma en mapas puede formar perfectamente una Carta Náutica, usada con motivos de navegación y seguridad.

Principalmente se obtiene mediante **ecosondas**, un sónar que emite un tren de pulsos ultrasónicos y transforma en distancia el tiempo que se tarda en recibir el eco de la señal tras chocar con el fondo marino.

Existen principalmente dos maneras de medir esta batimetría, acorde a **[SHOABAT]**:

- **Monohaz**: la energía acústica está confinada a un solo haz. Dada las dimensiones del mar, este angosto haz se muestra insuficiente, costoso y a menudo incorrecto. Era la tecnología habitual a principios de los setenta.
- **MultiHaz:** se emiten varios haces angostos de sonido ordenados en modo abanico, que barren el fondo marino de forma simultánea. Minimiza costos y cubre mayores áreas con exactitud y precisión. También poseen elementos correctores que revisan la calidad del dato obtenido. Se puede ver un ejemplo en la Figura 53.

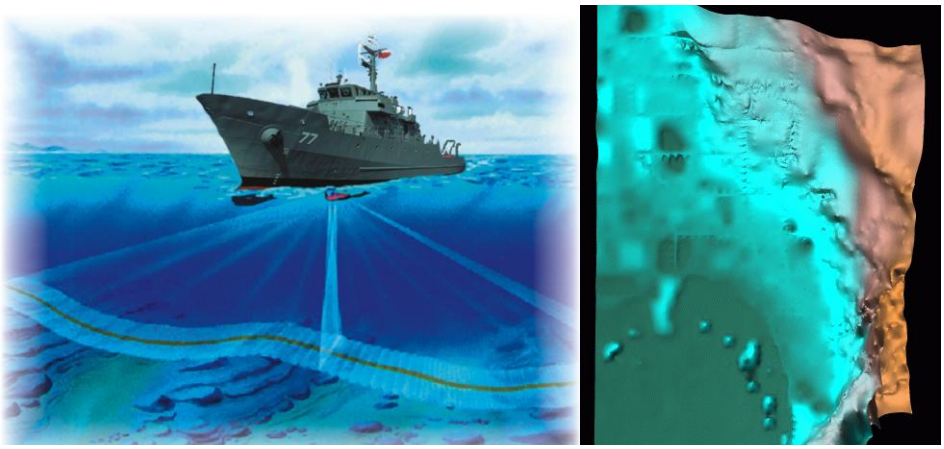


Figura 53: Izquierda, ejemplo de ecosonda multihaz. Derecha, ejemplo de batimetría. Estrecho de Yucatán, de 2'.

6.2.- Bases de Datos Batimétricas Actuales

Existen varias instituciones que ofrecen datos batimétricos. Estos repositorios son interesantes para conocer los diferentes formatos en los que especifican la batimetría y - de paso - descargar algunas batimetrías de ejemplos. Se ha seguido la referencia **[VTERRAIN]** como referencia para este Apartado.

NOS Data Explorer: Nacional Ocean Service [CCMA]

Es parte de NOAA (*National Oceanic and Atmospheric Administration*), siendo un portal que contiene enlaces a datos de Estados Unidos sobre batimetría. Para describir la batimetría utilizan dos elementos:

- Una descripción que explica la zona sobre la que se ha hecho la batimetría y forma en la que se ha hecho, en forma de metadatos. Para esto se utiliza el formato FGDC.
- Ficheros de batimetría descargables. Existen diversos formatos, como el DEM (digital elevation model, bastante usado en general), ASCII, ESRI grids, etc.

Podemos ver la batimetría correspondiente a:

NOAA ESRI Grid- 1m Bathymetry of St. John (South Shore - Area 3), US Virgin Islands, 2004, UTM 20 WGS84

Se divide en 4 ficheros ASCII, debido a su magnitud. Solo descargamos la primera parte. En los ficheros se delimitan por comas la longitud, latitud y profundidad, en metros. A continuación, en la Figura 54 se muestra un ejemplo en metros

302967.720	,	2014558.280	,	44.471
302968.720	,	2014558.280	,	44.475
302969.720	,	2014558.280	,	44.522
302965.720	,	2014559.280	,	44.412
…….				

Figura 54: batimetría en ASCII.

NGDC: National Geophysical Data Center [NGDC] y [NGDC2]

Con datos recopilados se creó una rejilla mundial de resolución 2'x2' por Sandwell y Smith, con más o menos 1,6 km^2 de resolución. El método utilizado usa predicciones e interpolaciones de datos batimétricos, por lo que puede haber errores y en consecuencia no deben ser usados para la navegación. Se ha primado el hecho de crear una digitalización de la batimetría mundial marina, frente a la resolución.

Aunque en NGDC se aceptan datos de diversa índole, hay preferencia por el formato MGD77, formato general para datos geofísicos.

En la página se encuentran diversas herramientas desarrolladas para el manejo de bases de datos relativas a datos geofísicos, entre ellos, batimetría. Destacamos las herramientas más directamente relacionadas con batimetrías.

- Batimetría MultiBeam:

  Está en un formato que puede ser tratado y visualizado con MB System, software libre. Este sistema se usa para el proceso y renderización de datos batimétricos MultiHaz. Se prevé que acepte ficheros en formato SURF. Acepta también ficheros en formato GMT grid y GMT-GRD grid principalmente, además de conversiones a ASCII y otros formatos.

- NGDC modelo de desarrollo para relieve de costas: no habla del formato en si de la batimetría, pero sugiere un proceso de transformaciones para integrar datos de diversas bases de datos, todas con resolución de 3 segundos de arco (error de unos 90 metros). Toman diversas informaciones de distintas bases de datos (5) (DEMs grids, USGS ASCII format, etc.) y los funden en una representación única en un formato propio.

El formato NGDC interno de representación de datos, tras un proceso de refinado, se articula en base al siguiente formato:

- Cabecera de 128 Bytes. Contiene 32 elementos de 4 bytes cada uno con descriptores de la malla, que se muestran en la Figura 55:

0..3 nº versión	4..7 tamaño Cabecera	8..11 Tipo de Datos(malla de elevación, malla de densidades, etc)	12..15 Grados Mayor Latitud al Norte	16..19 Minutos Mayor Latitud Norte.
20..23 Segundos Mayor Latitud Norte	24..27 Dimensión de Latitud de una celda de la malla (arc-seg)	28..31 Nº de Celdas por Fila de la Malla	32..35 Grados Mayor Longitud al Oeste	36..39 Minutos Mayor Longitud al Oeste
40..43 Segundos Mayor Longitud al Oeste	44..47 Dimensión de la Longitud de una celta de la malla (arc-seg)	48..51 Nº de Celdas por Columna de la Malla	52..55 Mínima Elevación de la Malla	56..59 Máxima Elevación de la Malla.
60..63 Radio de la Malla	64..67 Precisión de la Elevación (metro o decenas de metros)	68..71 Valor de la celda de la malla vacía (p.e.: NaN)	72..75 Tipo de valor (cómo se almacenan datos. P.e. Punto flotante.	76..79 Datos del Mar (nivel del mar o dato local)
80..83 Límite de datos	84..87 Sin Usar	88..91 Sin Usar	92..95 Sin Usar	96..99 Sin Usar
100..103 Sin Usar	104..107 Sin Usar	108..111 Sin Usar	112..115 Sin Usar	116..119 Sin Usar
120..123 Sin Usar	124..127 Sin Usar			

Figura 55: Cabecera Formato NGDC.

- Posteriormente, valores del grid o la malla almacenados en filas que deben ser interpretadas de izquierda a derecha y de arriba abajo. El inicio de la malla es la esquina superior izquierda.

- GEODAS: GEOphysical Data system:

  Es un programa para el manejo de bases de datos generado por NGDC para recompilar datos geofísicos, entre ellos batimetría.

- NOS HDB: Nacional ocean Service Hydrographic Data Base:

  Programa para mantenimiento y manejo de datos de las aguas costeras y Exclusive Economic Zone de los estados unidos.

- GSHHS: A Global Self-consistent, Hierarchical, High-resolution Shoreline Data base:

  Base de datos sobre costas. Usa un formato de datos específico.

- Información batimétrica de lagos de EEUU.

- ETOPO2, ETOPO5 , ETOPOx

  Se combina topografía y batimetría de resolución 2, 5, x minutos, en estas bases de datos. Desarrollado por Sandwell y Smith. Esta macrobase de datos toma como fuentes otras bases de datos.

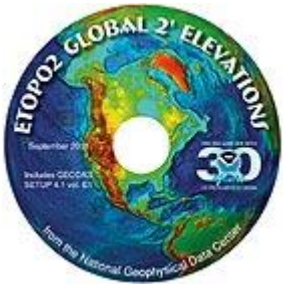


Figura 56 CD-ROM de ETOPO2.

Se encuentran algunas conversiones de ficheros ETOPO2 a otros formatos como es el NETCDF.

CleanTOPO2 es una donde se han eliminado los artefactos que se encuentran en ETOPO2.

GEBCO: General Bathymetric Chart of the Oceans [BODC]

Poseen un atlas mundial de batimetría (en versión CDROM), en plan gráficas y formato digital. Es el GEBCO Digital Atlas (GDA). Este CDROM es de pago. Comparado con ETOPO2, tiene mucho menos ruido y mucha mayor precisión, como se puede comparar en la Figura 57 y Figura 58. Aquí está esta comparación.

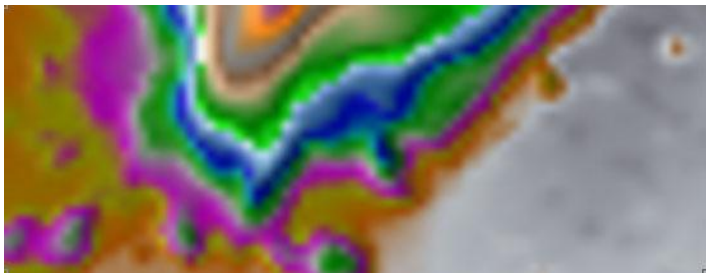


Figura 57: ETOPO2, ruidoso, pero libre y base de datos global.

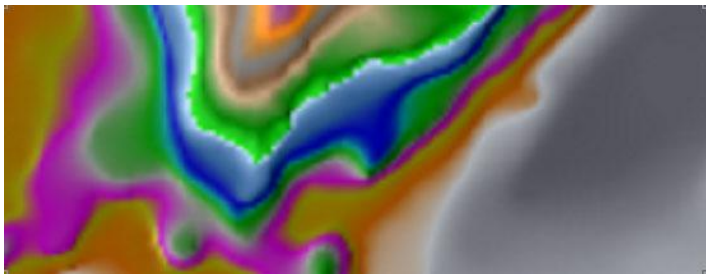


Figura 58: GEBCO, más preciso, pero caro y no libre.

Posee batimetría de 1'x1' minuto de resolución (unos 800 metros). Se da un fichero de entrada, en términos de **Longitud, latitud, Profundidad**. El formato de los ficheros es **netCDF** compatible con GMT (extensión grd), que aseguran la compatibilidad con el paquete software del **GMT**. Para descargar ficheros ejemplos y ver más información ver la referencia **[BODC]**.

Ocean Floor Databases (de la Universidad de Columbia) [OCEAN]

Se usa el formato CDF y HDF. Posee batimetría de tipo "MultiBeam" de la Antártida. Un ejemplo en formato ASCII (*.dat), se puede comprobar en la Figura 59.

```
#timestamp(UTC) longitude(deg) latitude(deg) gravity free-air
anomaly(mG) magnetic IGRF anomaly(nT) corrected depth(m)
1986-06-20 19:03:00   79.710000   6.781660  27.6  -152   NaN
1986-06-20 19:04:00   79.708800   6.779770  27.5  -153   -51
1986-06-20 19:05:00   79.707700   6.777930  28.2  -154   -56
1986-06-20 19:06:00   79.706500   6.776100  30.1  -151   -54
1986-06-20 19:07:00   79.705300   6.774260  27.6  -150   NaN
```

Figura 59: Ejemplo batimetría multbeam del Ocean Floor Databases

Se dan ejemplos de aplicaciones en Java para visualización de elevaciones topográficas y batimétricas. Se puede descartar.

IHO DCDB: Data Center for Digital Bathymetry

En general puede aceptar datos de muy diversa índole, aunque preferiblemente formateados en el formato MGD77.

6.3.- Formatos de ficheros para datos batimétricos

Analizaremos las distintas alternativas que hay en el mundo real para almacenar las medidas batimétricas, y en general para el almacenamiento de magnitudes en una malla real. Se atenderán características como:

- Flexibilidad.
- Compacidad.
- Rapidez en el acceso de rangos de datos en cualquier punto de la malla.

- Lo estándar que puede ser el formato, para permitir cargas batimétricas desde cualquier instituto oceanográfico.

NetCDF: network Common Data Form [NETCDF] y [GFDL]

Introducción.

Consiste en un conjunto de funciones / interfaces simples para almacenamiento y lectura de datos en forma vectorial, de libre distribución y con librerías de acceso para C, Fortran, C++, Java, Perl y otros lenguajes. Es adecuado, en general, para cualquier propósito que implique vectores, ya que sus funciones están optimizadas para acceso vectorial. Hay cinco características que resumen los datos de NETCDF.

- **Autodescripción:** el mismo fichero netCDF contiene información que describen los datos que contiene.
- **Portable:** Un fichero netCDF puede ser accedido por computadoras de diversas arquitecturas, con diferentes formas de almacenar enteros, caracteres, números en punto flotante, etc. Esto es así porque estas librerías netCDF soportan un formato independiente de la máquina para representar datos científicos, ya que NetCDF usa una forma extendida del formato XDR para la representación de la información presente en la cabecera y parte de datos de los ficheros.

  XDR es un estándar para codificación de datos y una librería de funciones para representación externa de datos, lo que permite al programador codificar sus estructuras de datos de forma independiente a la máquina, tal como se describió en el **Apartado 4.2.1**. Puede verse como un tipo abstracto de datos, donde la única manera de acceder a los datos contenidos en un fichero netCDF es mediante las funciones que este software proporciona, lo que nos permite trabajar con datos sin detalles de almacenado. Puede ser cambiable esta representación interna sin afectar a programas que ya haya usado este formato.
- **Acceso directo:** es un formato orientado a vectores, por tanto suministra funciones que permite el acceso directo a secciones de vector que se deseen, sin necesidad de un recorrido secuencial. Además, permite el acceso a vectores enteros o trozos de

los mismos como una unidad y de manera directa, característica que una base de datos no nos permitiría, pudiendo solo acceder a elementos individuales.

Cualquier base de datos de propósito general sería poco eficiente en el acceso a datos vectoriales, y suministraría facilidades no aprovechables en datos vectoriales. No sería adecuada.

- **Expandible:** datos pueden ser añadidos a un fichero netCDF sin copiar todo el conjunto de datos ni redefinir la estructura del mismo.
- **Compartible:** un escritor y múltiples lectores pueden acceder al mismo fichero simultáneamente.
- **Compatible**: se garantiza que cualquier versión futura de este software podrá acceder a ficheros netCDF desarrollados con distintas versiones.

Todo fichero netCDF es descrito por un fichero en formato **CDL** (Common Data form Language), que es un formato ASCII de fácil lectura que contiene los metadatos que describen a un fichero netCDF. A través del CDL se puede genera el fichero binario en netCDF, o viceversa, generando el CDL a partir del netCDF.

Para más información sobre el modelo de datos, ver el apartado correspondiente en el documento anexo al proyecto **Descripción Técnica de Formatos de Datos Utilizados (ANEXO IV).**

HDF

Menos usado que netCDF, el HDF es un formato estructurado de múltiples objetos diseñado en el National Center for SuperComputerAplications (NCSA) para facilitar la transferencia de datos entre máquinas diferentes. Es de libre distribución, muy bueno para ficheros que necesiten ser autodescribibles.

Soporta interfaces con C y FORTRAN, y es portable a distintas arquitecturas de computadores y sistemas operativos. Se trata de un formato de datos común sobre el cual muchas interfaces pueden aplicarse.

Existen 2 versiones, no compatibles entre sí, que son HDF versión 4 y 5. HDF4 ya es un formato basado en relaciones jerárquicas entre datos y dependencias entre los mismos.

Fue desarrollado de forma independiente a netCDF, aunque existen interfaces entre ambos. NCSA implementó un software que suministra una interfaz netCDF al HDF versión 4. Con este software se pueden usar las llamadas de la interfaz netCDF para acceder a datos de un fichero HDF4.

El HDF5 suministra un modelo de datos mucho más rico, con jerarquización, énfasis en la eficiencia del acceso, entrada salida paralela y soporte para computación de alto rendimiento. De hecho, netCDF-4 está implementando una interfaz de netCDF a HDF5 para aprovechar la potencia de cada uno: el uso masivo y simple de netCDF por la comunidad científica, y la generalidad y rendimientos de HDF5.

Una representación batimétrica consistiría en tres vectores:

1) Las profundidades para cada nodo de la malla, que son números en punto flotante de 32-bit IEEE.
2) Una imagen de la malla de tipo carácter sin signo de 8-bits.
3) Una paleta de colores.

CDF

El CDF (Common Data Format) es un formato abstracto para matrices multidimensionales autodescriptivas. Fue diseñado en el NASA/Goddar Space Flight Center. De hecho, fue el primero formato de estas características en surgir. Introdujo la idea de la abstracción de datos para almacenamiento, manipulación, acceso de datos multidimensionales e independencia de la arquitectura de la máquina.

NetCDF sería desarrollado años más tarde basándose en el modelo conceptual de CDF, pero añadiendo otras características, como Interfaces con C (inicialmente CDF solo ofrecía una interfaz en FORTRAN), formato independiente de la máquina, etc.

Así, han ido evolucionando a la par, de forma que más tarde CDF se apropió de algunas de las características de netCDF (acceso de datos, representación XDF, representación en simple fichero, atributos específicos para variables). Aún así, y hoy por hoy, ambos formatos son incompatibles, y de hecho no existe una interfaz entre ambos, cosa que sí se ha logrado entre netCDF y HDF. Les diferencian puntos como que netCDF no soporta representación en modo nativo, y CDF no soporta dimensiones con nombre. Las diferencias entre ambas interfaces son importantes, siendo más inteligible la interfaz de netCDF.

SURFER [GEODES]

Es un programa de pago para la visualización de datos batimétricos. Los ficheros están en formato GRD, y transportables a formato netCDF. Un ejemplo se puede comprobar en la Figura 60.

```
DSAA
376
253
-185556.375     194194.225
-127261.523     128262.152
81.000          4469.000
578 746 1097 1251 581 312 229 229 235 278 292 …
```

Figura 60: Ejemplo de fichero batimétrico en SURFER.

Usa un formato binario basado en etiquetas para permitir compatibilidades con el futuro. Cada sección es precedida por un tag o marca, que indica el tipo y tamaño de los datos incluidos en esa sección. Esto permite acceder a la sección del fichero que se desea. Las secciones pueden venir en cualquier orden, excepto la sección cabecera, que siempre viene al principio.

Para una descripción más detallada sobre este formato de datos, ver el apartado correspondiente en el documento anexo al proyecto **Descripción Técnica de Formatos de Datos Utilizados.**

6.4.- Formatos de propósito general para datos científicos.

GMT [GMT]

"Generic Mapping Tools". Es una colección de software en código abierto para la manipulación de datos Cartesianos del mundo, entre ellos grillas de representaciones batimétricas. Tiene la habilidad de leer grids batimétricos en distintos formatos de ficheros, aunque el formato usado por defecto es GRD. Dado que GMT usa para ejecutarse la librería netCDF, este formato GRD es fácilmente convertible al formato netCDF base (*.nc).

Se muestran algunos formatos existentes en las Tabla 4, Tabla 5, Tabla 6 y Tabla 7.

ID	*GMT 4 netCDF Standard Formats*
nb	GMT netCDF format (byte) (COARDS-compliant)
ns	GMT netCDF format (short) (COARDS-compliant)
ni	GMT netCDF format (int) (COARDS-compliant)
nf	GMT netCDF format (float) (COARDS-compliant)
nd	GMT netCDF format (double) (COARDS-compliant)

Tabla 4: Formatos GMT.

D	*GMT 3 netCDF legacy formats*
cb	GMT netCDF format (byte) (depreciated)
cs	GMT netCDF format (short) (depreciated)
ci	GMT netCDF format (int) (depreciated)
cf	GMT netCDF format (float) (depreciated)
cd	GMT netCDF format (double) (depreciated)

Tabla 5: Formatos GMT.

ID	*GMT native binary formats*
bm	GMT native, C-binary format (bit-mask)
bb	GMT native, C-binary format (byte)
bs	GMT native, C-binary format (short)
bi	GMT native, C-binary format (int)
bf	GMT native, C-binary format (float)
bd	GMT native, C-binary format (double)

Tabla 6: Formatos GMT.

ID	*Miscellaneous gridformats*
rb	SUN rasterfile format (8-bit Standard)
rf	GEODAS grid format GRD98 (NGDC)
sf	Golden Software Surfer format 6 (float)
sd	Golden Software Surfer format 7 (double)
af	Atlantic Geoscience Center AGC (float)

Tabla 7: Formatos GMT.

DEM

"Digital Elevation Model (DEM)": datos en forma de malla, donde el valor es la elevación, bien sea topográfica o batimétrica. Existen diversos formatos. Pero el más usado es el formato USGS (US Geological Survey).

Formato del ESRI (Environmental System Research Institute)

Suele ser usado para exportar ficheros a un formato de mallado ASCII compatible con ArcInfo. Forma parte de los DEM. La sintaxis es la mostrada en la Figura 61:

```
ncols [number of columns]
nrows [number of rows]
xllcorner [x coordinate of lower left corner]
yllcorner [y coordinate of lower left corner]
cellsize  [cell size in meters]
nodata_value [value which will be used if no data in grid cell]
row 1
row 2
```

Figura 61: Sintaxis del formato ESRI.

No va a ser objeto de estudio, pues es un formato muy específico de este instituto, y no se dispone de información detallada de este formato.

MGD77 [NGDC]

Es un formato digital de intercambio para datos geofísicos marinos (batimetría, magnetismo y gravedad). Su objetivo es el uso para transmisión de datos hacia y desde un centro de datos. Fue generado por NGDC. Se trata de un formato de propósito general y, por tanto, no consideraremos este formato de momento en nuestro simulador, ya que ni es el más usado, ni es óptimo ya que, al no centrarse en datos vectoriales, el acceso a los mismos sería muy ineficiente.

Conclusiones

De forma global, NETCDF es el formato más adecuado, ya que es un estándar que cumple con creces los objetivos a cubrir por un formato para almacenamiento de datos que nos marcamos al inicio del apartado. Además, es - de los formatos presentados - el más extendido y documentado, con gran cantidad de librerías en distintos lenguajes de programación para su interpretación.

CAPÍTULO 7.- *Modelos de Interés en el Entorno de los AUVs*

Para finalizar con la sección analítica del proyecto, se hace un estudio de los distintos parámetros fisicoquímicos en el ámbito submarino que puede ser interesante simular, así como las medidas que, a día de hoy, son más relevantes para los AUVs durante el desarrollo de sus misiones y los sensores utilizados para ellos.

También se estudian los dispositivos para comunicación que utilizan los AUVs actuales, y la hidrodinámica de movimiento del AUV. De todos estos elementos, se enfoca el análisis a estudiar modelos matemáticos que permitan su incorporación al simulador.

7.1.- Medidas de Interés en un AUV

A continuación, vamos a hacer un breve análisis de las magnitudes cuya medición por parte de AUV puede resultar de interés para asegurar el éxito de la misión. Estas magnitudes pueden clasificarse en dos tipos diferenciados:

- Medidas explícitas: las requiere directamente la misión por ser de interés científico.
    - Relieve de fondo marino.
    - Temperatura
    - Presión
    - Profundidad
    - Salinidad
    - Densidad
    - Conductividad
    - Y otras medidas como concentración de oxígeno disuelto, clorofila, etc.
- Medidas implícitas: son necesarias para la actividad del AUV. No suelen ser de interés científico, pero si de interés para la operación del vehículo.
    - Actitud (pose en 6D)

- Velocidad

- Medidas Híbridas: tanto de interés científico como operativo del AUV.
  - Distancia
  - Posición.

Relieve de fondo Marino

Es una aplicación de la medida de la distancia. Permite realizar mapas del fondo marino para la generación de ficheros batimétricos, como el que se muestra en la Figura 62. Suele medirse en metros de profundidad de cada punto (x,y,z).

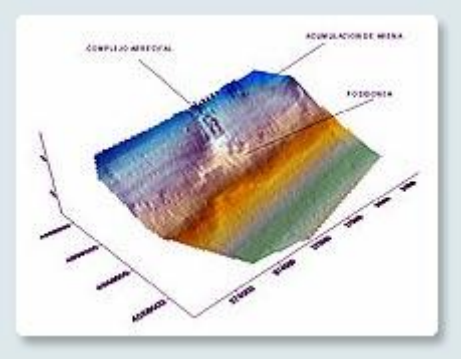


Figura 62: Ejemplo de mapa de fondo marino tomado.

Temperatura

Es un parámetro de interés fundamental en oceanografía. Suele medirse en ºC. En cuanto a este parámetro, existe el IPTS-68 (International Practical Temperature Scale), una escala donde desde hace 22 años se reporta la temperatura marina, con el fin de servir de referencia y calibración. De hecho se suele utilizar el IPTS en las distintas fórmulas y algoritmos.

Existe otra aproximación que es la ITS-90, que introduce el punto de fundición del galio (29.7646 ºC) como punto de referencia de la nueva escala. La conversión entre ambas escalas es lineal:

$$t_{68} = 1.00024 * t_{90}$$

Presión / Profundidad [NI1991]

Se usa la presión por lo general para estimar indirectamente la profundidad. La presión suele medirse en decibars (db), mientras que la profundidad se mide en metros. Como regla empírica, puede considerarse que por cada 10 metros de profundidad el aumento de la presión es de 1 atmósfera, añadiendo la presión atmosférica fuera del agua.

Otra manera un poco más precisa de deducir la presión a través de la profundidad o despejar la variable *h* de la ecuación es considerar que la Presión en el fluido (P_f) es igual a:

$$\mathbf{P_f = \rho * g * h}$$

Donde:

ρ: densidad del fluido, en este caso, el agua del mar.

g: la gravedad

h: la altura del fluido sobre el objeto.

Así la presión total sería:

$$\mathbf{P_{total} = P_{atmosf.} + (\rho * g * h)}$$

Sin embargo el modelo sigue siendo una mera aproximación.

Salinidad [RO2005]

Se expresa como el número de gramos de sales inorgánicas disueltas por kilo de agua. La variación de la salinidad depende del clima global y por ello es un factor muy importante de cara a medir el cambio climático. Se observa un histograma con la variación de la salinidad en medio oceánico en la Figura 63.

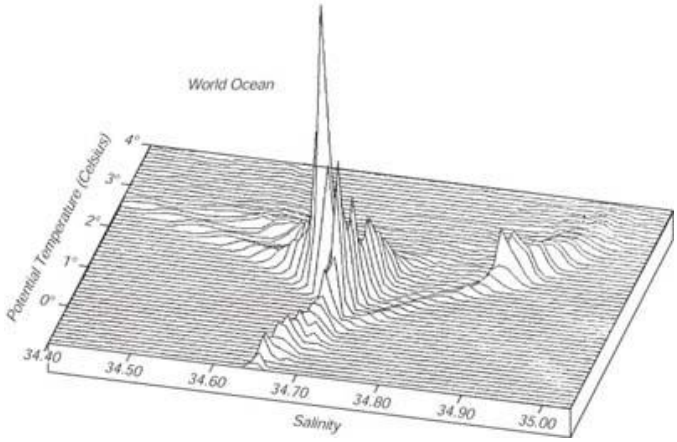


Figura 63: Histograma de temperaturas y salinidades en aguas frías de los océanos.

Suele medirse en psu (del inglés "Unidades Prácticas de Salinidad"). Tiene una escala internacional que permite la calibración de dispositivos, la PSS-78 (Practical Salinity Scale).

La salinidad se calcula a partir de la conductividad. A continuación se detalla la fórmula con la que puede obtenerse la salinidad (S) a partir de la conductividad (C). Decir de antemano que existen muchas y distintas aproximaciones, más o menos detalladas.

$$S = -0.08996 + 28.2929729\ R_{15} + 12.80832\ R_{15}^{2} - 10.67869\ R_{15}^{3} + 5.98624\ R_{15}^{4} - 1.32311\ R_{15}^{5}$$

$$R_{15} = C(s, 15, 0) / C(35, 15, 0)$$

Vemos que para calcular la Salinidad se usa la proporción R_{15}, donde C(S,T,P) es la conductividad de una muestra de mar a la temperatura T, presión P (0 indica que es la presión atmosférica nada más, ninguna presión del mar) con salinidad s. En concreto:

C(s,15,0): conductividad a 15ºC y presión atmosférica.

C(35,15,0): conductividad estándar "Copenhagen"

Otra aproximación interesante, más compleja, es la que incluye la corrección debido al efecto de la temperatura y presión:

1) Se considera C(S, T, P): la conductividad a una salinidad concreta, temperatura y Presión. También se considera C(35%, 15ºC, 0) = 42,914 mmho/cm

$$R(S,T,P) = \frac{C(S,T,P)}{C(35\%, 15^{o}C, 0)}$$

2) Se corrige el efecto de la presión en la conductividad por la fórmula:

$$R(S,T) = \frac{R(S,T,P)}{(1+F)}$$

Tal que: $$F = \frac{(1.60836\times10^{-5})P - (5.4845\times10^{-10})P^2 + (6.166\times10^{-15})P^3}{1 + (3.0786\times10^{-2})T + (3.169\times10^{-4})T^2}$$

3) Se corrige el efecto de la Temperatura,

$$R(S) = R(S,T) / r_T$$

4) Finalmente, se calcula la salinidad usando R(S) en la fórmula:

$$S = -0.08996 + 28.8567R + 12.18882R^2 - 10.61869R^3 + 5.9862R^4 - 1.32311R^5 + R(R-1)[0.0442T - 0.46\times10^{-3}T^2 - 4\times10^{-3}RT + (1.25\times10^{-4} - 2.9\times10^{-6}T)P]$$

Densidad [RO2005]

Se ve afectado por temperatura y salinidad y presión, que se relacionan en la Ecuación de Estado del Mar diseñada por la UNESCO y que relaciona todos estos parámetros (ver referencia **[UNE1981]**)

$$\delta(S,T,P) = \frac{\delta(S,T,0)}{1 - \frac{P}{K(S,T,P)}}$$

Donde δ es la densidad del agua marina en kg/m^3. T es la temperatura en ºC, y P la presión. Se puede calcular ρ(S,T,0) como sigue:

$$\delta(S,T,0) = \delta_0 + AS + BS^{\frac{3}{2}} + CS^2$$

Las constantes se calculan:

$$\delta = 999{,}842594 + 6{,}793952\times10^{-2}T - 9{,}095290\times10^{-3}T^2 + 1{,}001685\times10^{-4}T^3 - 1{,}120083\times10^{-6}T^4 + 6{,}536332\times10^{-9}T^5$$

$$A = 8{,}24493\times10^{-1} - 4{,}0899\times10^{-3}T + 7{,}6438\times10^{-5}T^2 - 8{,}2467\times10^{-7}T^3 + 5{,}3875\times10^{-9}T^4$$

$$B = -5{,}72466\times10^{-3} + 1{,}0227\times10^{-4}T - 1{,}6546\times10^{-6}T^2$$

$$C = 4{,}8314\times10^{-4}$$

Finalmente, se obtiene K(S,T,P). Una de las maneras más comunes para implementar esta ecuación es mediante redes neuronales. Es una alternativa heurística y que evita tener que evaluar la ecuación anterior en cada punto de una malla completa. En modelos de alta resolución esto puede ser muy costoso en tiempo.

Conductividad [NI1991]

Es la capacidad de conducción del mar. Suele medirse en MiliSiemens / cm. Viene afectado directamente por la salinidad, por lo que teniendo la salinidad se puede calcular la conductividad y viceversa.

Así, el radio de conductividad se calcula como:

$$R = C(S,t,p) / C(35,15,0)$$

Como vemos, es la conductividad in-situ dividida por la conductividad a salinidad 35 psu, temperatura 15ºC y presión atmosférico a nivel del mar (1 atmósfera, o lo que es lo mismo, 10.13250 dbars). C(35,15,0) se toma como la constante empírica 42,914 mS/cm.

Actitud

Actitud es la denominación que recibe la posición del vehículo respecto al horizonte. En los ejemplos de la Figura 64, Figura 65 y Figura 66 se muestra un avión, si bien es aplicable a los AUV en el entorno submarino respecto al horizonte.

La adquisición periódica de información de actitud permite, a la vez, actualizar la estimación de la deriva de los giróscopos y corregir periódicamente sus efectos sobre la estimación de la actitud. Los giróscopos son recalibrados cada vez que se recibe información de actitud lo que permite su predicción con alta precisión entre dos mediciones consecutivas y asegura la disponibilidad de información precisa a una alta tasa de muestreo.

Figura 64: Actitudes de cabeceo.

Figura 65: Actitudes de alabeo.

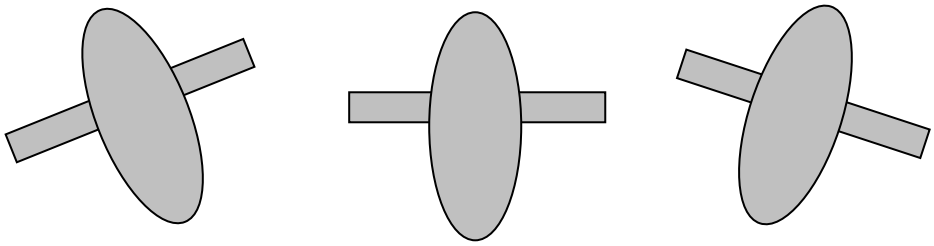


Figura 66: Actitudes de guiñada.

Velocidad

Se mide en m/s o nudos (knots), equivaliendo 1 nudo aproximadamente a 0.5 m/s. La velocidad que alcanzan los AUVs suele ser baja, ya que el consumo energético invertido en propulsión es un factor decisivo a la hora de limitar la autonomía del AUV.

Para estimar la velocidad de un AUV el instrumento más versátil es DVL (Doppler Velocity Logger) que tiene dos modos básicos de operación, en función de si es posible registrar el movimiento del AUV respecto al fondo marino o no.

Posición [DAV2003]

La posición de un AUV está formada por 3 componentes:

- Latitud: distancia angular medida sobre un meridiano, entre la línea del ecuador y el paralelo de una localización terrestre. Se mide en grados, y varía entre 0º y 90º norte, y entre 0º y 90º sur (ver Figura 67).

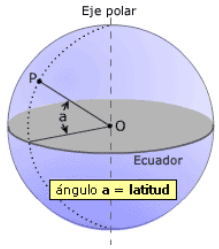


Figura 67: Latitud

Para determinar la Latitud de un punto P, hay que trazar el radio entre O y P. Pues bien, el ángulo de elevación de ese punto sobre el ecuador es la latitud. Latitud norte si está al norte del ecuador (como el ejemplo de la) y sur en caso contrario.

- Longitud: distancia angular entre el meridiano Greenwich (meridiano 0º) y el meridiano de un lugar, como se ve en la Figura 68. Son círculos completos que van de polo a polo. Puede medirse de varias maneras:

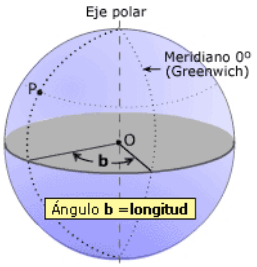


Figura 68: Longitud

  - De 0º a 180º hacia Este, y de 0º a -180º hacia Oeste.
  - De 0º a 180º, añadiendo una E si es Este o O (W en inglés) si es Oeste.
  - De 0º a 360º siempre medidos al este del meridiano Greenwich.

- Profundidad: este dato nos lo proporciona la batimetría, y se mide en metros desde el nivel del mar.

Distancia

Se mide en metros o en km. Es relevante considera el modelo geodésico en uso a la hora de estimar distancias a partir de las coordenadas de latitud y longitud de dos puntos, en particular si esos puntos están alejados el uno del otro.

7.2.- Modelos Sensoriales

Directrices Generales

En general, los sensores pueden ser simulados siguiendo las siguientes directrices:

- A la medida de cualquier sensor siempre se le puede aplicar un modelo de error, que permita simular errores en sensores reales.
- Muchos de los sensores que indican posición y orientación pueden ser modelados directamente con el dato de inclinación y orientación que presenta el vehículo en la simulación, sin necesidad de uso de ecuaciones específicas de funcionamiento de los aparatos de forma ideal.
- Los inclinómetros calculan su orientación de la orientación del cuerpo al que están enganchados o el fuselaje del AUV en este caso con respecto a la posición de equilibrio del inclinómetro.
- De forma similar, los giróscopos calculan su orientación según la velocidad angular del cuerpo al que van enganchados, y su propio eje de rotación.
- Los velocímetros calculan su velocidad en una dirección determinado de su eje de orientación e información de velocidad del cuerpo al que van enganchados.
- Todo sensor que sea de contacto puede ser simulado según la posición del objeto colisionante con respecto a la posición en la que se encuentra el sensor en el fuselaje del AUV. En función de esta posición relativa será la intensidad.

- Todo sensor que permita medidas de distancia, como sonars, etc, pueden ser simulados por técnicas de trazado de rayos, en principio, sin tener en cuenta efectos de reflexión-refracción debidos a cambios de densidad.

Vamos a hacer un repaso por los sensores más interesantes a equipar en un AUV, de acuerdo a las necesidades científicas para las que son usados los AUVs.

Medida de Distancia: Sonar Definición [COI2005]

Un sensor de ultrasonidos proporciona información de distancia por medida del tiempo de vuelo entre la iniciación del tren de pulsos y el retorno de su eco. Midiendo este "tiempo de vuelo"(TOF) y conociendo la velocidad del sonido en el medio es posible calcular la distancia cubierta en su recorrido de ida y vuelta por el tren sonoro.

Existen diferentes tipos de sónar: altimétrico, de imagen, de barrido lateral, … que se emplean para cubrir diferentes aspectos de la misión de un AUV en función de sus características. Así en los AUVs se emplean los altímetros sónar para mediar la altura de vuelo, o los sónar de imagen para evitación de obstáculos durante la navegación o reconocimiento de objetos. Los sónar de barrido lateral proporcionan precisos mapas batimétricos del fondo marino.

En realidad el haz de sonido produce ondas de sonido múltiples secundarias que interactúan sobre diferentes regiones del espacio alrededor del transductor antes de disiparse. Las ondas secundarias se denominan lóbulos laterales. La mayor parte de los sistemas robóticos asumen que solamente el sonido del lóbulo principal o más centrado es el responsable de la medida de distancia, pues el que presenta mayor resolución en la medida, como se aprecia en la Figura 69:

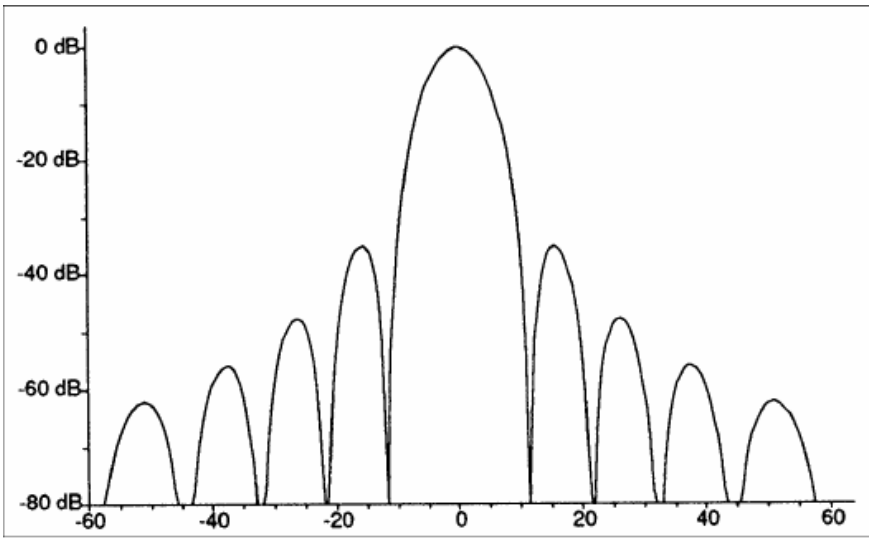


Figura 69: Lóbulo principal y ondas secundarias.

Otras características de este tipo de sensores son:

- *Activo*: envían una señal al entorno y miden la interacción de la señal con este (radar, sonar …)
- *Externo:* tratan con el mundo externo.
- *Sensor de distancia*: puede establecer la posición del AUV basado en mapa. Este tipo de sensores se usan en la navegación del vehículo y puede implicar cierto tipo de errores, sistemáticos y no sistemáticos en la posición estimada.

Modelo matemático.-

Un modelo básico y general sería:

$$d = \frac{1}{2} c\, t$$

Es decir, la distancia desde la posición del sonar al objeto (d) es la mitad de lo que tarda en llegar el eco, multiplicado por la velocidad a la que viaja el sonido (c) en ese medio. La velocidad del sonido depende fundamentalmente de la temperatura:

$$c = c_o + 0.6\,T \quad m/s$$

Errores del sónar.-

Existen una serie de fuentes de error inherentes a los sistemas basados en TOF entre ellas:

- Variaciones en la velocidad de propagación, debido al medio en el que se traslada. Puede que atraviese una corriente de agua o una zona donde la densidad cambie de forma abrupta y que haga que durante un tiempo, el rayo sónico se propague a velocidad distinta.
- Incertidumbres en la determinación exacta del tiempo de llegada del pulso Reflejado.
- Imprecisiones en la circuitería utilizada para medir el tiempo de vuelo de la onda.
- Interacciones de la onda incidente con la superficie del objeto. Puede presentar irregularidades que puedan provocar serias distorsiones en el eco devuelto.
- Crosstalk: puede que el sonar reciba ecos de otras superficies, o si hay otros objetos con sonares, señales provenientes de estos otros robots (interferencias) o del mismo robot pero ser una señal que es eco de otro sonar del robo. Se muestra en la Figura 70. Se suele solucionar promediando varias medidas repetidas, para así eliminar posibles ruidos.

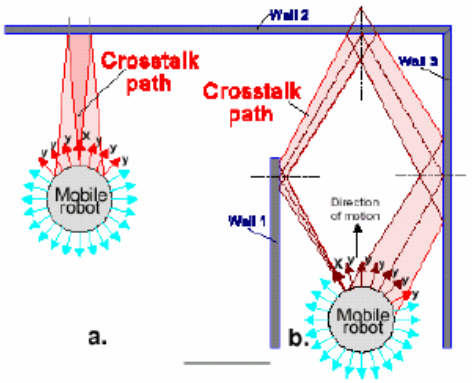


Figura 70: Crosstalk.

Como todo sensor, va a tener errores sistemáticos inherentes al sensor y no sistemáticos, como se muestra en la Figura 71. Pero usado para la navegación, siempre presenta un problema crítico, que es la **deriva**: la posición devuelta por el sonar debe ser

contrastada con el tiempo, y reevaluado. Para esto se suele usar el **filtro de Kalman** que trabaja con dos modelos: el modelo del vehículo y otro de las medidas.

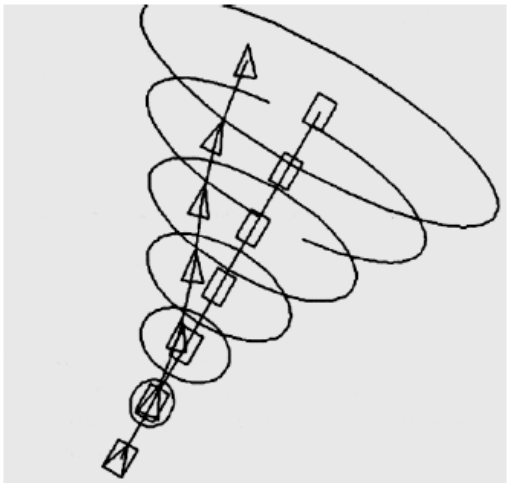


Figura 71: Errores en los sensores.

Medida de Relieve de Fondo Marino: Sonar de Barrido Lateral [AKE1999]

Es un tipo de sonar que se usa para creación de imágenes de amplias áreas del suelo marino, como la creación de mapas del entorno, mapas batimetrías, etc. No solo esto, sino detectar elementos cercanos al AUV. Se suele usar para detectar objetos concretos, clasificación de fondos marinos y clusterización de los mismos. Un ejemplo claro son las batimetrías, que suelen ser creadas a partir de estos sonars.

Este sensor emite haces de sonido en forma de abanico, en un amplio ángulo, perpendicular al camino que sigue el sensor, o el AUV o barco en el que está puesto dicho sensor. Avanzando el sensor a través de su camino, permite ir recolectando líneas que en su conjunto conforman todo el suelo marino que rodea el camino recorrido por el sensor. Las frecuencias de sonido que se suelen utilizar son del rango de 100 a 500 kHz, pero a más frecuencia mejor resolución. Son bastante utilizados, pues son relativamente baratos y fáciles de integrar en un sistema como un AUV.

Los sonars multihaz (como el mostrado en la Figura 72) pueden medir tanto la distancia al fondo como la intensidad que depende del material del que está hecho el fondo, respetando la misma filosofía expuesta.

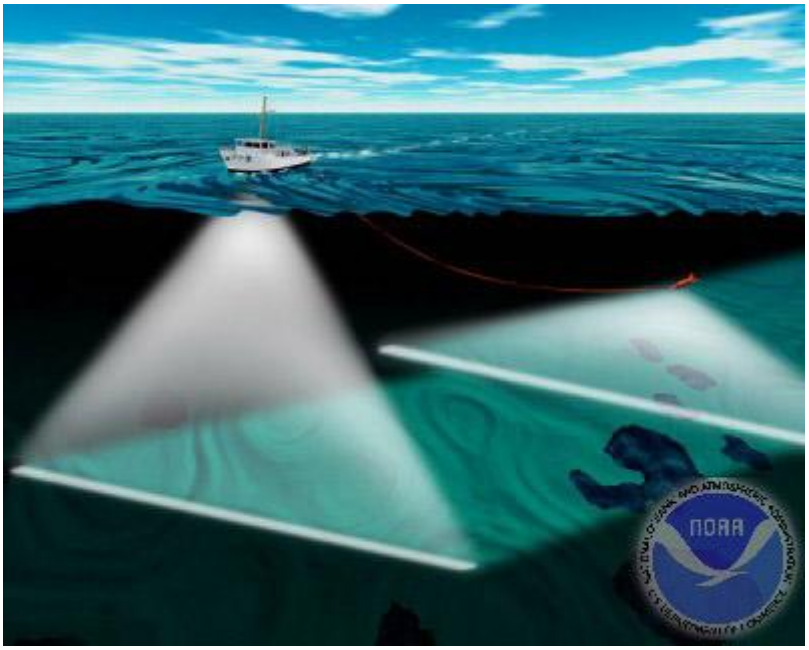


Figura 72: Ejemplo de sonars multihaz.

Modelo.-

Un modelo simple, aplicado al uso de este tipo de sensores en análisis de suelos marinos, es considerar que el sónar nos devuelve directamente el dato referido al fondo marino, pero hablaríamos de un sensor muy ideal, a no ser que en el modelo o función de distribución que asocia cada medida del sensor con un dato se introduzcan funciones de error.

Para hacer una simulación fidedigna de este tipo de sensores, hacen falta dos pasos:

- Formación de la imagen que se crea el sonar, al interaccionar con el entorno.
- Reconstrucción de las características del entorno a partir de la imagen del sonar.

Este modelo se va a acercar mucho más al proceso real, y nos permitirá reutilizar código. El mismo algoritmo de reconstrucción de las características del entorno a partir de la imagen del sonar, puede embeberse en el sistema de control del AUV para deducir información que afecte a su misión o navegación.

Figura 73: Intercepción del haz del sonar con superficie marina.

Formación de la Imagen.

Vemos en la Figura 73 que en el barrido lateral, el haz del sonar va a interceptar con una serie de puntos (**p**) pertenecientes al fondo marino u obstáculo. La imagen que se forme en el sensor, depende de la cantidad de energía difundida de la onda sonora que choca con el punto **p** y que llega al sensor en *o*.

Esta energía difundida depende de una serie de parámetros:

- del pulso emitido por el sonar, definido por el perfil del haz Φ(α). Como vemos no es **isotópico** (igual en todas las direcciones) sino que presenta diversos lóbulos. El lóbulo principal de acción es el central, por tanto los otros se suelen ignorar a la hora de simplificar el modelo. Depende este perfil del ángulo α que forma el vector que une **p** con el sensor, con el eje x del sensor.
- De la reflectividad en el punto **p**, que es lo que se denomina **R(p)**. Este parámetro depende del material del fondo, y es lo que nos permitirá clasificar ese punto del fondo en gravilla, arena, roca, etc.

El modelo de la difusión del rayo suele simplificarse y usarse el *modelo de Lambert,* que permite obtener la intensidad que en el sonar produce un determinado punto **p** en función de sus características. El modelo solo considera el ángulo de incidencia del pulso de sonido, y no el ángulo de observación ni la frecuencia del pulso.

La intensidad ***I,*** devuelta para un punto del fondo **p**, sería:

$$I(p)= \Phi(\alpha)\, R(p)\, \cos(\theta(p))$$

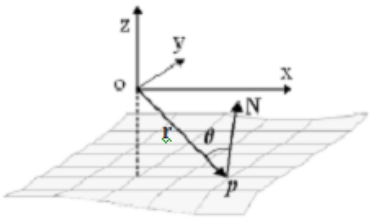


Figura 74: Ángulo de Incidencia θ

Finalmente, el ángulo de incidencia θ, que es el mostrado en la Figura 74, se puede calcular a partir del coseno que interviene en el producto escalar entre r y N:

$$\cos(\theta(p)) = \frac{\vec{r}(p) * \vec{N}(p)}{\left|\vec{r}(p)\right| * \left|\vec{N}(p)\right|}$$

El cálculo de **r** y **N** es muy sencillo: **r** se calcularía por diferencia de puntos entre *o* y *p*. El cálculo de **N** sería el producto vectorial de los vectores v1 y v2, que también se calculan por diferencia de puntos, como se muestra en la Figura 75:

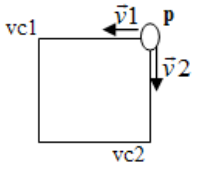


Figura 75: elementos para el cálculo de N.

La normal también puede ser aproximada de la forma

$$\vec{N}(\vec{p}) = \left(-\frac{\partial Z}{\partial x}(x,y), -\frac{\partial Z}{\partial y}(x,y), 1 \right)$$

Este método permite, a posteriori, hacer inferencias y establecer relaciones indirectas entre la intensidad devuelta y Z o nivel de profundidad del punto. Esta es la manera indirecta, a la que antes aludíamos, de deducir la altura de cada punto **p**.

Todo esto conforma una imagen del fondo, en forma de intensidades. Pero el modelo no incluye aún la atenuación de las ondas sonoras con la distancia. Por ello, la mayoría de dispositivos físicos corrigen este error mediante el Time-Varying Gain, que compensa el decaimiento de la intensidad con la distancia y con el ángulo de incidencia. En el lazo de control del sensor ultrasónico Polaroid 6500 (Figura 76), ampliamente empleado en robótica, ilustra los elementos básicos de un sistema ganancia variable.

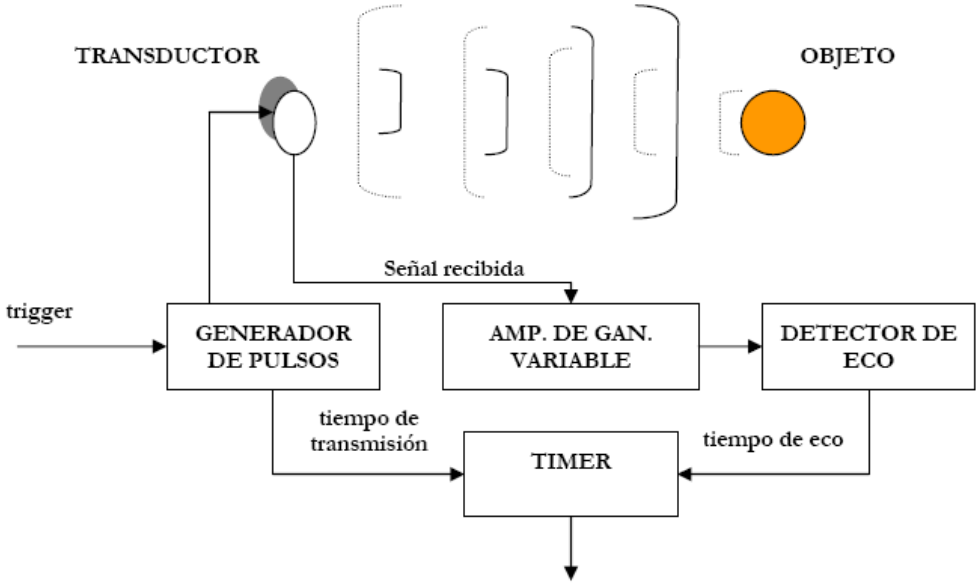


Figura 76: Lazo de control del sonar Polaroid 6500.

Un generador de pulsos sonoros, activa el timer. A partir de ahí, la señal devuelta pasa por el Time-Varying Gain, o lo que en este esquema se llama Amp. de Gan. Variable, que a modo de filtro, toma la señal y le aplica una compensación en función de la distancia. Finalmente, la señal amplificada pasa el detector del eco que vuelve al módulo timer, y detiene la cuenta del tiempo que pasa entre que se emite el pulso y se recibe. A partir de esta medida de tiempo se deducirá la distancia.

El decaimiento radial no ha sido considerado tampoco en este modelo, pero en resumidas cuentas y en pos de simplificar el modelo, se puede interpretar que estas compensaciones se incluyen en la función $\Phi(\alpha)$.

Estimación de parámetros.

La segunda parte de la simulación trata de analizar la imagen y deducir los parámetros que produjeron esos resultados. Este elemento va a ser muy interesante, ya que no sólo sirve para el simulador, sino que puede ser incorporado a la lógica de control del propio AUV para analizar los datos del sonar.

De forma resumida, esta estimación trata de hacer el camino inverso: dada la distribución de intensidades ***I*** en cada punto, deducir los 3 modelos antes utilizados, que son:

- **R**: nos da información sobre los materiales que conforman el suelo marino.
- **Z**: nos da información sobre la elevación de cada punto, por tanto permite deducir la profundidad o batimetría, y cercanía de objetos al AUV.
- **Φ(α)**: nos da la información asociada al modo en que se tomó la muestra en cada punto.

Para esto, se utiliza una técnica, denominada **Maximización de Expectativas** (EM, de las siglas en inglés de Expectation-Maximization**)** (mostrado en la Figura 77), que iterativamente irá convergiendo a la optimización de una serie de parámetros, dado una distribución ***I*** de partida del sensor y resultado del paso anterior.

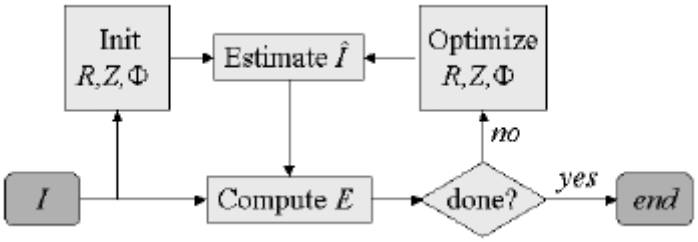


Figura 77: Técnica "expectation-maximization".

Iterativamente, se optimizan **R**, **Z**, y, **Φ,** y se estima a partir de ellos una distribución de Intensidad o ***Î*** *(Intensidad estimada)*, y con ésta se computa el error, para decidir si se ha alcanzado el óptimo y, por tanto, hay que salir o no.

El error se puede calcular como:

$$E = \sum_{x,y} E(x,y) = \sum_{x,y} \left(I(x,y) - \hat{I}(x,y)\right)^2$$

Básicamente, la diferencia entre las intensidades estimadas y reales en cada punto.

La optimización de los parámetros, se puede realizar de muy diversas formas. Una manera es mediante un *backpropagation* o retroalimentación: en función de la proporción de error debido a cada parámetro, se corrige el parámetro en sí:

$$R(x,y) \leftarrow R(x,y) - \lambda \cdot \frac{\partial E}{\partial R}(x,y)$$
$$\Phi(x,y) \leftarrow \Phi(x,y) - \lambda \cdot \frac{\partial E}{\partial \Phi}(x,y)$$
$$Z(x,y) \leftarrow Z(x,y) - \lambda \cdot \frac{\partial E}{\partial Z}(x,y)$$

Donde λ es una constante que permite controlar en que medida afecta el error a la corrección.

Pero esta manera puede ser muy pesada: véase que estamos intentando crear un simulador, por tanto, los tiempos pueden ser cruciales. La última tendencia es el uso de **algoritmos genéticos**, que son técnicas de búsqueda heurística que permiten encontrar el óptimo de una función multimodal.

Problemas asociados al modelo de Estimación de parámetros.

El principal problema del algoritmo es la resolución: tal y como se ha discretizado, solo tenemos datos puntuales, y es imposible tratar todos los puntos que puedan conformar el fondo marino por su coste computacional; por tanto, suelen realizarse interpolaciones que afectan a la calidad de la simulación.

Una solución, es la versión "multiresolución" que se muestra en la Figura 78: consiste en aplicar el proceso iterativo varias veces. En la primera vez, se consideran una R, Z y Φ nada optimizados. Por ejemplo, Z puede ser un fondo marino completamente plano. Se aplica el algoritmo, y se obtiene una distribución de R, Z y Φ. Pues bien, este resultado va a ser la entrada de la siguiente iteración y, por tanto, van a volver a entrar al

módulo **Init(R, Z, Φ)**, y se disminuye la cantidad de error que estamos dispuestos a aceptar para la siguiente salida.

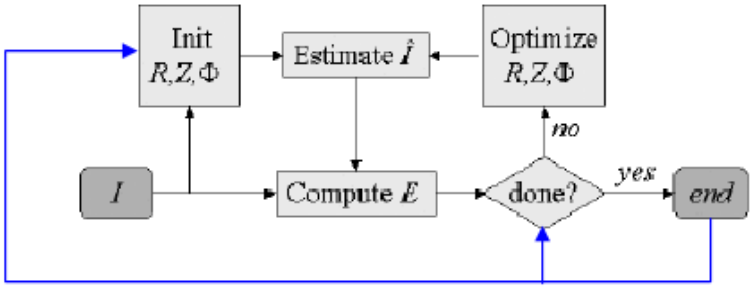


Figura 78: Versión multiresolución.

Medida de Actitud: Magnetómetro (brújula) [GIO2006], [NOR2003] y [UNNE]

Un magnetómetro mide la fuerza y dirección de campos magnéticos. Dado que la Tierra dispone de un campo magnético, una fuente disponible de información de actitud es el campo magnético terrestre. Un magnetómetro vectorial mide las componentes del campo magnético en las coordenadas del cuerpo y un modelo del campo magnético provee la dirección del campo en coordenadas de la Tierra. Entre ambos se extrae información de orientación respecto a la dirección del campo magnético local.

Esta información de la actitud, además de ser parcial, está afectada de algunos errores. En primer lugar, están los errores de medición del magnetómetro; en segundo lugar intervienen las anomalías locales del campo magnético, que es posible compensar si se dispone de un mapa de anomalías magnéticas, normalmente en forma de declinación magnética. Finalmente, se pueden producir errores debidas a la influencia de fuentes de ruido magnético tales como bobinas y motores presentes en el interior o en las proximidades del AUV. En cualquier caso, a pesar de sus limitaciones, y salvando condiciones extremas que podrían tenerse en cuenta durante la navegación, la medición del campo magnético resulta una fuente viable de información.

Un magnetómetro puede medir campos geomagnéticos en 3 ejes: Norte / Sur, Este / Oeste, Arriba / Abajo. Norte / Sur es la desviación en fuerza de la componente horizontal del campo geomagnético. Este / Oeste es la desviación en grados desde la dirección normal al norte magnético. Y Arriba / Abajo es la desviación en fuerza de la componente vertical del campo geomagnético.

Nótese que posee las mismas limitaciones que una brújula convencional: puede detectar el Norte magnético, que no coincide con el Norte geográfico debido a la declinación magnética, que es la diferencia entre ambos nortes, como se puede apreciar en la Figura 79. El norte geográfico, es constante, pero el magnético de la tierra varía constantemente. Esto hace que este tipo de aparatos, basados en campos magnéticos, no sean fiables al 100%.

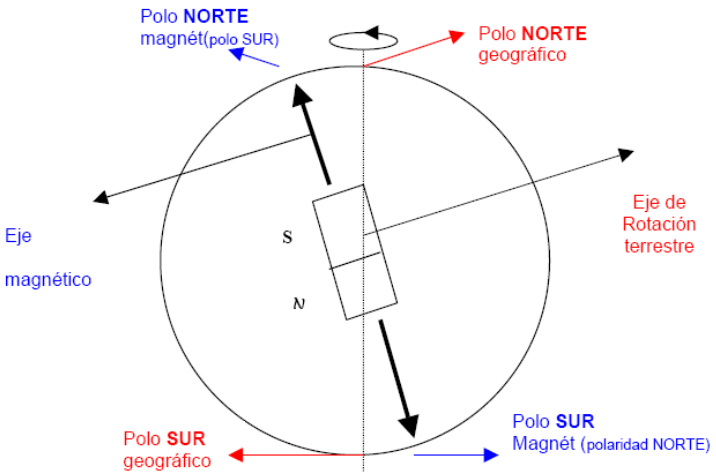


Figura 79: desviación de la orientación magnética frente a la geográfica.

Para desarrollar la simulación será necesario disponer de un modelo del campo magnético terrestre, en la zona de evolución del AUV, que proporcione una medida de orientación respecto del norte magnético.

Medida de Actitud: Inclinómetro [OT2001]

Por lo general, la brújula proporciona información del rumbo o dirección en que se mueve el vehículo o, lo que es lo mismo, la guiñada. El inclinómetro da información del **alabeo** y **cabeceo**. En principio es muy simple: se usa la inclinación de una superficie de referencia como medida de la orientación a lo largo de dos ejes perpendiculares entre sí. Esta superficie de referencia se une al esqueleto del AUV, de forma que cuando éste alabee o cabecee, por efecto de la gravedad, esta superficie de referencia presentará un cierto ángulo respecto a los ejes de referencia del inclinómetro. Un ejemplo se muestra en la Figura 80.

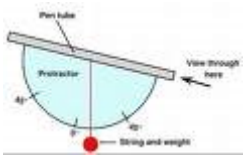


Figura 80: Principios básicos del inclinómetro.

Medida de Posición GPS

El Sistema de Posicionamiento Mundial es un Sistema Global de Navegación por Satélite (GNSS) el cual permite determinar en todo el mundo la posición de una persona, un vehículo o una nave, con una precisión hasta de centímetros usando GPS diferencial, aunque lo habitual son unos pocos metros.

El GPS funciona mediante una red de 24 satélites que se encuentran orbitando alrededor de la tierra. Cuando se desea determinar la posición, el aparato que se utiliza para ello localiza automáticamente como mínimo cuatro satélites de la red, de los que recibe unas señales indicando la posición y el reloj de cada uno de ellos. En base a estas señales, el aparato sincroniza el reloj del GPS y calcula el retraso de las señales, es decir, la distancia al satélite. Por "triangulación" calcula la posición en que éste se encuentra. La triangulación en el caso del GPS, a diferencia del caso 2D que consiste en averiguar el ángulo respecto de puntos conocidos, se basa en determinar la distancia de cada satélite respecto al punto de medición. Conocidas las distancias, se determina fácilmente la propia posición relativa respecto a los tres satélites. Conociendo además las coordenadas o posición de cada uno de ellos por la señal que emiten, se obtiene la posición absoluta o coordenadas reales del punto de medición. También se consigue una exactitud extrema en el reloj del GPS, similar a la de los relojes atómicos que desde tierra sincronizan a los satélites.

Este sistema de posicionamiento resuelve cualquier problema de posición para vehículos o dispositivos en la superficie. Pero el GPS no funciona bajo el agua, porque las ondas electromagnéticas para la comunicación satélite son atenuadas de forma rápida en pocos centímetros de agua marina.

A nivel práctico, simular un GPS a bordo del AUV solo implica proporcionar la posición exacta en el mapa simulado al AUV con un cierto retraso y error, en función del GPS.

Medida de Posición Bajo el Agua: LBL, SBL, USBL [BU2005]

La determinación de la posición de un AUV sumergido en uno de los problemas más complejos de la navegación de este tipo de vehículos. La solución tradicional a este problema consiste en corregir la posición del AUV cada vez que emerge usando el GPS y emplear un conjunto de sensores (brújula, giróscopos, …) y un filtro de Kalman para mantener una estima de la posición del AUV mientras está sumergido. La calidad de esta estimación se degrada rápidamente en el tiempo por razones evidentes.

La otra posibilidad es el empleo de **Sistemas de Posicionamiento acústico**. En estos sistemas se emplean diferentes combinaciones de transductores acústicos para determinar distancias y direcciones a emisores que se encuentran, bien anclados al fondo marino en posiciones conocidas, o bien se sitúan en superficie y pueden determinar su posición mediante GPS. Todos estos sistemas implican, en mayor o menor medida el despliegue de un sistema complejo de transductores. Existen diversas variantes.

Primeramente, el **Long Base Line (LBL)** es usado en grandes profundidades de más de 1000 metros de profundidad, como se muestra en la Figura 81. Lo conforman los siguientes elementos:

- Un grupo de Balizas o marcas de referencia situadas en el fondo marino. Éstas poseen lo que se llaman “respondedores”, y están situadas en posiciones fijas y bien conocidas, y que al recibir una señal acústica producen un eco, también acústicamente.
- Un transceptor montado sobre el AUV, que emite y recibe señales acústicas.

Así, el AUV va haciendo pings a todas las balizas y cada respondedor le responde con un determinado retardo, de manera que usando el retardo calcula la posición en que se encuentra respecto a cada baliza.

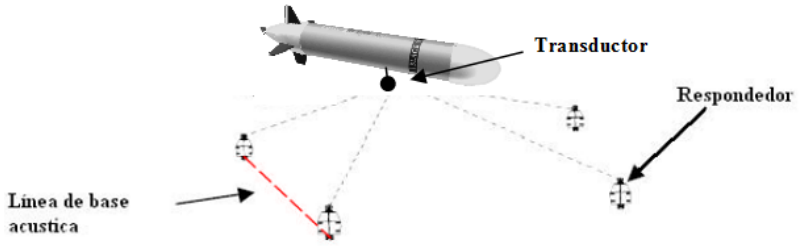


Figura 81: Ejemplo de LBL.

Integrando todas las medidas y conociendo la posición de cada baliza, el AUV puede obtener una estimación precisa de su propia posición. Es el método más exacto, pero requiere un proceso de calibración de las balizas, además de que el medio por el que se mueve el AUV debería ser acotado y controlado, ya que las balizas son fijas. Para recorridos demasiado largos puede ser una solución costosa dada la cantidad de balizas necesarias.

Por otra parte, el **Short Base Line (SBL)** está formado por los siguientes elementos:

- Una plataforma en la superficie, con 3 o 4 transductores de referencia. Ahora la referencia no es el fondo marino, sino la plataforma. También tiene un ordenador central que hace los cálculos para calcular la posición a partir de las señales recibidas por los transductores.
- Un respondedor en el AUV.

La forma de proceder es la siguiente (mostrada en la Figura 82): El AUV emite señal acústica, que es recibida por los transductores de la plataforma en superficie. De aquí pasa a un ordenador central, que hace el cálculo de la posición en que se encuentra el AUV, según la información que le envía cada transductor. Así, la redundancia se emplea para estimar la calidad de la exactitud de la posición sincronizada.

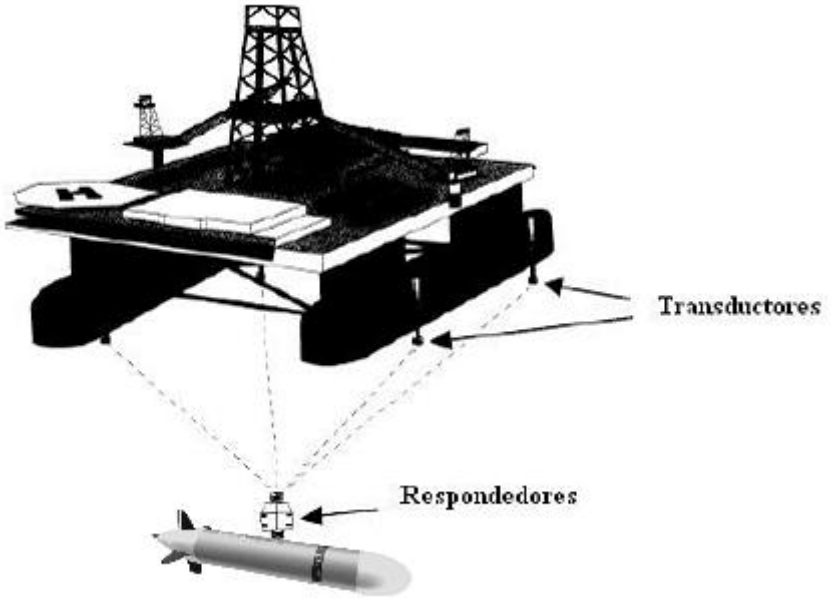


Figura 82: Ejemplo SBL

Finalmente, el **Ultra-Short Base Line (USBL)**, para cuando el AUV no está en grandes profundidades. Lo conforman estos elementos:

- Un transceptor: permite transmitir y recibir señales acústicas desde el medio marino. Normalmente montado en un barco o en una boya.
- Un transductor / receptor fijado en el AUV: permite recibir señales acústicas y responderlas a su propio modo también acústicamente. El procesamiento de la señal es mínimo.
- Un ordenador a bordo de la embarcación o dispositivo de superficie que contiene el transceptor. Hará los cálculos de la posición en función de la demora y rangos medidos en el transceptor.

El proceso es el siguiente: un pulso acústico se transmite por el transceptor. El transductor / Receptor en el AUV recibe la señal y la devuelve. Este eco es recibido por el transceptor a bordo del dispositivo de superficie. Así, el sistema USBL a bordo de la unidad de superficie calcula en función de la demora el rango.

Para calcular la posición se usa, por una parte, la posición GPS de la plataforma en la superficie. Para conocer el offset o el desplazamiento desde la posición GPS de la

plataforma hasta el AUV, se usa el rango, y el ángulo medido por el transceptor hasta el vehículo submarino.

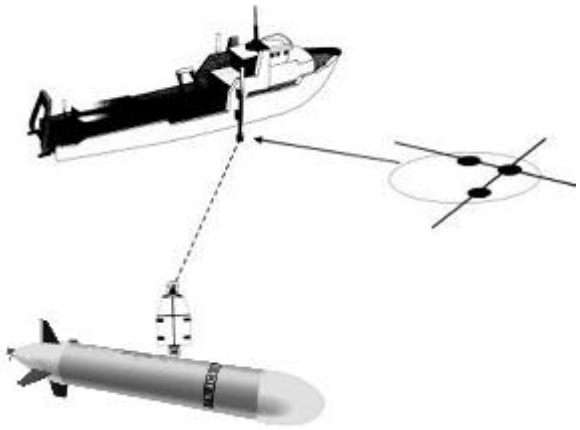


Figura 83: Ejemplo de USBL

Existe una variante, que es el **Inverted Ultra-Short Base Line (iUSBL)**, donde el transceptor y el centro de procesamiento para cálculo de posición se encuentran en el AUV, y el transductor /receptor en la superficie.

Medida de Actitud: Giróscopo [AGU2005]

Dispositivo que muestra el principio de conservación del momento angular. La versión mecánica de este dispositivo es en esencia una masa con forma de rueda girando alrededor de un eje, el cual está a su vez montado sobre un sistema que permite que el eje pueda tomar cualquier orientación. Una vez que está girando tiende a resistir los cambios en la orientación del eje de rotación. Los antiguos giróscopos mecánicos consistían en una masa esférica o en forma de disco montados sobre un soporte cardánico, de forma que pudiesen girar libremente en cualquier dirección. En la actualidad, los giróscopos pueden emplear diferentes principios de medida que definen su tipo, precisión y estabilidad. Así tenemos giróscopos de estado sólido, láser o de fibra óptica.

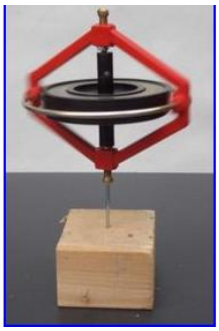


Figura 84: Giróscopo mecánico tipo "flying-wheel"

Aunque los diferentes tipos de giróscopos emplean diferentes efectos para determinar la aceleración angular, podemos emplear el modelo mecánico para ilustrar su modo de operación. Un giróscopo siempre tiene un eje en torno al que girar y éste nunca cambia, está fijo, como se muestra en la Figura 84. Siendo más exactos, carece de movimiento angular. Si el eje cambiase de posición angular se estaría cambiando el movimiento de rotación, para lo que se necesita una aceleración sin la cual no se puede hacer sin comunicarle una fuerza extra al giróscopo. Cuanto más masa tenga el cuerpo giratorio o más velocidad haya adquirido, más fuerza inercial adquiere ese movimiento de rotación, más difícil es variarlo pues hace falta más energía. Lo mismo se puede decir del eje en torno al que gira.

Esta es la propiedad que usa el giróscopo o giroscopio, y la tecnología lo ha aprovechado para mantener la estabilidad en los aviones y barcos, ya que cualquier mínimo cambio de posición del vehículo es detectado por el giroscopio al mantener su eje totalmente fijo respecto a aquel.

Se basa en dos principios:

- **Rigidez giroscópica**: se deriva de la ley del movimiento de Newton, según la cual un cuerpo tiende a continuar en su estado de reposo o movimiento uniforme si no está sometido a fuerzas externas. Es el ejemplo de la peonza, que puede moverse libremente a través de 2 ejes además de su eje de giro, pero no lo hace.
- **Precesión**: Cuando a alguno de los anillos que sostiene la masa, bien sea el horizontal o vertical, se le aplica un par de fuerzas perturbador, aparece este efecto. Un par aplicado da lugar a un giro, siendo este par un vector normal al plano en que tiene lugar el giro. El movimiento de precesión

tiende a llevar el vector que representa el giro del rotor a coincidir con el que presenta el par perturbador.

Dado un giro de la masa en cualquiera de sus ejes (por lo general 3 ejes), se deduce de él un grado de inclinación que presenta el AUV al que va sujeto el giróscopo.

La ecuación fundamental que describe el comportamiento del giróscopo es:

$$\vec{\tau} = \frac{d\vec{L}}{dt} = \frac{d(I\vec{\omega})}{dt} = I\vec{\alpha}$$

El vector **τ** es el momento sobre el centro de masa del giroscopio. El vector **L** es su momento angular o cinético. El vector **ω** es la velocidad angular. El vector **α** es la aceleración angular. El escalar *I* es su momento de inercia.

De aquí se deduce por las propiedades del producto vectorial que un momento **τ** aplicado perpendicularmente al eje de rotación, y por tanto perpendicular a **L**, resulta en un movimiento perpendicular a ambos **τ** y **L**. Este movimiento se denomina **precesión**. La velocidad angular de precesión Ω_P viene dada por:

$$\vec{\tau} = \vec{\Omega}_P \wedge \vec{L}$$

Medida de Temperatura: Sensor de temperatura

Hay termómetros de muy diversa índole. No presenta ninguna dificultad su simulación. Sólo hay que generar una distribución de temperaturas en el mapa o sección de mar donde navegue el AUV, y según la posición real en la que se encuentre el AUV y no la posición que mide mediante sus instrumentos, el valor de la temperatura en ese punto sea el que lea el AUV, añadiéndole distintos errores en función del sensor de temperatura usado.

Medida de Presión /Profundidad: Sensor de Presión / Profundidad

Se trata de un sensor que mide la presión hidrostática que después se convierte en profundidad, previa calibración del sistema. Un ejemplo, es el usado en el AUV URIS

[FAK2004] en la Figura 85, donde el sensor funciona con un bucle de 4-20 mA que se transforma en una tensión de 8-28V para el rango de 1 a 10 bars (0-100 metros).

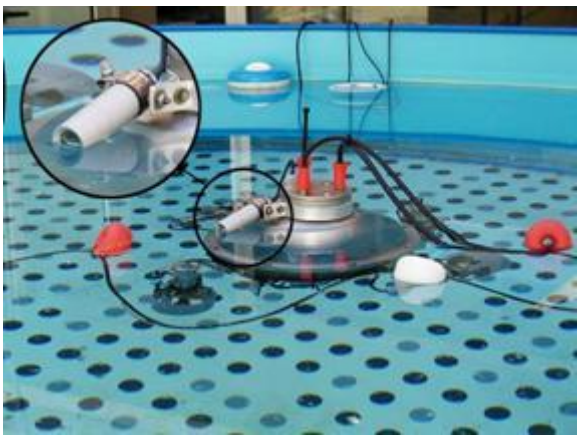


Figura 85: Sensor de Presión de URIS.

Simularlo consiste simplemente en suministrar al AUV la presión que el agua ejerce sobre el AUV, traduciendo esta presión en metros de profundidad según una sencilla interpolación lineal, tras una calibración previa, usando el modelo explicado en el **Apartado 7.1** sobre la presión/profundidad.

Medida de Conductividad, Temperatura y Profundidad: CTD "Conductivity, Temperature and Depth"

Figura 86: CTD

Las siglas CTD identifican de forma genética al sensor que permite medir tres parámetros esenciales en los estudios oceanográficos: temperatura, conductividad y profundidad. Además, de forma indirecta y a través de los parámetros anteriormente explicados también permite la medición de la salinidad. Concretamente a través de la conductividad.

Un CTD esta formado por un conjunto de "probetas" unidas a un cilindro central de metal, conectado por un cable a un ordenador o cualquier otro dispositivo. Así, estas "probetas" toman agua y realizan análisis que traducen en las medidas de los parámetros, y los transfieren por la interfaz del cilindro central al ordenador.

Medida de Velocidad: Sensor de Velocidad (DVL, Doppler Velocity Logger)

Es un sonar que utiliza 4 rayos acústicos que viajan en direcciones distintas pero conocidas como se muestra en la Figura 87, para poder calcular la velocidad relativa al medio marino a lo largo de 3 ejes de referencia ortogonales, X, Y, Z. También puede deducirse la posición X, Y, Z a partir de la información de velocidad, y de la situación inicial del DVL.

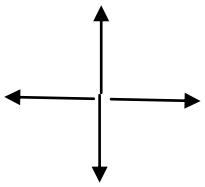


Figura 87: Rayos de DVL.

Un DVL es un dispositivo complejo que puede operar en varios modos, en función si puede recibir un eco del fondo o no. En el primer caso, puede proporcionar una estimación directa de la velocidad real (relativa al fondo) del AUV. Cuando se encuentra en una zona donde no recibe eco del fondo sólo puede determinar su velocidad relativa a la masa de agua que le rodea. Además, los modernos DVL pueden combinar sus propias medidas con medidas de procedentes de diferentes dispositivos de navegación, como giróscopos y brújulas, para producir una estimación de actitud del vehículo robusta mediante un filtro de Kalman.

Por su complejidad, no abordaremos la simulación de este dispositivo en este proyecto.

7.3.- Modelos de dispositivos de Comunicación.

Módem Acústico [ARL][DOSITS]

El principal problema de comunicación que existe en los AUVs y ROVs es la transmisión de datos, bien a través del agua para comunicarse con otros vehículos, o bien para enlazar con un satélite o una estación de radio situada fuera del agua. Evidentemente, los dispositivos para comunicación vía satélite y de posicionamiento GPS necesitan que el vehículo esté en superficie, ya que la señales electromagnéticas se atenúan rápidamente a

al penetrar la superficie del mar. A diferencia de las ondas electromagnéticas, las ondas acústicas pueden propagarse a largas distancia en el medio acuático, lo que hace que sean un medio más adecuado para la comunicación bajo el agua.

Así, los módems acústicos son usados para la transmisión de datos bajo el agua de forma inalámbrica, así como los módems telefónicos transmiten datos a través de la línea telefónica. Convierten datos digitales en ondas acústicas que se propagan a través del agua. Son usados en varios campos, desde la telemetría, como para las comunicaciones y control bajo el agua de los ROV y AUV, comunicaciones entre buzos, monitorización bajo el agua y logging de datos, y en general cualquier aplicación que requiera comunicaciones inalámbricas bajo el agua.

Figura 88: Módem Acústico usado en proyectos del NOAA.

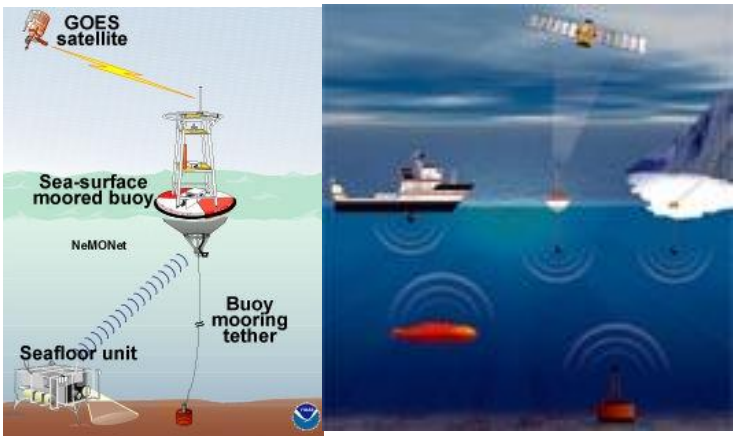


Figura 89: Uso de Módem acústicos en proyectos del NOAA

Un ejemplo de uso es uno de los proyectos del NOAA que se ilustra en las Figura 88 y Figura 89, donde un dispositivo toma fotos y temperatura en el fondo marino, y envía acústicamente los datos a una boya que transmite los datos vía satélite.

Comunicación de Larga Distancia: Comunicación Vía Satélite

Un satélite de comunicaciones permite la comunicación en zonas amplias o poco desarrolladas, ya que actúan como enormes antenas suspendidas en el cielo. Existen diversos tipos de satélite, en resumen:

- **Satélites orbitales o no síncronos**: giran alrededor de la Tierra en un patrón elíptico o circular de baja altitud. Si gira en la misma dirección que la Tierra y a velocidad angular superior a la de la Tierra, se llama **órbita de prograd**o. Si por el contrario, gira en sentido opuesto y a velocidad angular menor, es **órbita retrograda**.

  Por tanto, no permanecen estacionarios respecto a ningún punto en particular de la Tierra. Por ello solo se pueden usar cuando están disponibles: es decir, que el dispositivo que desea comunicar vía satélite tiene que esperar a que tenga cobertura del satélite, y entonces emitir. La cobertura dura unos 15 minutos como máximo.

- Satélites geoestacionarios o geosíncronos: giran alrededor de la Tierra en un patrón circular, con velocidad angular igual a la Tierra y en el mismo sentido de rotación de la misma. Por ello siempre cubren una posición fija con respecto a un punto de referencia de la Tierra.

Los elementos básicos de un sistema de comunicación por satélite son los siguientes:

- Una antena satélite: permite detectar cuando el satélite tiene cobertura, y así emitir datos al satélite.
- Un comunicador satélite: conectado a la antena, es el centro de proceso que envía datos, y los recibe, además de procesarlos para su uso.

En cuanto a la labor del satélite en sí, se encarga de tomar los datos enviados por comunicadores satélite y devolver la señal para que lleguen al comunicador destinatario de los datos. Por tanto, es una labor de mero repetidor.

Dependiendo del satélite usado, puede cubrir desde distancias concretas a prácticamente todo el globo terráqueo. Se muestra un ejemplo en la Figura 90, donde el

satélite interconecta centros de control en ubicaciones ditintas, y también al AUV navegando en océanos.

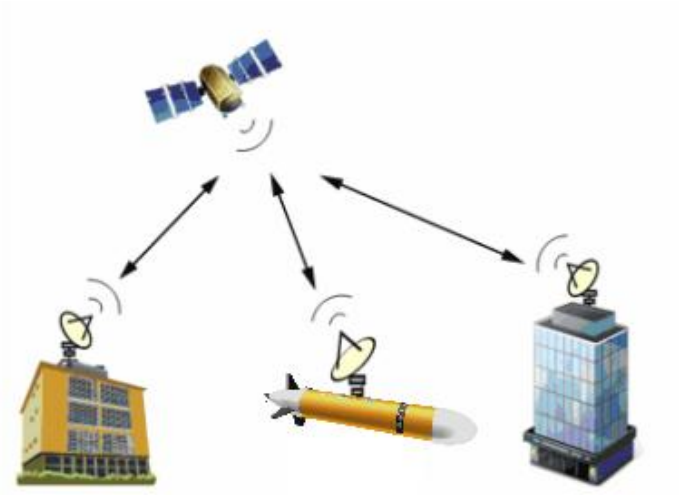


Figura 90: Comunicación vía satélite.

7.4.- Modelos de hidrodinámica de AUV

La hidrodinámica de un AUV es un parámetro importante en los entornos de simulación. El modelo hidrodinámico del mismo se traduce finalmente a la manera en la que se desplaza a través del medio submarino.

En la Figura 91 observamos un ejemplo, del AUV SWAN **[CAV2004],** donde se ve la propulsión (o vector T) de un motor orientado en las tres componentes tridimensionales, Tx, Ty, y Tz:

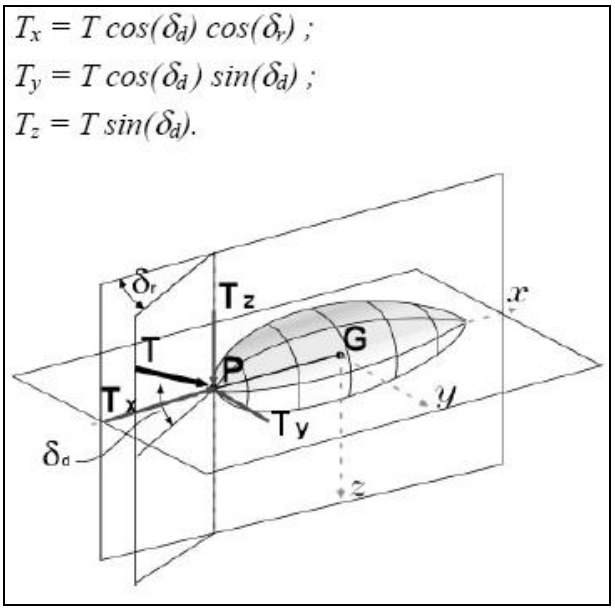


Figura 91: Orientación del motor orientado de un AUV.

Los modelos hidrodinámicos pueden tener mayor o menor grado de detalles:

- Desde un modelo que incluye la acción de cada uno de los actuadores del AUV, para conformar un movimiento global del mismo.
- A un modelo que considera al AUV un todo, sin desglosar en cada uno de sus actuadores.

Dada la fase inicial en que se encuentra el proyecto dentro del marco de la ULPGC, no se pretende modelar la hidrodinámica de los vehículos de manera fidedigna, si bien, para tal efecto habría que hacer un estudio de la referencia que se adjunta en la bibliografía **[F2002]**.

7.5.- Modelos Fisicoquímicos

Podríamos incluir en este apartado muchos modelos de magnitudes fisicoquímicas. Sin embargo, nos vamos en centrar en aquellos que estamos interesados en medir, de acuerdo con los modelos de sensores que hemos analizado. Los clasificaremos en dos tipos:

- Estáticos: su valor en cada coordenada tridimensional bajo el mar se mantiene invariable. Se pueden representar como una simple matriz, o una función independiente del tiempo.
- Dinámicos: su valor varía en función de otros aspectos, entre ellos el tiempo. Por ello suelen modelarse mediante funciones matemáticas dependientes del tiempo, o por autómatas celulares donde se representa el medio marino mediante una matriz y donde el valor de cada punto o celda puede hacerse depender o no de su vecindad.

Como meros ejemplos de posibles magnitudes dinámicas, presentaremos modelos para simular corrientes marinas y temperatura.

Simulación de Corrientes marinas

El movimiento de las corrientes marinas, al igual que corrientes termosalinas, de densidad, etc se suelen simular de dos maneras distintas:

- Mediante **función matemática**: dada una distribución inicial, en forma matricial donde a cada punto del espacio marino asigna un valor, se aplica una función que en cada instante de tiempo modifica dicha distribución, normalmente dependiendo del valor anterior de la función en ese punto o en su vecindad más cercana.

$$f(x,y,z)_{t+1} = A(f(x,y,z)_t)$$

El valor de la función puede ser una magnitud escalar, como el de densidad, el de salinidad, temperatura en grados Celsius, etc en el punto x, y, z, o puede ser una magnitud vectorial, como la velocidad de la corriente en el punto x, y, z. Con un vector se indica dirección, y con el módulo del mismo, se indica el valor de la velocidad.

- Mediante **Autómatas Celulares**: dada una distribución inicial, donde cada rejilla del mapa se considera una célula, que tiene un estado concreto (valor escalar o

vectorial), y dadas unas reglas de transición, en cada instante de tiempo, el estado de la célula se ve afectado por sus vecinos y su propio valor. Un ejemplo es el Juego de la Vida de Conway **[GAR1970]**, donde cada célula vive o muere en función de las que la rodean. En nuestro caso, cada región marina, incrementa un valor escalar o modifica la dirección de la corriente marina en función de el valor en las regiones marinas cercanas.

Autómatas Celulares: dinámica de fluidos. **[LI2003]**

Un autómata celular se compone de los siguientes componentes:

- Matriz de celdas bidimensional o tridimensional. En el caso que nos ocupa, simulación de corrientes marinas, será tridimensional, pues consideramos variaciones en profundidad. Esto implica dividir el fondo marino en el que se mueve el AUV en voxeles de igual tamaño todos.
- Cada voxel o celda puede estar en un conjunto finito de estados, que hay que especificar. En este caso, puede ser el valor de temperatura en ese punto, el valor de densidad en ese punto, el vector de velocidad en ese punto para una corriente marina, todos éstos discretizados y acotados superior e inferiormente en sus valores.
- Una configuración inicial: rellenar todos los voxeles con un estado inicial.
- Definir una vecindad, que son los voxeles que influyen con sus estados sobre el voxel a tratar.
- Un conjunto de Reglas de transición de estados que se aplica a cada celda. Suele ser una fórmula que asocia el estado de los vecinos y de la propia celda a un nuevo estado resultante, y sustituye el estado anterior por éste.
- Un reloj del sistema, que hace que en cada instante *t* cada celda se actualice con las reglas de transición, partiendo de un estado inicial, y así vaya evolucionando el sistema.

Simulación de Corrientes.-

Las reglas de transición definen el comportamiento del parámetro fisicoquímico; y cómo evoluciona en el tiempo. Un ejemplo muy recurrido es para simular fluidos, es la **Retícula HPP**. Hay muchas formas de diseñarlo

Veamos el caso 2D:

- En cada casilla, el "bloque de agua" se mueve en una sola dirección y sentido: hacia el norte, hacia el sur, hasta el Este o hacia el Oeste.
- La vecindad
- En cada instante de tiempo, si no hay colisiones, cada bloque se mueve a donde indica su estado anterior, donde el estado indica la dirección y sentido de movimiento.
- Si hay colisión, usar las siguientes reglas:
    - Si chocan dos bloques que se mueven en la misma dirección pero sentidos opuestos (Norte / Sur, Este / Oeste) entonces los bloques en el instante siguiente cambian el sentido de su movimiento, como se muestra en la Figura 92.

Figura 92: Cambio de sentido de bloques que colisionan en igual dirección.

    - En caso de chocar bloques con direcciones distintas, seguirán su curso como se muestra en la Figura 93.

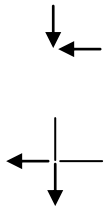


Figura 93: bloques colisionantes en direcciones distintas.

Existen muchas más configuraciones: los bloques pueden no ser rectangulares, sino triangulares, con lo que las direcciones son tanto horizontales como en diagonal, etc. Todo va en el diseño del Autómata Celular.

Por otro lado, se pueden refinar mucho estos Autómatas, e incluir múltiples direcciones (no solo norte, sur, este y oeste) a la vez que múltiples velocidades de choque, o la posibilidad de tratar la colisión de múltiples bloques, no solo 2 cada vez.

Veamos una aproximación 3D, que es la que nos interesa en el contexto del simulador que se pretende diseñar posteriormente.

- Cada voxel o bloque de agua se rodea de 26 vecinos, como en la Figura 94.

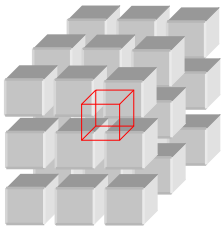


Figura 94: Voxel con 26 vecinos.

- Cada voxel tiene un vector de velocidad, con tres componentes (x, y, z).
- La velocidad de un voxel, es la sumatoria de los vectores velocidad de los 26 vecinos, más su propio vector de velocidad. En esta suma se puede usar ponderaciones, por ejemplo, para que la velocidad del propio voxel tenga más influencia que la de sus vecinos.

$$\vec{v}_{i+1} = \delta\vec{v}_i + \sum_{j=1}^{26} \beta\vec{v}_j$$

La velocidad en el instante siguiente ($\vec{v}_{i+1}$) es igual a la velocidad anterior ($\vec{v}_i$) por un factor o peso δ, más la suma de las velocidades del resto de vecinos en el momento anterior ($\vec{v}_j$) ponderador por el peso β. Un caso muy común es hacer una media de las velocidades de los vecinos y velocidad propia, con lo que β y δ son 1/27.

- Puede admitir una variante la regla de transición, y ser:

$$\vec{v}_{i+1} = \delta\vec{v}_i + \sum_{j=1}^{26} \beta\vec{v}_{j+1}$$

De manera que si sus vecinos ya están actualizados al instante t+1, se use el valor ya actualizado. Esto acelera la simulación.

Perfilación de simulación de corrientes.-

En muchos simuladores, la corriente es vista como una matriz de multidatos, donde cada casilla posee una velocidad máxima, mínima, un ángulo de ataque de la corriente y un ángulo de deslizamiento lateral. Esta matriz, aplicada una ecuación, permite la simulación de corrientes marinas.

Simulación de Evolución de Temperatura Marina: Función Matemática de Ecuación del Calor [PREE]

Esta ecuación expresa la distribución del calor de forma ideal, sin considerar el medio marino, salinidad, etc. para mapas en 2 dimensiones:

$$\frac{\partial u}{\partial t} - k\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0$$

Podemos aproximar esta ecuación diferencial de la manera que sigue en la Figura 95:

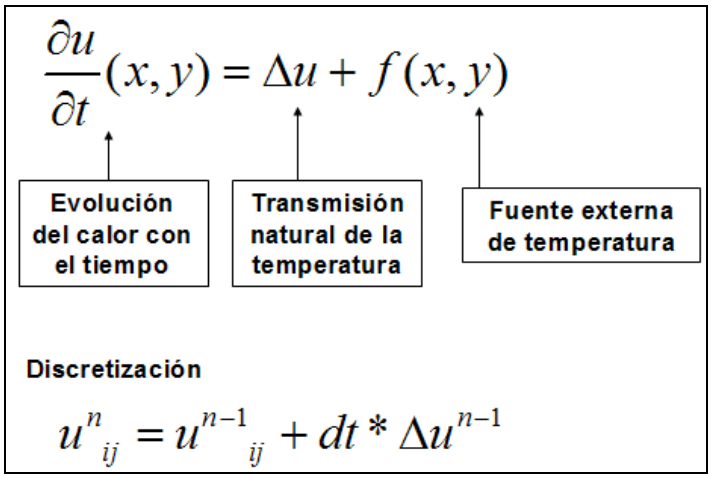


Figura 95: Aproximación de la ecuación diferencial de distribución de temperatura.

Como vemos en la Figura 95, el valor del calor en la casilla (i,j) es el valor en el instante anterior más una constante dt por el resultado de aplicar la máscara del laplaciano sobre sus vecinos. La máscara del laplaciano simplemente es sumar el valor de calor de todos los vecinos de distancia 1, es decir, los 8 vecinos que lo rodean, y restarle 8 multiplicado por el valor de calor en la casilla ij, todo esto en el instante anterior. f(x,y) puede tener cualquier valor que queramos.

En el caso de 3 dimensiones, la ecuación fundamental del calor es:

$$\frac{\partial u}{\partial t} - k\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) = 0$$

Véase el efecto de aplicar la ecuación del calor en un medio tridimensional en la Figura 96. La temperatura en el color rojo es mayor que en el color amarillo.

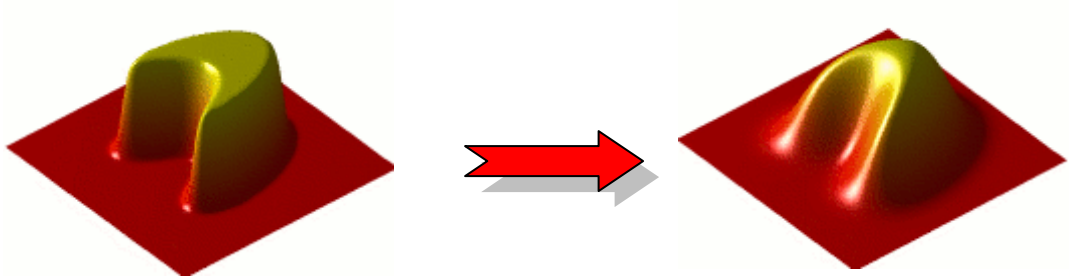
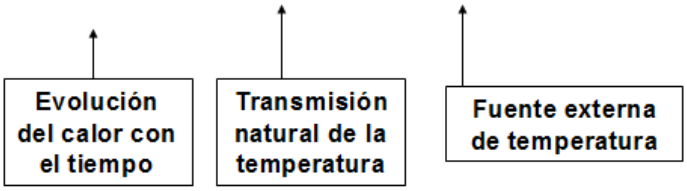


Figura 96: Efecto de ecuación de calor en 3D.

Un ejemplo de discretización lo encontramos en la referencia **[PET2005]:**

$$\frac{\partial u}{\partial t}(x,y,z) = \Delta u(x,y,z) + f(x,y,z)$$



$$u(x,y,z,t) = \frac{4\exp(-\frac{x^2+y^2+z^2}{5(t+1)})}{5\pi(t+1)}$$

donde,

$$f(x,y,z,t) = \frac{4(x^2+y^2+z^2+5(t+1))}{125\pi(t+1)^3}\exp(-\frac{x^2+y^2+z^2}{5(t+1)})$$

PARTE II

Arquitectura del Simulador

CAPÍTULO 8.- *Arquitectura General*

En el presente capítulo se sentarán las bases generales de la arquitectura del simulador objeto del proyecto, y que se ha apodado ***SUBES***, cuyas siglas significan "***Sub****acuatic* ***E****nvironment* ***S****imulator*".

Se describirán los subsistemas de lo forman, así como las herramientas para interaccionar entre sí.

8.1.- Arquitectura Cliente Servidor

Tras el estudio de diversas arquitecturas de distintos simuladores de vehículos autónomos submarinos actuales, realizado en el **Apartado 5.2**, la arquitectura cliente / servidor que se expone en el simulador CODA / CADCON es la que mejor se adapta a las necesidades y objetivos del proyecto. Esta arquitectura también guarda parecido arquitectónico con la solución MVS descrita en el mismo capítulo.

Este esquema distribuido cliente / servidor permite que SUBES ofrezca un entorno cuya complejidad sea escalable en función de los componentes que se integran en él, para simulaciones con resolución y nivel de abstracción adecuados a los objetivos del proyecto, sin cerrar las puertas a escalar en el futuro con módulos con modelos más realistas y a bajo nivel. Dado que el proyecto se desarrolla en el marco de un grupo de proyectos que abordan la temática de los AUV, es imprescindible no hacer un sistema cerrado y monolítico.

Definida la interfaz entre Servidor y Clientes, se pueden incorporar de forma **flexible** a la simulación toda una gama de clientes de diversa índole (AUVs, ROVs,, boyas, embarcaciones, obstáculos y fauna marina, cuadro de mando de monitorización y control de la simulación, etc) y en diversas plataformas que, siguiendo una interfaz de comunicación con el Servidor, permita usar los servicios del mismo para simular sensores, condiciones ambientales, parámetros fisicoquímicos sub-acuáticos, monitorizar misiones, etc.

Cada entidad del simulador (servidor y clientes) será diseñada **modularmente**, de forma que sea sencillo escalar su funcionalidad mediante el reemplazo de sus módulos.

Esto aporta al proyecto un valor añadido, pudiendo simplificarse muchos procesos inicialmente y aumentar la complejidad del sistema incrementalmente a medida que se vayan obteniendo resultados.

Como punto de partida, el simulador permitirá una simulación con un nivel medio de fidelidad: simplificando las ecuaciones de hidrodinámica, y reduciendo el AUV a un punto en el espacio. Esto significa que dejan de tener importancia la posición que ocupa cada dispositivo en el AUV, pues todo él se considera un único punto en el entorno virtual submarino. Esto no impide que en el futuro pueda darse otro enfoque al simulador mucho más realista. En futuros proyectos basados en los resultados del presente se podrán mejorar estos modelos.

Se planteará una solución para simular AUVs con misiones individuales. Estos AUVs podrán ser completamente lógicos o tener componentes hardware, o AUVs físicos que se conecten al simulador en modo "hardware-in-the-loop" (HIL) para simular que se conecte como cliente para simular ciertos aspectos como sensores no disponibles, o el entorno submarino y no tener que sumergirlo hasta que no se haya verificado.

No es objetivo del proyecto garantizar una simulación en tiempo real, al menos inicialmente.

8.2.- Descripción de Subsistemas Principales de *SUBES*

Siguiendo las características que en el **Apartado 5.2,** en la arquitectura CADCOM se describieron, se identifican tres tipos de subsistemas sistemas bien diferenciados: Servidor del Entorno, Clientes y Servidor Temporal.

- *Servidor del Entorno* (**SE**): es el motor o núcleo de la simulación. Por ello, debe garantizar:
  - **La Simulación del Entorno de Simulación**: con el objetivo de simular el medio marino, básicamente la variación especio-temporal de sus parámetros fisicoquímicos y la batimetría.

    De todos estos elementos, debe proporcionar el valor en cualquier punto del subacuático requerido y en cualquier instante, así como actualizar los datos de los modelos temporales con el avance temporal.

- **La Simulación de los dispositivos de los principales subsistemas que puede tener el Agente Móvil**: Debe permitir simular:
  - Cualquier sensor del agente móvil, proporcionando una medida de la magnitud deseada en cualquier posición que se encuentre el agente móvil.
  - Cualquier dispositivo de comunicación del agente móvil, simulando el retardo que supone el envío de un paquete de datos a través del medio marino o por medio de GPS, y garantizando que el paquete de datos llega a su destinatario, sea un agente móvil real, hardware-in-the-loop, etc.
  - Los actuadores o, simplificando, la hidrodinámica del agente móvil, calculando la variación de la posición de cada agente móvil simulado a través del motor en cada instante, y comunicando al correspondiente cliente el cambio de posición.

    En el simulador que se presenta, se considera agente móvil principalmente a los AUVs, pero el término extiende la simulación no solo a AUVs, sino también a ROVs y boyas.

- **Gestión de Servicios y Peticiones de clientes:** registrar toda petición, procesarla y generar una respuesta en su caso al cliente / clientes adecuados. Implícitamente conlleva:
  - Gestión adecuada de errores.
  - Evitar la inanición de cualquier petición.
  - Dirigir la petición al hilo correspondiente para balancear la carga del servidor y evitar demoras innecesarias en la respuesta.
  - Evitar el acceso no autorizado a los servicios del sistema por parte de los clientes.

- *Clientes*: usan los servicios del Servidor del Entorno para realizar su operación. Existen diversas variantes en función del tipo de su función dentro del proceso de simulación y, por tanto, de los servicios que va a solicitar del **SE**. Los clientes posibles son los siguientes:

- **Cliente /Servidor Agente Simulado (AS, de "Agente Simulado")**: En principio el simulador no debe tener un límite en el número de agentes a simular. Su objetivo es simular la operación de un agente real únicamente con software, de forma que si es un AUV, ROV, o boya, el objetivo es cumplir una misión especificada en un plan.

  La misión a cumplir por el agente, es traducida en medidas a realizar por sus sensores, paquetes de datos a comunicar, y cambios del vector de velocidad (con ello dirección y velocidad del agente) en los actuadores, y en general en el agente. Todas estas acciones básicas son pedidas al servidor del entorno.

  A su vez, existe una retroalimentación, y las medidas tomadas y devueltas por el servidor, o los paquetes de datos que llegan al agente a través del servidor en la simulación de comunicaciones, serán interpretadas por el agente y provocarán los cambios adecuados en el flujo de ejecución de la misión.

  También puede considerarse un cliente de este tipo a los AUVs, ROV o boyas tipo HIL (con hardware-in-the-loop), agentes no solo a nivel lógico, sino físico también. No simulan todos sus sensores o actuadores, sino solo parte de ellos. Es de gran utilidad para sensores caros o sencillamente no disponibles en laboratorio, o simulación de entornos virtuales que eviten pruebas de inmersión costosas. Por tanto la interfaz de comunicación con el servidor debe ser la misma en ambos casos.

  Desempeñan también función de servidor, los "Clientes Controladores" (de los que hablaremos más adelante) pueden pedir datos o sencillamente comandar al **AS**, por lo que estos clientes tienen que ofrecer un catálogo de servicios también.

- **Cliente de Interfaz Gráfica** (**CIG, "Cliente Interfaz Gráfica"**) La interfaz gráfica permite monitorizar y gestionar la simulación, tanto a nivel del entorno simulado como de los agentes simulados.

  - Entorno simulado: el cliente preguntará por la matriz de valores de una determinada magnitud en una determinada sección del mar, como puede ser batimetría, temperatura, salinidad, conductividad, presión, etc. Obtenida la matriz de datos, el objetivo es hacer una representación gráfica para el usuario.

- Agentes simulados: el cliente preguntará al servidor por la posición, estado del **AS**, batería, últimas medidas, etc. Obtenida la información, será representada bien gráficamente la posición en un mapa, o bien en un panel de visualización de datos del **AS**.

- **Cliente Configurador del Servidor**: (**CCS**) generalmente también está integrado desde la interfaz gráfica de usuario, trabajando en paralelo con el **CIG**, pues es el propio usuario el que quiere configurar el servidor. Si bien, puede también ejecutarse ad-hoc sin necesidad de una interfaz gráfica, accediendo a los ficheros de configuración del servidor, lo que requiere conocimientos avanzados de la aplicación).

  La misión es configurar los distintos parámetros del Servidor del Entorno, así como enviar el comando de comienzo y fin de simulación al servidor. Puede configurar desde los modelos usados para simular cada sensor, o cada magnitud marina, a características como contraseña, límite de agentes simulados máximos a simular, etc.

Existe otro elemento relacionado con el entorno de simulación de forma indirecta, si bien no va a ser objetivo de este proyecto desarrollar este elemento. Es el caso del **Cliente Controlador de Agente Simulado (CCAS)**: permite comandar al Agente Simulado, especialmente importante en los ROVs, pero también aplicables en AUVs, simulando permitiendo al investigador interaccionar con los **AS**. Por tanto se relaciona directamente con el **AS**.

No es objetivo de este proyecto estudiar este cliente, pues está más relacionado con el proyecto "Sistema integrado de planificación y control de misiones con Vehículos Submarinos Autónomos", dentro del grupo de proyectos dentro del cual se enmarca el presente proyecto.

- *Servidor Temporal* (**CLOCK**): se lo puede considerar como un "Meta-Sistema" en esta arquitectura, pues se encarga de la gestión del tiempo de simulación en todo momento –

como avanzar, parar el tiempo, etc.- y con ello coordinar a todos los subsistemas que participan de la simulación. Es el nexo de unión de los distintos subsistemas de ***SUBES***.

CLOCK se puede encontrar integrado dentro del **SE,** pues al fin y al cabo es un servidor puro con un catálogo de servicios. Considerarlo dentro del **SE** va a simplificar mucho tanto el diseño de la aplicación final como la implementación, pues va a aprovechar los mismos recursos que se implementen para el **SE**, mientras que por separado habría que duplicar esfuerzos en generar módulos para comunicación con subsistemas externos, etc. Además, se facilita el mantenimiento.

De todas formas puede tanto lógica como físicamente ser un componente aparte del **SE**. Se observan ambas alternativas en la Figura 97.

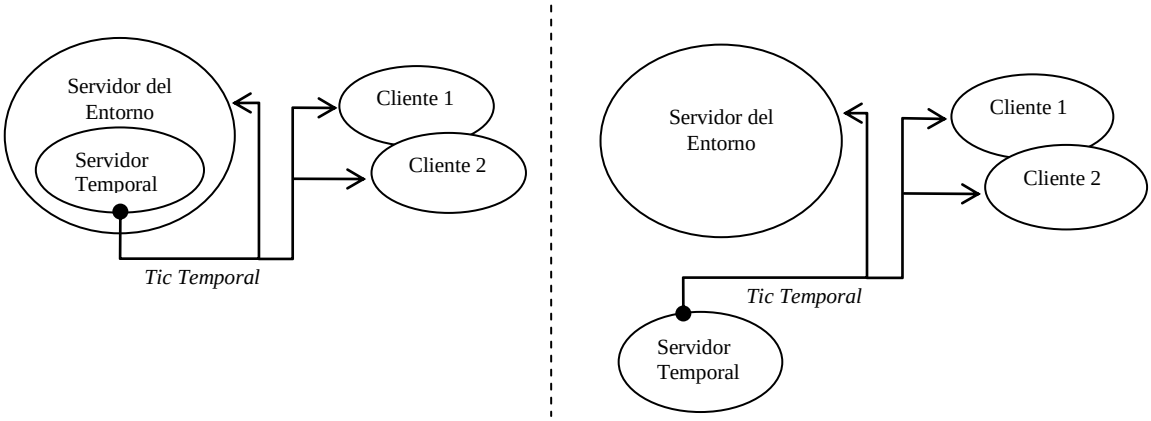


Figura 97: Servidor Temporal Embebido en SE, frente a Servidor Temporal independiente

8.3.- Integración entre Subsistemas del simulador

Como vía para intercomunicar a todos los subsistemas que forman ***SUBES*** se utilizará el protocolo de TCP/IP, como se muestra en la Figura 98:

- Esto suprime las restricciones espaciales, pudiendo cada componente ejecutarse independientemente en distintas ubicaciones a través de Internet, Intranet o en la misma máquina.
- Robustez y fiabilidad en las comunicaciones que aporta el protocolo TCP/IP.

- Independencia de la plataforma en la que se ejecute cada subsistema. Fundamental para la integración con otros proyectos de la ULPGC relativos al mundo de los AUVs.

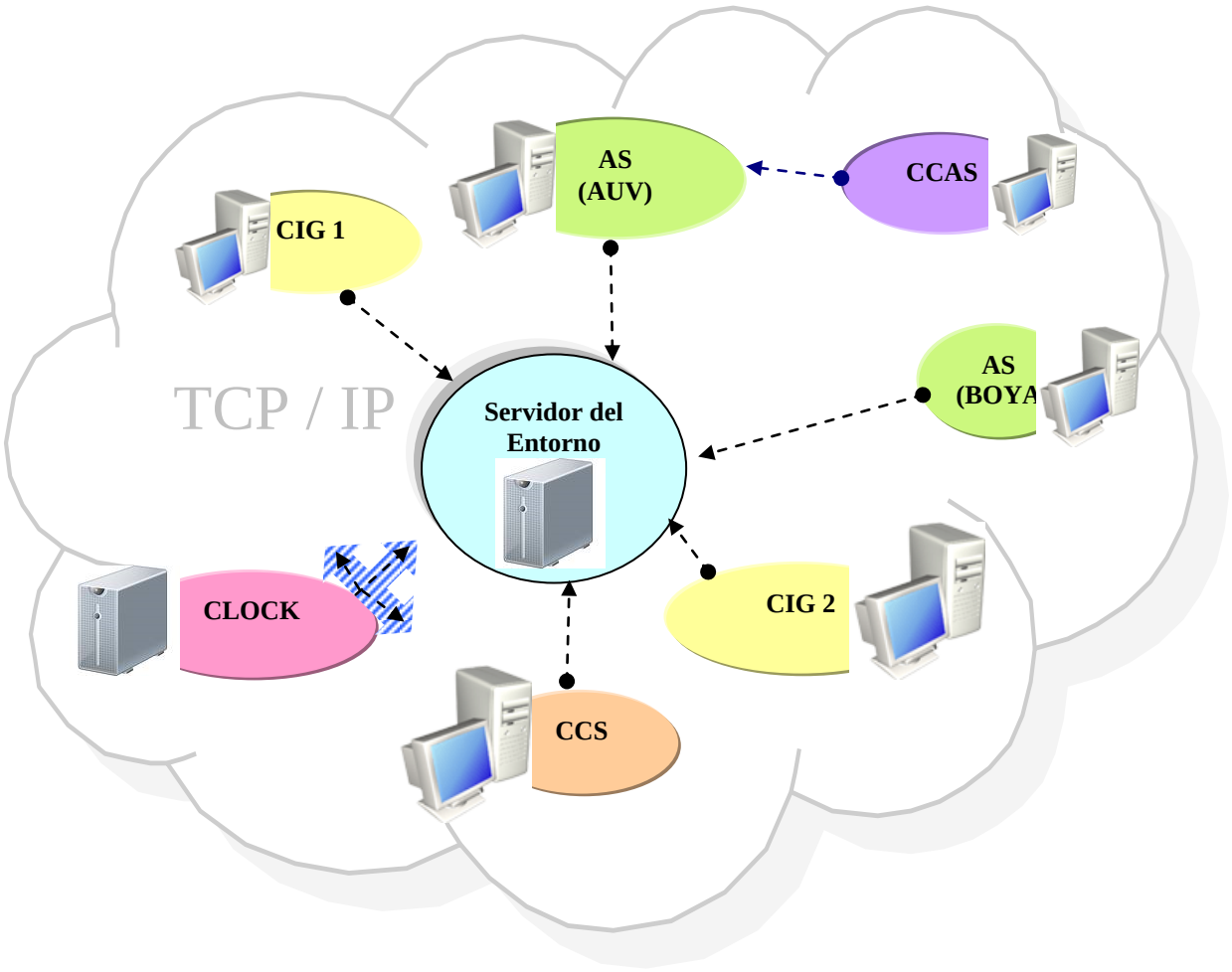


Figura 98: Subsistemas de la arquitectura SUBES .

El diseño de la comunicación del simulador se estructura en diversas capas de abstracción que se muestran en la Figura 99:

- **Elementos de Comunicación**: son los elementos base sobre los que se soporta toda la comunicación del sistema. Éstos son intercambiados entre los distintos subsistemas.
- **Flujo de Comunicación**: capa que define las interrelaciones posibles entre los subsistemas del simulador. No solo qué subsistema de comunica con qué otro, sino también la cardinalidad: cuántos se pueden conectar.

- **Protocolos de Comunicación**: capa que establece, entre cada subsistema, la secuencia de acciones necesarias para que se produzca la comunicación.

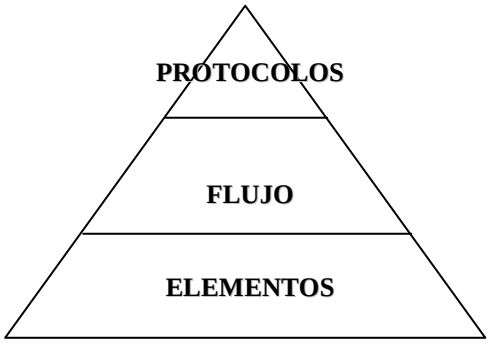


Figura 99: Capas de abstracción de SUBES.

8.3.1.- Elementos de Comunicación

La comunicación entre los distintos subsistemas se forma de los siguientes elementos de comunicación:

- Peticiones (**PET**): son pedidas pro activamente por el subsistemas origen, y demanda la ejecución de algún servicio en el servidor de destino, que puede desencadenar otros elementos de comunicación desde el servidor, como son respuesta al subsistema de origen, paquetes de datos de entrada, etc.

Figura 100: Símbolo para representar Peticiones

- Respuestas (**RES**): siempre van a colación de una petición hecha desde un subsistema origen, y son dirigidas a ese subsistema origen desde el servidor en el que se hizo la petición. Pueden ser de distinto tipo:
    - Respuesta: contiene un/os datos que fueron solicitados en la petición relacionada.
    - Erróneas: contiene un error, con una descripción que para que el cliente pueda conocer el porqué no recibe respuesta.

- Informes: es una simple confirmación de que la petición fue tratada. Es de utilidad en servicios que no tienen una respuesta específica asociada, ya que en las que tienen respuesta, la misma respuesta puede hacer las veces de informe.

Esto permite un seguimiento por parte del origen de la petición, y le permite tomar acciones, como por ejemplo realizar acción sólo cuando esté confirmado por el servidor la ejecución de un servicio.

Figura 101: Símbolo de respuestas.

- Tics de Reloj (**TIC**): este es un elemento de comunicación solamente emitido por el Servidor Temporal (**CLOCK**), e informa a todos los subsistemas del tiempo de simulación. El **CLOCK** lo envía pro activamente y no bajo petición. Por tanto no tiene por objetivo solicitar ninguna clase de servicio en el destino.

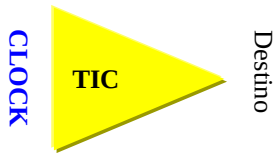


Figura 102: Símbolo de tics de reloj para sincronización

- Paquetes de datos de entrada (**DATA**): son enviados a subsistemas de destino sin que respondan a una petición previa. P.e.: cuando un **AS** simula comunicación a través del simulador, el **SE** enruta el paquete a comunicar al **AS** de destino sin que éste lo haya pedido de antemano.

Figura 103: Símbolo para paquetes de datos de entrada.

8.3.2.- Flujo de Información entre Subsistemas.

Se van a usar en este apartado diagramas específicos para la representación del flujo de información. Revisar el **ANEXO III**, apartado de **Diagramas de Flujo de Información entre Subsistemas** para más información.

Se van a recorrer los distintos subsistemas para describir la interacción con el resto de subsistemas. Comenzaremos por los más complejos que son los servidores Servidor del Entorno **SE** y Servidor Temporal **CLOCK**.

- *Servidor del Entorno (**SE**)*: ofrece un catálogo de servicios que puede ser accedido a través de peticiones por
  - Agentes Simulados (**AS**) que usan los recursos del servidor para simular sensores, actuadores, comunicación, etc.
  - Clientes Interfaz Gráfica (**CIG**) que usa los recursos del servidor para conocer el estado en que se encuentra la simulación en cada ciclo y representarla gráficamente para el usuario.
  - Clientes Configuradores de Servidor (**CCS**), cuya actividad en el servidor es puramente de envío de comandos para su configuración.

Estos servicios generarán en la mayoría de los casos respuestas que son enviadas al cliente adecuado. Incluso ciertos servicios podrían ocasionar respuestas a múltiples clientes. Por ejemplo, si se pide simular el envío de datos desde un dispositivo de comunicación de un AUV a otro/s, actuando el servidor del entorno como un enrutador del mensaje.

En la Figura 104 se muestra el flujo de comunicación posible, con los distintos elementos de comunicación que se pueden usar:

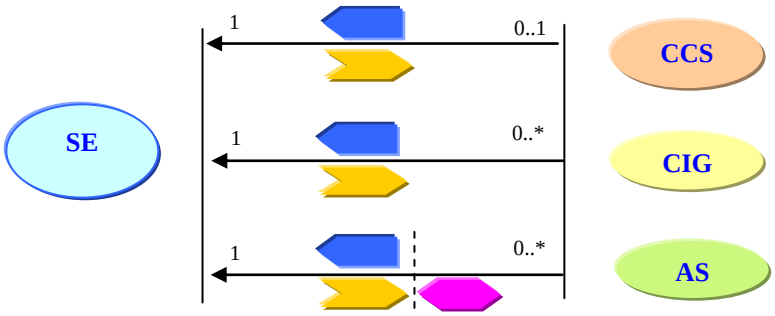


Figura 104: Flujo de comunicación con el SE.

Hay que resaltar que en la comunicación con el **CCS**, sólo puede haber un **CCS** conectado al mismo tiempo en el Servidor del Entorno. Es decir, sólo una aplicación que configure al servidor como máximo, o ninguna. Sin embargo durante la vida útil del **SE** puede conectarse a distintos **CCS** (uno a la vez). El resto de clientes se pueden dar múltiples instancias corriendo en el mismo servidor.

Otra peculiaridad es que el **SE** puede comunicarse con los **AS** mediante elementos de comunicación de tipo **DATA** y sin responder a una solicitud previa, por ejemplo, un broad-cast de un paquete comunicado de un **AS** a otro.

Todos los clientes (**CCS**, **CIG**, **AS**) pueden estar conectados únicamente a un sólo servidor del entorno (**SE**), que será el que simule los distintos parámetros y ofrezca los distintos servicios usados por los clientes.

- *Servidor Temporal* (**CLOCK**)

Primeramente y por claridad consideremos dos subsistemas separados el **CLOCK** del Servidor del Entorno (**SE**).

El **CLOCK**, aparte de ofrecer un catálogo de servicios a los distintos subsistemas de la arquitectura que se representan en el diagrama - que son accedidos mediante los elementos de comunicación petición-respuesta, segunda columna del diagrama - envía elementos de comunicación de tipo **TIC** en cada comienzo de ciclo de simulación una vez comenzada a todos los subsistemas registrados de la arquitectura ***SUBES*** de forma pro-activa. Se muestra en la Figura 105.

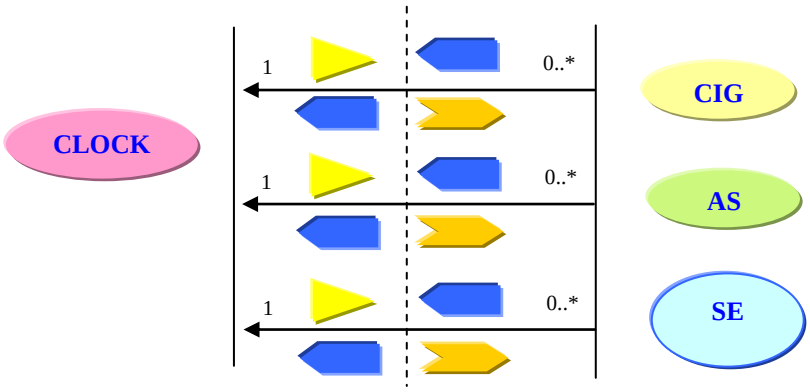


Figura 105: Flujo de comunicación con CLOCK independiente

La sincronización la hace con clientes cuya operación requiere conocimiento del comienzo de cada ciclo de simulación, por tanto son del tipo mostrado en la Figura 105:

- **CIG**, que cada ciclo tiene que solicitar la posición en que se encuentra cada **AS**.
- **AS**, que en cada ciclo tienen que actualizar su posición, además de realizar todas las medidas y actuaciones que les dicte su misión.
- **SE**, que gestionan la simulación de cada entorno de simulación virtual (parámetros fisicoquímicos, etc).

La cardinalidad en los 3 casos expresa que al mismo **CLOCK** se pueden conectar de 0 a muchos clientes de cada tipo, si bien cada uno de los clientes se debe conectar únicamente a 1.

No se sincroniza temporalmente con los clientes Configurador del Servidor ni con el Cliente Controlador de Agente Simulado, por ser clientes menos ligados a la simulación en sí y más centrados monitorización y configuración de ciertos subsistemas.

En el **Apartado 9.3** detallaremos por qué un **TIC** genera en todos los clientes una petición hacia el **CLOCK**, ya que forma parte del protocolo de sincronización de los subsistemas del simulador. En resumen, es una manera de que

el servidor temporal conozca si todos los clientes sincronizados ya han terminado las acciones que tenían que hacer durante el ciclo.

Consideremos ahora que el **CLOCK** es en realidad un módulo del **SE**, en cuyo caso el catálogo de servicios del **CLOCK** pasa a ser gestionados por el **SE**, de igual manera que el **SE** enviará los **TIC** durante la simulación y recibirá y gestionará las peticiones de los subsistemas.

La única diferencia sustancial es que en este caso, existirá una comunicación similar a la expresada en el diagrama anterior entre los módulos del **SE** y el **CLOCK** interno, con elementos de comunicación diferentes y adaptados al diseño que tendrá la comunicación entre módulos del **SE**, por lo que se expresan con diferente tono de colores, de manera que éste sincronice todos los módulos internos del **SE** con el resto de subsistemas del simulador. Se muestra en la Figura 106.

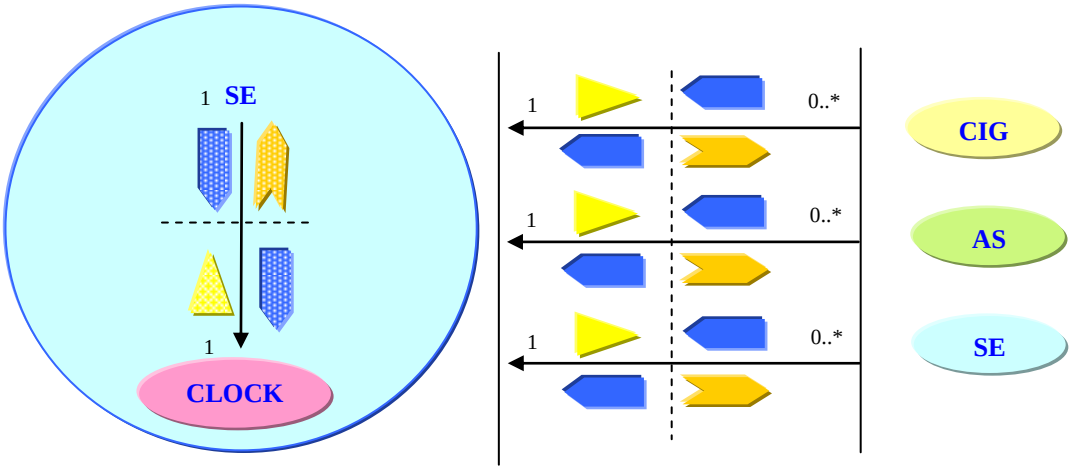


Figura 106: Flujo de Información con CLOCK embebido en SE.

Esto no quita la posibilidad de que un SE externo pueda sincronizarse con el **CLOCK** interno al **SE**. Si bien esto no es lo más habitual ni óptimo: Todo tiene que ser gestionado por el **SE'** que encapsula el **CLOCK**, con lo que la comunicación no es directa en un solo paso (**SE** → **CLOCK**), sino que tiene que dar dos pasos (**SE** → **SE**' → **CLOCK**)

En el caso de tener varios **SE** sincronizados con el mismo **CLOCK** es una arquitectura más sencilla y práctica que el **CLOCK** de ambos sea externo, como se muestra en la Figura 107.

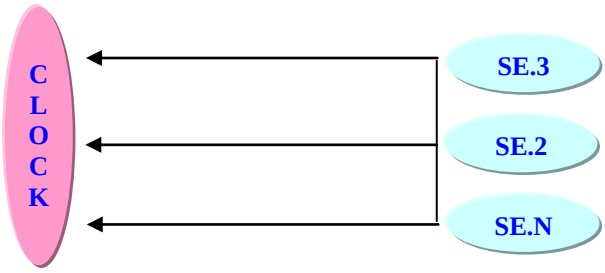


Figura 107: CLOCK compartido entre varios SEs.

Este aspecto del flujo de comunicación dota al sistema de gran potencia: pueden coexistir varios entornos de simulación simultáneos y sincronizados temporalmente. Es por tanto un banco de pruebas perfecto, donde podemos recrear y comparar distintas condiciones o aspectos simultáneamente, reduciendo el tiempo de pruebas de N entornos a uno sólo.

- *Cliente/Servidor Agente Simulado (**AS**):*

  Solo puede hacer peticiones a un solo **SE** y recibir del mismo, así como interpretar sus respuestas. Esta comunicación será necesaria para recrear ciertos subsistemas del **AS** que no estén implementados y se deseen simular, como es sensores, actuadores, dispositivos de comunicación, etc. En caso de necesitar comunicarse con otros **AS**, se hace indirectamente a través del Servidor del Entorno.

  Pero también actúa como "servidor", pues puede ser interrogado por un solo *Cliente Controlador de Agente Simulado* (**CCAS**) para telecomandarlo.[6]

  Ambas interacciones pueden verse en la Figura 108.

Figura 108: Flujo de comunicación del AS.

[6] A cada cliente / Servidor Simulador del Agente Simulado solo se puede conectar un Controlador de Agente Simulado cada vez, en exclusión mutua. Pero a lo largo del transcurso 1 mismo cliente/servidor simulador del gente Simulado puede haber estado conectado a distintos clientes Controladores de Agente Simulado..

- *Cliente Interfaz Gráfica (**CIG**)*

  Es un cliente puro, que envía peticiones a un solo **SE**. No tiene acceso a los **AS** por lo que cualquier información que necesite de los mismos se los pedirá al servidor, y éste le responderá en función de la representación interna que tiene de los **AS** en ese momento.

- *Cliente Controlador del Agente Simulado* (**CCAS**)

  No puede ser Interrogado, es un cliente puro, que envía peticiones a un solo **AS**, para pedir su estado, sus datos dinámicos, modificar su configuración, su misión, o pedir los datos recogidos por el cliente a través de sus sensores, y su misión.

  A cada cliente **AS** solo se puede conectar un **CCAS** cada vez, en exclusión mutua. Pero a lo largo de la vida de un **AS** puede haber estado conectado a distintos clientes Controladores de Agente Simulado.

- *Cliente Configurador del Servidor del Entorno:*

  Solo puede interrogar / comandar al SE para enviarle comandos de configuración y manejo de la simulación. Pueden conectarse varios al mismo Servidor del Entorno, pero uno cada vez. Establece sesión, y una vez abierta, se lanzan los comandos directamente contra el servidor, como puede verse en la Figura 109.

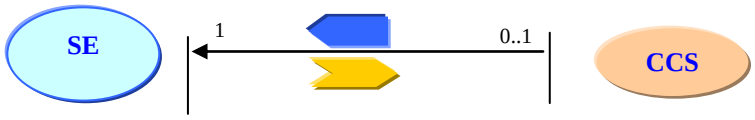


Figura 109: Flujo de comunicación del CCS.

Como ya se mencionó, sólo un **CCS** puede estar conectado al mismo tiempo al mismo **SE**, si bien durante la vida útil del **SE** puede haber estado conectado a diferentes **CCS**.

8.3.3.- Protocolos entre Subsistemas.

Este nivel de la comunicación entre los subsistemas del simulador se describirá por claridad en los capítulos que siguen al presente (**Capítulo 9, 10, 11 y 12)**, a medida que se describan los distintos subsistemas.

8.4.- Formato para Intercambio de Paquetes: serialización XDR

Este apartado persigue establecer el formato de los paquetes de comunicación que se intercambiarán los distintos subsistemas descritos en la arquitectura del simulador. Pero el objetivo es más amplio que el del presente proyecto: el formato de datos debe permitir la interacción entre subsistemas en cualquier plataforma. Así, dentro del marco de proyectos de la ULPGC que engloba al mismo, se encuentran dos proyectos más:

- Sistema Integrado de Control para un Vehículo Submarino Autónomo: destinado a implementar el lazo de control embebido en el AUV, gestionando todos los dispositivos (sensores, actuadores, dispositivos de comunicación, etc) y ejecutando los planes de misión. Este proyecto puede actuar como un **AS**, y conectarse al **SE** para simular sensores, actuadores, condiciones del entorno submarino sin necesidad de la inmersión del dispositivos, etc.

  Este proyecto, ya entregado y defendido por su autor Enrique Fernández Perdomo, usa C++ como lenguaje de programación principal.

- Sistema Integrado de Planificación y Control de Misiones con Vehículos Submarinos Autónomos: destinado a monitorizar la actividad de un AUV, planificar sus misiones y enviárselas, y enviar comandos de control que permitan hacer modificaciones durante la misión en curso.

Por tanto, puede actuar como el subsistema **CCAS** ligado a un **AS,** que puede ser el proyecto mentado anteriormente o un agente puramente simulado como los que abarca el

presente proyecto. Este proyecto, desarrollado por Eliezer Ramírez Cabrera, usa Java como lenguaje de programación principal.

Como se concluyó en el **Capítulo 4** el presente proyecto va a abordar la implementación usando Java como lenguaje de programación.

Establecido el objetivo, para el diseño del formato de los paquetes de intercambio se trabajó conjuntamente con los autores de ambos proyectos, para diseñar un formato común que cumpla con las necesidades, y resulte interpretable de forma eficiente en cualquiera de los proyectos descritos.

El resultado ya fue avanzado en el **Apartado 4.2**: El XDR es un formato estándar robusto y ampliamente utilizado para serialización de paquetes de información a través de protocolos como el TCP/IP. No solo garantiza el funcionamiento en el grupo de proyectos que se desarrollan en la ULPGC al respecto, también puede permitir colaboración con otros sistemas externos y al margen de la ULPGC.

8.4.1.- Tipos Básicos

Son los tipos de datos que se usarán como base para construir los mensajes. Se resumen en la Tabla 8:

Descripción	NombreClase
Entero Corto (4 bytes, con signo)	**Short**
Entero (4 bytes) con signo	**Integer**
Entero (4 bytes) sin signo	**Unsigned Int**
Flotante (4 bytes)	**Float**
Flotante doble precisión (8 bytes)	**Double**
Ristras de caracteres	**String**
Vector de ristras de caracteres	**String[]**
Valores buleanos (4 bytes, según XDR)	**Boolean**
Bytes de datos (8 bits, sin signo)	**Byte**

Tabla 8: tipos Básicos para serialización XDR.

8.4.2.- Encabezado Estándar del Paquete de Comunicación

Se van a distinguir distintos tipos de mensaje según la semántica y contexto en el que van a ser usados para la comunicación. Los mensajes implementan ciertas interfaces que ya proporcionan parte del formato del mismo; dichas interfaces se comentan en el

modelo de datos para las comunicaciones. Según éste, un mensaje es: **Identificable**, **Temporizable** y **Comunicable** (véase Figura 110), además de **Empaquetable,** lo cual ahora es indiferente. Por este motivo se comenta previamente la serialización de éstos y en el mensaje se comenta el orden de serialización de los mismos.

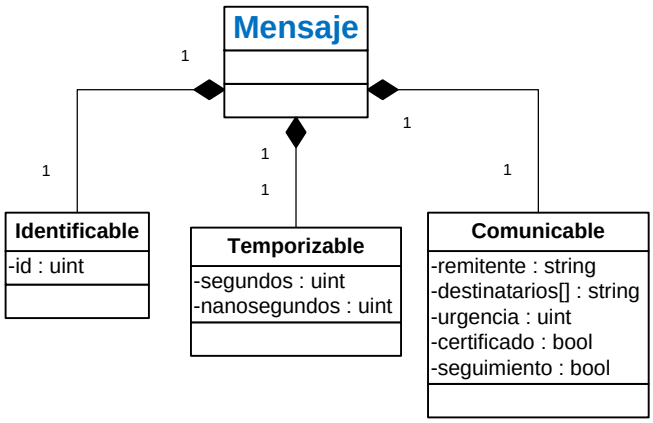


Figura 110: Diagrama UML de mensajes para serialización XDR.

Identificable

Semántica: Proporciona un identificador que distinga el mensaje unívocamente

Formato de Mensaje:

Serialización	Tipo	Nombre	Descripción
XDR	Unsigned Int	id	Identifica al elemento, y permite asociarlo con otros.

Tabla 9: Información incluida en la interfaz "Identificable".

Temporizable

Semántica: Proporciona un sello temporal, que señala el momento en el que se generó el mensaje.

Formato de Mensaje:

Serialización	Tipo	Nombre	Descripción
XDR	**Tiempo**	selloTemporal	Sello temporal del elemento.

Tabla 10: Información incluida en la interfaz "Temporizable".

Tipos de datos necesarios:

Tipo: Tiempo

Formato del Mensaje:

Serialización	Tipo	Nombre	Descripción
XDR	Unsigned int	segundos	Según la codificación segundos + nanosegundos transcurridos desde el 'epoch' de UNIX (1970).
XDR	Unsigned Int	nanosegundos	Según la codificación segundos + nanosegundos transcurridos desde el 'epoch' de UNIX (1970).

Tabla 11: Información que contiene el tipo compuesto "Tiempo".

Comunicable

Semántica: Proporciona el remitente, los destinatarios, la urgencia, e indica si el elemento está certificado y si tiene seguimiento.

Formato de Mensaje:

Serialización	Tipo	Nombre	Descripción
XDR	String	remitente	Dirección del emisor. Suele ser ip:puerto y para componentes CoolBOT se extiende a ip:puerto:componente:puerto.
XDR	String	destinatarios[]	Vector con las direcciones de destinatarios. Su formato suele ser como el indicado para el remitente.
XDR	Unsigned Int	urgencia	Indica el nivel de urgencia/prioridad.
XDR	Bool	certificado	Indica al receptor, que debe enviar una de aceptación o negación.
XDR	Bool	seguimiento	Activado, indica que el receptor del mensaje enviará informes al cliente cada vez que se produzca algún evento en la solicitud mientras es procesada.

Tabla 12: Información que contiene la interfaz "Comunicable".

8.4.3.- Mensajes para TeleComando

Semántica: Para transmitir comandos en forma de **solicitudes** para el control remoto fundamentalmente, las **respuestas** asociadas, e **informes** para monitorizar la solicitud hecha; las **respuestas** también serán **informes**. Es, por tanto, el paquete de comunicación más usado.

En los "Elementos de Comunicación" descrito en el **Apartado 8.3.1** de este capítulo, se corresponde con los elementos **TIC**  , **PET**  y **RES**  .

Se definirá la estructura de un TeleComando recursivamente.

Formato de Mensaje:

Serialización	Tipo	Nombre	Descripción
XDR	**Identificable**		Identificación del mensaje.
XDR	**Temporizable**		Sello temporal del mensaje.
XDR	**Comunicable**		Información de comunicación del mensaje.
XDR	String	asunto	Pueden ser tres tipos: **"solicitud" (**elemento de comunicación **PET**), **"informe" (RES), "tiempo" (TIC)**
XDR	***Cuerpo***	cuerpo	Datos adjuntos al mensaje. Dependiendo del **asunto**, el tipo Cuerpo varía.

Tabla 13: Mensaje para Telecomando en la serialización XDR.

Tipos de datos necesarios:

- El **cuerpo**, al ser una clase con diferentes herederas, debe desambigüarse, serializándose el nombre de la clase que es. Se muestra en la Figura 111.

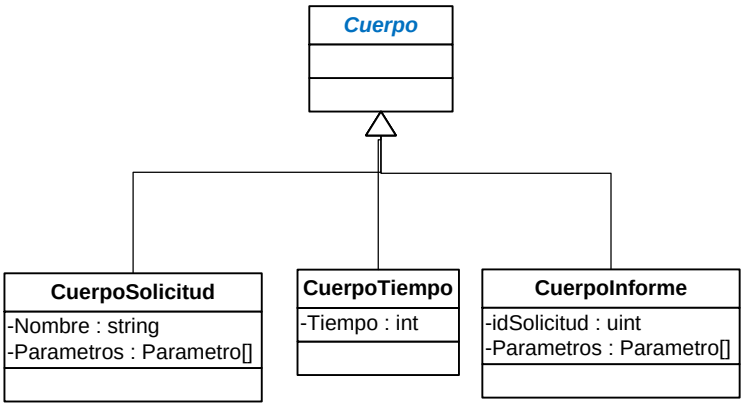


Figura 111: Diagrama UML de la clase Cuerpo.

Tipo: CuerpoSolicitud: **Cuerpo**

Semántica: Sólo se usa para asunto=solicitud. Se trata de un comando o petición (en **nombre**) con una ristra de parámetros. Se muestra en la Tabla 14.

Serialización	Tipo	Nombre	Descripción
XDR	String	nombre	Nombre del comando.
XDR	***Parámetro***	parámetros[]	Cada uno de los parámetros necesarios para ejecutar el comando anterior.

Tabla 14: Composición del tipo "CuerpoSolicitud".

Como se muestra en la Tabla 14, cada elemento es un comando (en **nombre)** con un vector de parámetros.

Tipo: CuerpoTiempo: **Cuerpo**

Semántica: Sólo se usa para asunto=tiempo. Tiene un único parámetro de tipo entero, que es el tiempo marcado. Sirve para marcar el comienzo de ciclo temporal, e indicar el tiempo de simulación o real que comienza. Se muestra en la Tabla 15.

Serialización	Tipo	Nombre	Descripción
XDR	Integer	Tiempo	Tiempo Marcado en el Time_Clock.

Tabla 15: Composición del tipo "CuerpoTiempo".

Tipo: CuerpoInforme: **Cuerpo**

Semántica: Solo se usa para asunto=informe. Este informe o respuesta está relacionado con una solicitud (idSolicitud) y contiene un vector de parámetros, que son la "respuesta" o datos del informe a colación de la solicitud relacionada. Se muestra en la Tabla 16

Serialización	Tipo	Nombre	Descripción
XDR	Unsigned Int	idSolicitud	Nombre del comando.
XDR	***Parámetro***	variables[]	Vector con variables que comunica el informe, o variables que son devueltas como respuestas.

Tabla 16: Composición del tipo "CuerpoInforme".

- El **Parámetro**, es una clase que representa un dato de un determinado tipo, como puede verse en la Tabla 17

Serialización	Tipo	Nombre	Descripción
XDR	String	nombre	Nombre de la parámetro, i.e. el parámetro formal
XDR	***Empaquetable***	valor	Valor del mismo como empaquetable, de modo que tendrá su propia serialización. Éste sería el parámetro real.

Tabla 17: Ttipo compuesto "Parámetro".

- El **Empaquetable**, al ser un clase con diferentes herederas (como se muestra en la Figura 112, debe desambigüarse, serializándose el nombre de la clase que es.

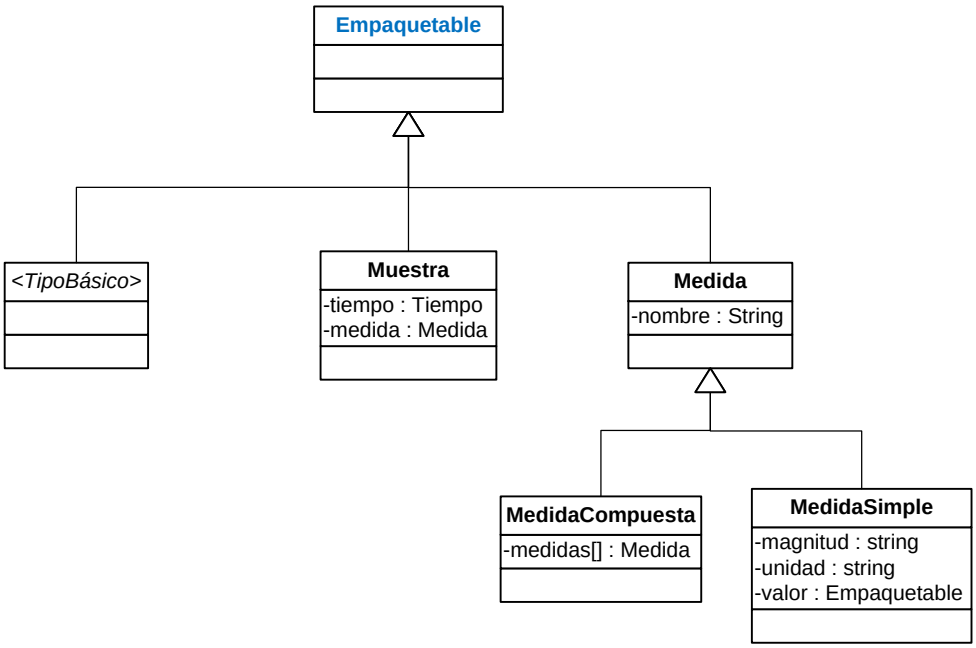


Figura 112: Diagrama UML del tipo Empaquetable y sus clases heredadas.

Tipo: *tipoBásico*

Semántica: Los tipos básicos utilizados en los **empaquetables** son los nombrados en el **Apartado 8.4.1**.

Tipo: **Muestra**

Semántica: Sirve para la transmisión de muestras temporizadas de medidas, normalmente de sensores del sistema. Será **Empaquetable**.

Formato de Mensaje:

Serialización	Tipo	Nombre	Descripción
XDR	**Temporizable**		Sello temporal del momento de medición de la muestra.
XDR	**Medida**	medida	Medida de la muestra.

Tabla 18 Tipo Compuesto "Muestra".

Tipos de datos necesarios:

Tipo: **Medida**

Semántica: Sirve para disponer de una medida, que tendrá un nombre (e.g. temperatura exterior); la medida podrá especializarse a dos tipos de medida diferentes, por lo que es una clase base abstracta. Será **Empaquetable**.

Formato de Mensaje:

Serialización	Tipo	Nombre	Descripción
XDR	String	nombre	Nombre de la medida (e.g. temperatura exterior).

Tabla 19: Tipo Compuesto "Medida"

Tipo: **MedidaCompuesta**

Semántica: Sirve para disponer de vector de medidas, teniendo así cada una organizada con su propio nombre. Será **Empaquetable**.

Formato de Mensaje:

Serialización	Tipo	Nombre	Descripción
XDR	**Medida**	medidas[]	Vector de medidas. Es útil para medidas compuesta como por ejemplo la posición tridimensional. Así, el vector tendrá 3 elementos, que serán **MedidaSimple**, donde el nombre de cada una puede ser: **longitud**, **latitud** y **altitud**.

Tabla 20: Tipo Compuesto "MedidaCompuesta"

Tipo: **MedidaSimple**

Semántica: Sirve para disponer de una medida simple, indicando la magnitud, el valor y la unidad; el valor será realmente el dato *RAW* del sensor, por lo que se deja como un **Empaquetable**. Será **Empaquetable**.

Formato de Mensaje:

Serialización	Tipo	Nombre	Descripción
XDR	String	magnitud	Indica el tipo de dato escalar que se envía.
XDR	**Empaquetable**	valor	Dato Escalar que se envía.
XDR	String	unidad	Unidad en la que se expresa el **valor** de la **magnitud** enviados.

Tabla 21: Tipo compuesto "MedidaSimple".

8.4.4.- Mensajes para Transmisión de Datos Escalares

Semántica: Sirve para la transmisión de datos escalares, principalmente muestras concretas. Evita tener que encapsular todos los datos en ficheros, con la redundancia que ello conlleva. En el marco de este proyecto, se corresponderían con los paquetes de dato **DATA**  .

También podría corresponder a la respuesta a una petición hecha por el **CCAS** al **AS**, en la que, por ejemplo, le solicite todos los datos almacenados durante la misión.

Por extensión, aparece el concepto de **Paquete**, que tiene un vector de elementos **Empaquetable** como **contenido**. Así, cualquier elemento que sea empaquetable se meterá en el vector **contenido** y se podrá enviar en el **Paquete**.

Formato del Mensaje:

Serialización	Tipo	Nombre	Descripción
XDR	**Identificable**		Identificación del paquete.
XDR	**Temporizable**		Sello temporal del paquete.
XDR	**Comunicable**		Información de comunicación del paquete.
XDR	**Empaquetable**	contenido[]	Es el vector de contenido. Cada elemento es un empaquetable, con su propia especificación de serialización.

Tabla 22: Composición de mensajes para Transmisión de Datos Escalares.

Un caso común de contenido a empaquetar en el mensaje expuesto en la Tabla 22 es la **Muestra**, que, por tanto, será **Empaquetable**; en principio, en el sistema se harán empaquetables casi todas las clases que sean susceptibles de comunicarse entre sí.

8.4.5.- Mensajes para Transmisión de Ficheros

Semántica: Se utiliza este formato para transmitir ficheros de cualquier índole, aunque lo más frecuente es que el formato sea alguno de los siguientes:

- Netcdf: matrices o vectores de datos muestreados o batimetría.
- XML: ficheros de configuración, descripción de AUV y subsistemas, y planes de misión.
- Plano: otros usos.

El envío de ficheros es más usual en la comunicación entre el **CCAS** y **AS** para enviar los planes de misión el **AS** al **CCAS**, etc., si bien puede corresponder a un paquete de tipo **RES**, como respuesta a una petición hecha por el **CIG** al **SE**, como por ejemplo, obtener el fichero batimétrico de una región.

Se integra como un **Empaquetable** que irá dentro de un Paquete, en lo que se refiere para el envío de la información de metadatos. La propia serialización de la clase con la información del fichero (previa indicación) serializará el fichero; ídem para deserializar.

Así, de la misma manera que se serializará el fichero (se abrirá como lectura, se sacará por el *stream* y se cerrará), también se deserializará (se creará un nuevo fichero con permiso de escritura y se escribirá en él lo recibido por el *stream* y luego se cerrará), Todo esto será en modo raw, es decir, sin seguir una serialización XDR.

Formato del Mensaje:

Serialización	Tipo	Nombre	Descripción
XDR	**Identificable**		Identificación del mensaje.
XDR	**Temporizable**		Sello temporal del mensaje.

XDR	**Comunicable**		Información de comunicación del mensaje.
XDR	String	tipoFichero	Indica el tipo del fichero, si es "**NetCDF", "xml"**, o **"plano"**.
XDR	String	nombre	Como cada fichero se suele asociar a un servicio (temperatura, batimetría, planes de misión, etc) por lo que el nombre podría ser **Servicio:ip** (la ip del emisor del fichero).
XDR	**Tiempo**	fechaCreacion	Fecha en que fue creado el fichero
XDR	**Tiempo**	fechaUltimoAcceso	Fecha en que fue accedido el fichero por última vez.
XDR	**Tiempo**	fechaModificacion	Fecha última en la que fue modificado el fichero.
XDR	Empaquetable	Fichero	Se envía el/los fichero en sí en modo **raw**, bien en formato en el formato que esté el fichero. .

Tabla 23:Composición de mensajes para Transmisión de Ficheros.

Donde en este caso el Empaquetable es un fichero, como se muestra en la Figura 113.

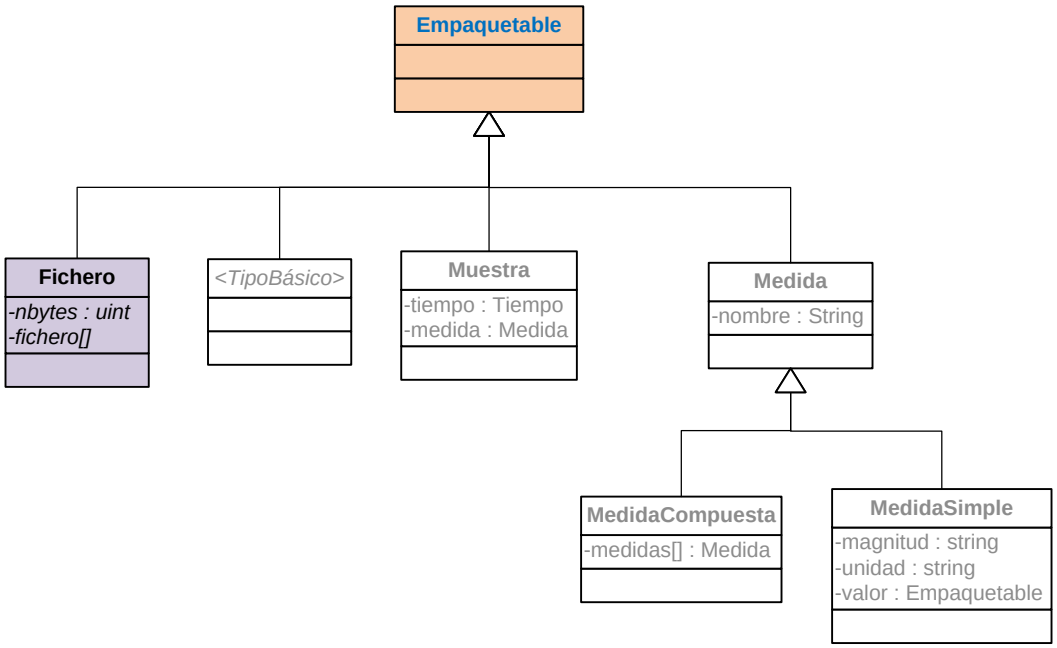


Figura 113: Diagrama UML con la clase "Fichero" heredera de "Empaquetable".

Sólo en los mensajes de transmisión de ficheros, el **Empaquetable** se serializa obligatoriamente como la clase "Fichero" de la forma mostrada en la Tabla 24:

Serialización	Tipo	Nombre	Descripción
XDR	Unsigned Integer	nbytes	Indica el tamaño del fichero. Esto permitirá al receptor leer el fichero por completo sin omitir información.
Binario	raw	nombre	Se envía el fichero en binario.

Tabla 24: Tipo compuesto "Fichero".

CAPÍTULO 9.- *Gestión Temporal y Sincronización del Simulador*

El objetivo es abordar de forma integral la sincronización de los distintos subsistemas descritos en el **Capítulo 8** de forma descendente: comenzaremos haciendo un análisis de la situación, pasando por el diseño del protocolo de sincronización, y finalizando con aspectos muy concretos del mismo. Junto al **Capítulo 8**, sentará las bases del proyecto sobre la que se desarrollarán los distintos subsistemas del simulador.

9.1.- Planteamiento General

En el **Apartado 5.3** se hizo un recorrido por las distintas técnicas para sincronizar procesos en arquitecturas distribuidas. Acorde al mismo y sus conclusiones, de las opciones expuestas se va a centrar el diseño del simulador en la opción 2, es decir, la figura de un servidor temporal central (véase Figura 114) que sincroniza al resto de subsistemas de la arquitectura. Dentro de las opciones la simulación será guiada por tiempo, por su sencillez respecto a la simulación guiada por eventos.

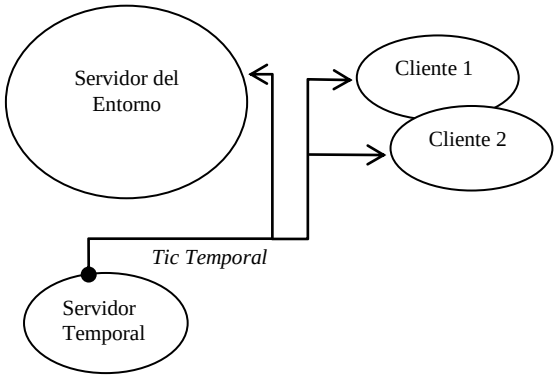


Figura 114: Servidor Temporal Central

Las funciones principales del Servidor Temporal centralizado serán:

- Suministrar información a todos los subsistemas sobre el estado de la simulación.
- Suministrar información a todos los subsistemas sobre el instante en que se encuentra la simulación, tanto proactivamente como bajo demanda de algún subsistema de la arquitectura.

- Gestionar el paso del tiempo (avance temporal) en función del estado de las tareas de todos los subsistemas. Es decir hay retroalimentación de los subsistemas.
- Gestión de contingencias ante clientes que pierdan conexión, etc.

Este servidor centralizado tiene dos opciones de configuración:

- La que se muestra en la Figura 114 anterior, en el que el Servidor Temporal es un subsistema más de la arquitectura independiente del Servidor del Entorno. Es ideal cuando se quieren sincronizar varios Servidores del Entorno.
- Un Servidor Temporal dentro del Servidor de Datos. La ventaja es que se simplifican las comunicaciones, ya que hay un único servidor tanto para datos como tiempo, pero presenta desventajas en caso de tener varios Servidores del Entorno que se quieran sincronizar.

  Por otra parte también se simplifica la implementación, puesto que no hay que implementar dos servidores, cada uno con sus protocolos de comunicación, sino que el Servidor Temporal Embebido utiliza los recursos ya implementados en el Servidor del Entorno.

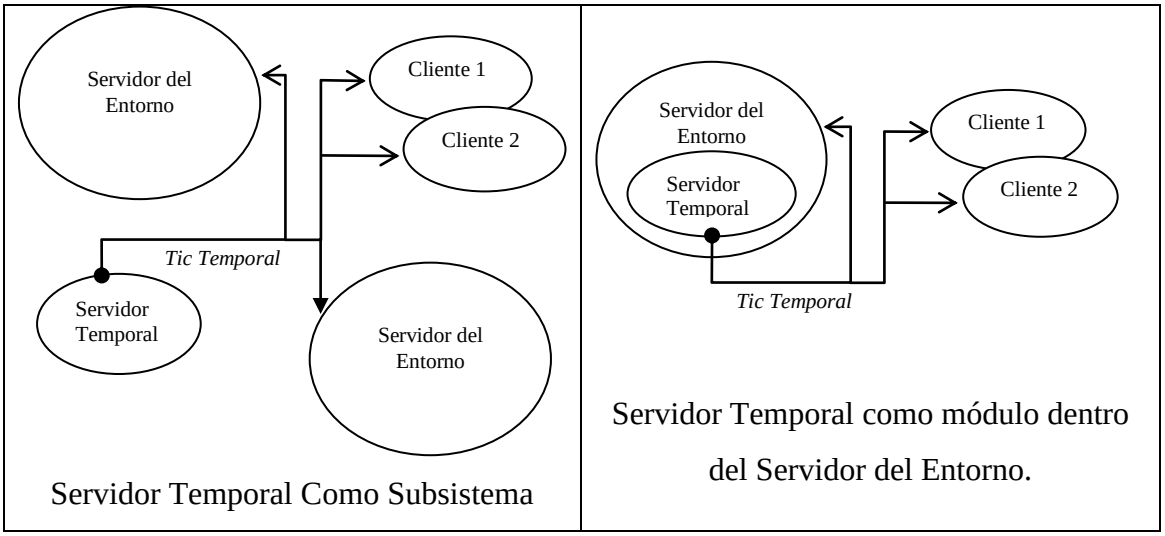


Figura 115: Comparación de Servidor Temporal independiente frente a embebido en SE.

Internamente, el Servidor Temporal se estructurará en tres componentes básicos que se muestran en la Figura 116:

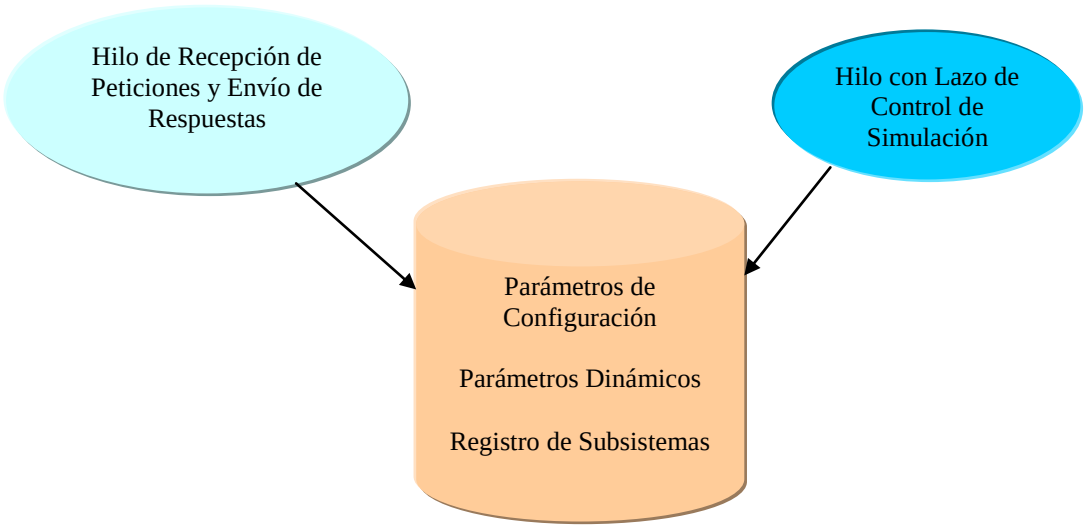


Figura 116: Componentes internos del Servidor Temporal.

Ambos hilos corren en paralelo. Mientras el de Recepción de Peticiones y Envío de Respuestas se encarga de la recepción y ejecución de peticiones desde los Subsistemas exteriores, el Lazo de Control tiene la lógica de la simulación. Ambos se interconexionan mediante una estructura de datos común, donde se encuentran parámetros de configuración y dinámicos, y el registro de subsistemas.

9.2.- Parametrización

A continuación se detallan en la Tabla 25 los parámetros que configuran la operación del subsistema encargado de gestionar el tiempo en el simulador.

Parámetro	Abreviación	Descripción	Información extra
Tiempo de Ciclo	*tCicle*	Duración real de cada ciclo de simulación. Se expresa en segundos	
Tiempo de Simulación	*tSimulated*	Duración a la que equivale en la simulación un ciclo. Se expresa	

		en Segundos.	
Límite de Espera	*MaxCicles*	Número de ciclos que el sistema espera la respuesta de todos los clientes. Pasado este tiempo, son expulsados los clientes que no manden la confirmación y se fuerza el avance temporal.	Esto evita que la simulación se detenga si uno de los clientes se bloquea o se pierde la conexión con el mismo.
Completar Tiempo de Ciclo	*Wait*	Indica si todo ciclo debe durar tCicle, o si un ciclo se resuelve antes se puede avanzar sin completar tCicle.	Permite acelerar la simulación o ralentizarla.
Esperar Cliente Externo	*ExternalSynchtonization*	Indica si el gestor espera la respuesta de los clientes para avanzar, o si avanza terminado el tCicle.	Siempre espera por el Servidor del Entorno, pues sin el mismo no puede proseguir la simulación.

Tabla 25: Parámetros del Servidor Temporal.

Explicaremos ahora en detalle las posibilidades de configuración del Servidor Temporal, incidiendo en la versatilidad del gestor temporal del presente proyecto.

Modo básico de Operación sin Sincronización Externa

Primeramente, recordar que los mensajes de tipo **TIC** fueron descritos en el **Capítulo 8**, y son aquellos mensajes destinados a la sincronización con el Servidor

Temporal. En cambio, los mensajes de tipo **PET** sirven para hacer una petición a un cliente.

El servidor Temporal envía cada tCicle segundos un mensaje de tipo **TIC** a todos los subsistemas de la simulación, que deberán estar registrados en el Servidor Temporal para que el mismo tenga conciencia de ellos, como es mostrado en la Figura 117.Con esto se consigue que los subsistemas de la simulación compartan un mismo reloj, si bien, no hay mecanismos para supervisar cuando un subsistema necesita más tiempo de tCicle para ejecutar todas sus tareas. Esto obligará a adoptar una solución de compromiso:

- Un tCicle demasiado grande ralentiza toda la simulación.
- Un tCicle demasiado ajustado hace que si un cliente no tiene tiempo de ejecutar sus tareas termine desincronizándose.

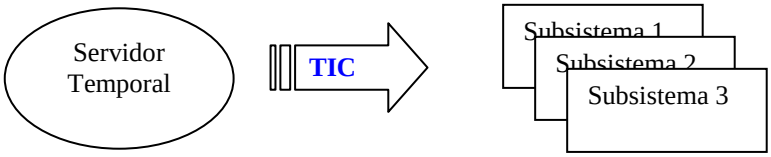


Figura 117: Distribución del TIC a subsistemas de SUBES.

ExternalSynchronization: Simulación evitando pérdidas de sincronismo.

En este caso si el parámetro ExternalSynchronization tiene valor positivo, indica que el Servidor Temporal tiene que esperar a que todos los subsistemas registrados hayan realizado sus tareas.

Esta posibilidad permite subsanar la deficiencia antes comentada en el modo básico de operación e introduce nuevos elementos en la simulación. Ahora:

- El Servidor Temporal genera un **TIC**.
- El subsistema cliente recibe el mismo, y comienza a ejecutar todas sus tareas programadas para ese instante temporal.
- Cuando termina todas las tareas, envía un mensaje de conformidad al Servidor Temporal, para avisarle de que ya terminó su operación.

El control es más complejo, como se muestra en la Figura 118, si bien el sistema es mucho más robusto. Se introduce, sin embargo, un nuevo problema, que son los bloqueos

del simulador: si un cliente cae, o pierde conexión con el servidor temporal por alguna razón, la simulación queda bloqueada. En este caso, la solución es el parámetro **MaxCicles**, que define el número de ciclos máximo que el servidor temporal espera conformidad de todos los subsistemas. Completado el máximo, se pueden tomar diversas acciones, si bien la más sencilla de plantear es expulsar del servidor temporal a todos los subsistemas que no enviaron conformidad en el tiempo establecido.

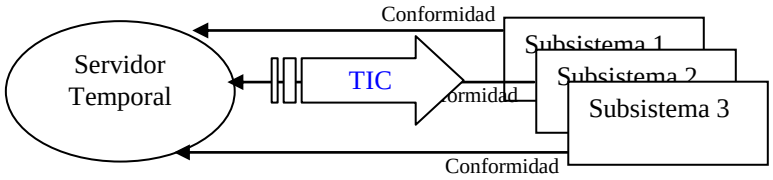


Figura 118: Sincronización con retroalimentación de subsistemas.

Wait: Optimización del Tiempo de ciclo: parámetro

En el planteamiento anterior, el Servidor Temporal tenía que esperar a que todos los Subsistemas enviaran la conformidad, lo que debería ocurrir cada "tCicle" segundos, es decir, cada ciclo. Pero puede que en un determinado ciclo todos los subsistemas terminen y manden la conformidad antes de que se complete el "tCicle", con lo que se estaría desperdiciando tiempo.

El parámetro Wait, cuando está a falso (interprétese como "no esperar"), permite optimizar este punto: cada vez que el Servidor Temporal recibe una conformidad de un Subsistema, comprueba si ya las tiene todas las conformidades, en vez de hacerlo sólo una vez cada tCicle. De esta manera, tan pronto tenga todas las conformidades, avanzará el ciclo, tal cual se muestra en la Figura 119.

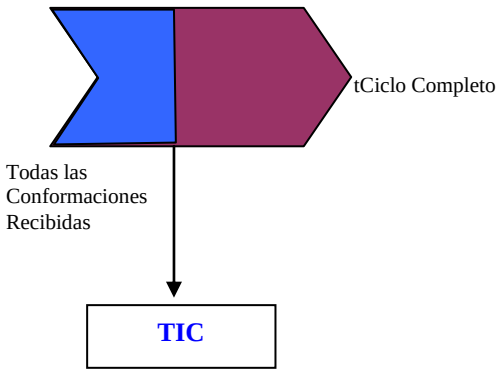


Figura 119: Envío de TIC tan pronto como subsistemas estén sincronizados

En cuanto a los parámetros dinámicos gestionados por el Lazo de control de la simulación, nos encontramos con los mostrados en la Tabla 26:

Parámetro	Abreviación	Descripción	Información extra
Instante de Simulación	*SimulationCount*	Es el tiempo actual en que se encuentra la simulación.	Es la suma de todos los ciclos que han pasado por el tCicle de los mismos.
Estado de Simulación	*stateSimulation*	Indica si la simulación está corriendo o parada.	

Tabla 26: Parámetros dinámicos durante operación del Servidor Temporal.

9.3.- Sincronización Temporal

En el proceso de sincronización temporal operan dos protocolos que se ejecutan en dos hilos independientes:

- El primero y fundamental es el **Protocolo de Registro / Desregistro de Subsistemas**: el Servidor Temporal necesita conocer qué subsistemas tienen que sincronizarse - y por tanto - a cuáles tiene que enviar los **TIC** temporales y de cuáles debe esperar recibir respuesta.

- El **Protocolo de Sincronización** que ejecuta el hilo principal y es el que gestiona (avanza, para, etc.) el tiempo de la simulación y hace llegar a todos los Subsistemas registrados el TIC o instante temporal.

    - El **Protocolo de Contingencias** está dentro del anterior, y solventa algunas situaciones que pudieran bloquear la simulación.

Veamos los distintos protocolos a través el siguiente diagrama de Flujo en la Figura 120, que resume la Operación del Servidor Temporal:

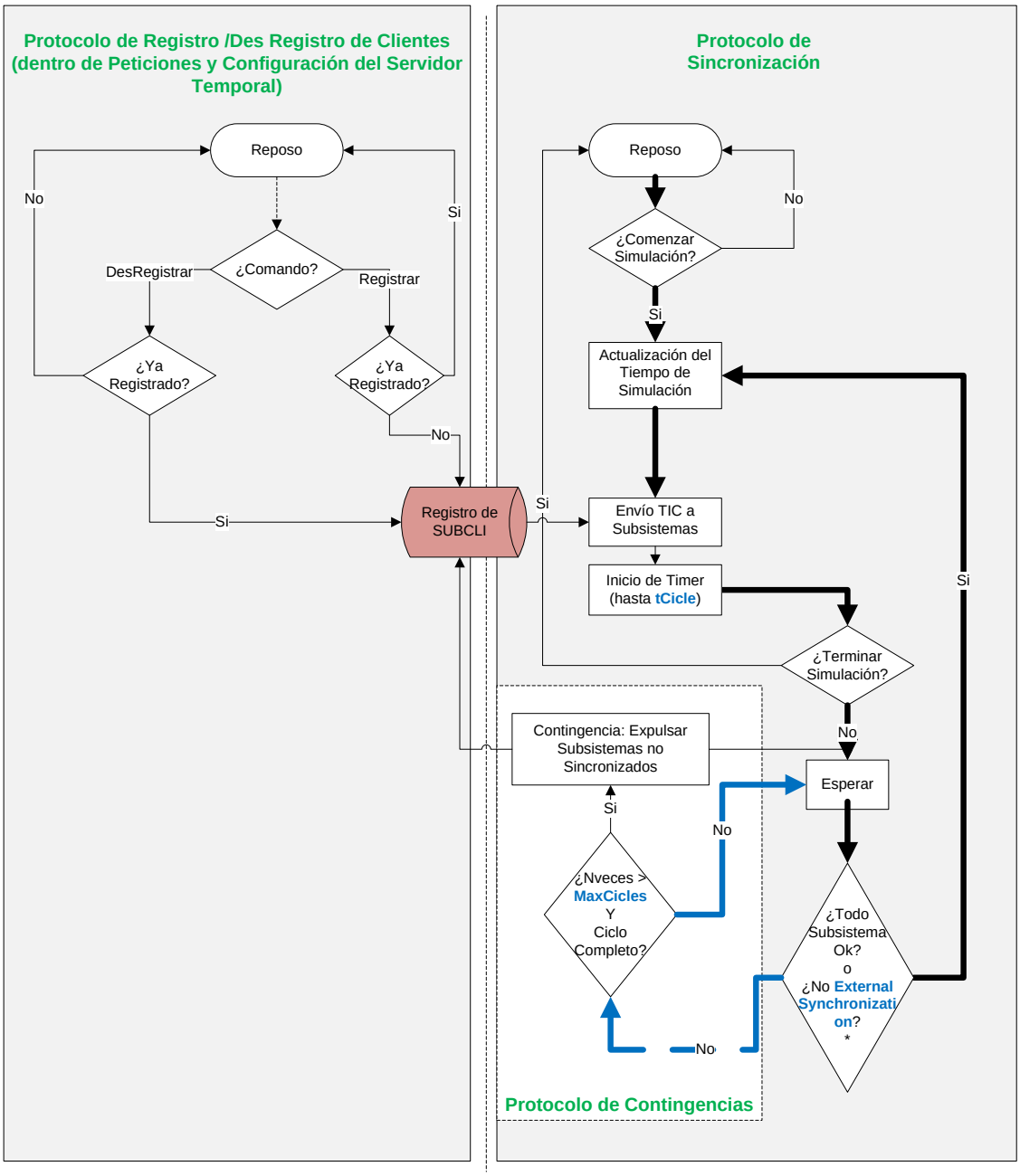


Figura 120: Diagrama de flujo de los distintos protocolos del Servidor Temporal.

El nexo de unión entre ambos protocolos principales, el de Registro/Desregistro (dar de baja) y el de Sincronización, es la base de datos que contiene el **Registro de Subsistemas Cliente (o Clientes Externos).** Sólo serán sincronizados los que estén en esta lista, por lo que el Protocolo de Registro culmina y trabaja directamente sobre esta base de datos.

En negrita se muestra el bucle principal de ejecución del simulador. En negrita y azul se muestra un camino alternativo y opcional, usado en la resolución de contingencias.

Protocolo de Registro / DesRegistro o baja de Clientes Externos

Éste se ejecuta sobre el hilo de Peticiones y Configuración del Servidor Temporal, y corre en paralelo al hilo principal de simulación para permitir que los subsistemas se puedan registrar, incluso con una simulación en marcha. Como veremos más adelante, no es más que un caso concreto que se ha querido realzar de las Peticiones y Configuración del Servidor del Temporal, que se verá en el Apartado **9.4**.

Simplemente el Servidor Temporal recibe una petición (**PET**) con la petición de Registro. Siempre y cuando ya no esté registrado, es registrado el cliente, lo que implica que el subsistema recibirá en cada ciclo del simulador el mensaje tipo **TIC**. Paralelo es el caso de desregistrar o dar de baja un cliente: el Servidor Temporal recibe una petición (**PET**) de desregistro, con lo que, siempre y cuando esté registrado, es desregistrado del Servidor Temporal, lo que equivale a no recibir más mensajes de tipo **TIC**. En ambos casos, el subsistema origen de la petición es notificado del resultado de su petición: verdadero o falso, según haya podido realizarse o no la petición.

Protocolo de Sincronización

Se ejecuta sobre el hilo de Lazo de Control de Simulación. Vamos a resumir el diagrama de flujo que muestra su proceso:

- Para comenzar la simulación, tiene que llegar un comando o petición de "comienzo de simulación". De forma simétrica, para detener la simulación hace falta un mensaje de tipo "Finalizar Simulación".
- Desde el momento que comienza el lazo de control de la simulación, que en forma de bucle repite:

  1) Se actualiza el tiempo de la simulación

  $$T_{nuevo} = T_{anterior} + \mathbf{tSimulado}$$

  2) Se recorre todo el **Registro de Clientes Externos**, y para cada uno se envía un mensaje de tipo TIC, con el T_{nuevo}.

3) Se inicia un timer, que se parará cuando se completen "**tCicle**" segundos de tiempo real.

4) Si no ha llegado mensaje de "Finalizar Simulación", se continúa. Si ha llegado, se vuelve al estado de reposo inicial.

5) *"Esperar"*: en este estado el hilo que ejecuta este protocolo se encuentra en reposo. No se trata de espera activa, por lo que es menor el consumo del proceso. Está a la espera de dos posibles eventos que pueden provocar el fin de la espera:

   - Si el parámetro **Wait** es verdadero, sólo se despertará del estado en espera cuando el timer configurado en el punto 3 se dispare.
   - Si el parámetro **Wait** es falso, se puede despertar por el timer, o porque un subsistema de la arquitectura ha enviado un mensaje de conformidad indicando que ha terminado de ejecutar todas sus tareas. Al obligar a despertar al hilo ante conformidades externas, hace que no haya que esperar **tCicle**, y por tanto se acelera la simulación tal como se pretende como **Wait**.

6) Tras despertar de la "Espera":

   - Si el parámetro **ExternalSynchronization** está a verdadero, entonces todo Subsistema debe haber enviado el mensaje de conformidad.
     - Si es así, se ha completado el ciclo, y por tanto se vuelve al punto inicial 1.
     - Si no es así, se va al punto 7.
   - Si el parámetro **ExternalSynchronization** está a falso, entonces, ya completado el ciclo, obligatoriamente se vuelve al punto inicial 1. [7]

[7] En el caso de que el Servidor Temporal sea un módulo dentro de un Servidor del Entorno, significa que es un servidor temporal dedicado al mismo. Por tanto, se da una salvedad en este punto, ya que aunque SincronizadoExterno esté a falso, y el ciclo esté completado, el Servidor del Entorno tiene que haber enviado adecuadamente la conformidad. En caso contrario se detendrá la simulación. Se detallará más en el Capítulo 13.

Protocolo de Contingencias

7) Si se llega a este punto es porque, aunque SincronizaExterno es verdadero, no todos los subsistemas han transmitido la conformidad por tanto:

   - Si el motivo de que despertara el hilo en el punto 5 fue el timer, esto significa que se acabó el ciclo.
     - Se contabiliza, de forma que si la cuenta supera **MaxCicles**, en definitiva, si el simulador ha esperado MaxCicles ciclos para que todos los subsistemas dieran la conformidad, y esto no se ha producido, entonces se resuelve la contingencia (punto 8). En caso contrario va al punto 5, en "Esperar".

8) Para resolver la contingencia, se analizan qué subsistemas no han dado su conformidad en el tiempo límite establecido, y son forzados a darse de baja del Servidor Temporal.

   Tras ello se vuelve al punto 5 para forzar a que en el siguiente ciclo sólo hayan quedado subsistemas con conformidad, y por tanto se pueda avanzar la simulación[8]. Se evitan así los bloqueos del simulador por los subsistemas externos.

9.4.- Peticiones y Configuración del Servidor Temporal.

Aparte de los protocolos y lazos de control explicados, se da un último proceso o lazo de control, que es una generalidad del "Protocolo de Registro / DesRegistro de Clientes" (de hecho son el mismo hilo): recibir y servir peticiones de clientes externos, bien para obtener información o para cambiar la configuración del Servidor Temporal.

En este caso el flujo de información es bastante simple, como se muestra en la Figura 121:

[8] Al igual que se aclaró en el punto 6, si se trata de un Servidor Temporal embebido en un Servidor del Entorno, y éste es uno de los subsistemas que no han enviado su conformidad, la simulación se detendrá, pues sin el Servidor del Entorno no puede avanzar la simulación.

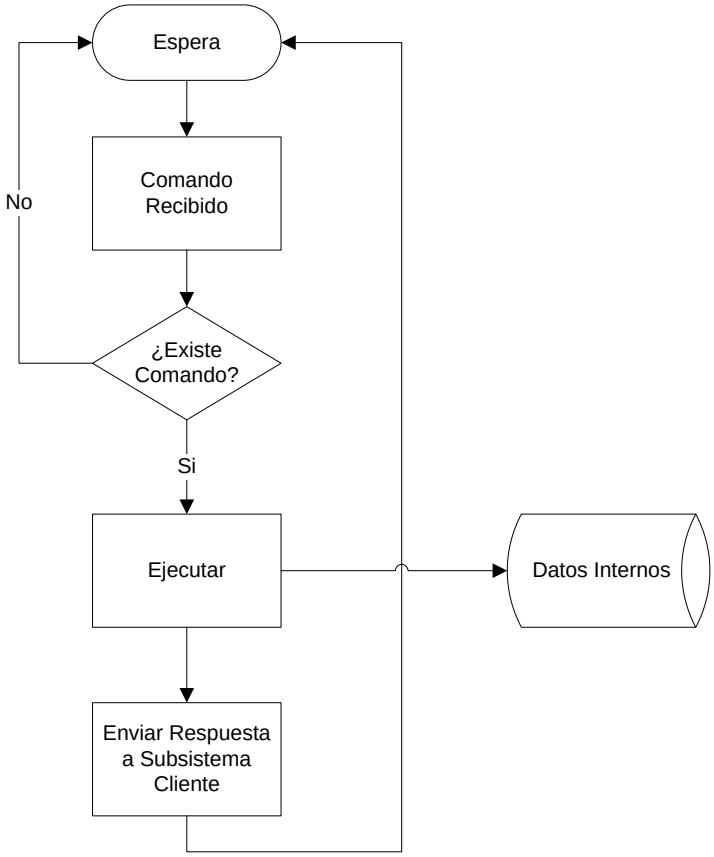


Figura 121: Flujo de flujo de ante peticiones al Servidor Temporal.

Éstos son los pasos del bucle de control:

0) Espera: este hilo queda en reposo hasta que llega una petición externa de un subsistema.

1) Se comprueba que existe. Si no se vuelve al punto 1.

2) Se ejecuta el comando. Este puede ser una simple petición de información, o puede ser un comando más potente que modifique lo que en el diagrama de flujo se llama "Datos Internos", que bien pueden ser parámetros de configuración o parámetros dinámicos.

3) Se envía la respuesta al subsistema cliente, y se vuelve al punto 1.

En el caso del Registro y DesRegistro, el diagrama de flujo expuesto en el "Protocolo de Registro / Desregistro de Clientes" sólo realza el proceso interno que acontece dentro del "Ejecutar", sustituyendo el resto por una flecha con puntos

suspensivos, siendo en realidad este hilo para Peticiones y Configuración del Servidor Temporal, como se muestra en la Figura 122:

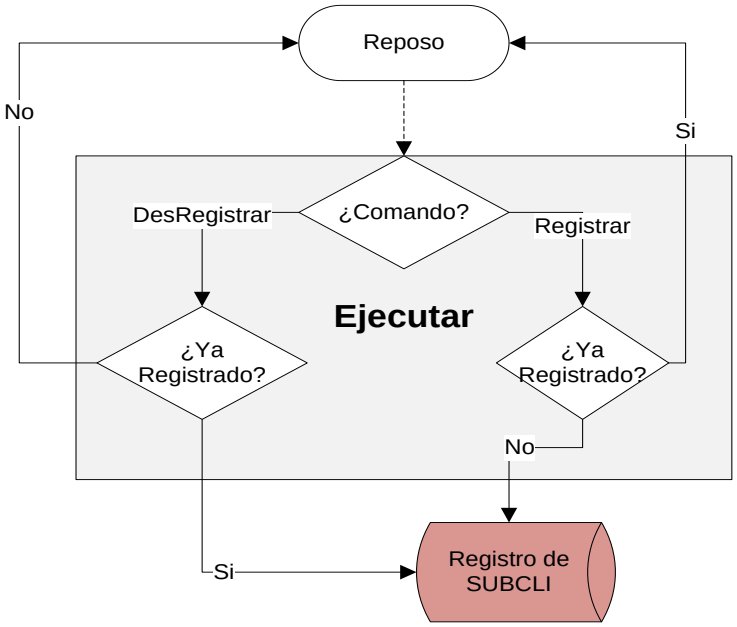


Figura 122: Diagrama de Flujo del protocolo de registro / desregistro del Servidor Temporal.

El catálogo de peticiones que admite de clientes externos son mostrados en la Tabla 27:

Origen	Servicio	Comando	Argumentos	Devuelve	Descripción
Servidor Temporal	GTemporal	startSimulation		Boolean	Comienza la simulación. Devuelve si se ha podido comenzar o no.
"	"	stopSimulation		Boolean	Para la simulación. Devuelve si se ha podido parar o no.
"	"	SimulationRunning		Boolean	Devuelve **Verdadero** si la simulación está corriendo en este momento.
"	"	AddExternalClient	Cliente	Boolean	Suscribe al cliente para que el Servidor Temporal le envíe TICs.

					Devuelve si se ha podido añadir o no al Cliente.
“	“	DelExternalClient	Cliente	Boolean	Similar, pero para desregistrar al Cliente.
“	“	getEnvParam	parámetro	String	Permite a un subsistema obtener el valor (en forma de String) de un parámetro de configuración (no dinámico) del Servidor Temporal.
“	“	setEnvParam	Parámetro Nuevo_valor	void	Simétrico al anterior, permite modificarlo.
“	“	ExternalReport	Cliente Tiempo	void	Petición mediante la cual un subsistema comunica al Servidor su conformidad indicando que en el presente ciclo ya concluido todas sus tareas, y que puede avanzar la simulación. Recibidos todos los InformesExternos la simulación puede avanzar un ciclo.

Tabla 27: Catálogo de peticiones que se pueden solicitar al Servidor Temporal.

Sólo resaltar que **setEnvParam** es un comando que permite configurar el Servidor Temporal, aunque esté la simulación ya en marcha. Esto implica versatilidad, como por ejemplo, redefinir el **tSimulated**, o modificar el **MaxCicles,** activar o desactivar **Wait** para acelerar o decelerar la simulación, etc. Esto se puede hacer en tiempo de ejecución de la simulación porque desacoplados el hilo que toma la petición de modificación de un

parámetro y el lazo de control de la simulación están completamente aunque están relacionados mediante la estructura de datos común. Al hacer modificación de un dato en la estructura de datos, automáticamente el lazo de control puede tomar estos datos modificados e incorporarlos a la simulación en curso.

CAPÍTULO 10.- *Servidor del Entorno*

Se explicarán en este capítulo las principales características del Servidor del Entorno, comenzando por una breve introducción para situar al lector en las funciones principales del servidor, para, posteriormente, ahondar en cada un de las piezas que forman el engranaje de este motor de la simulación y describir cómo ofrece servicios a clientes externos.

Adicionalmente, se describirá como se realiza la interacción con los clientes externos y se profundizará en otros aspectos del Servidor del Entorno, como configuración, etc.

Este capítulo es una síntesis del **Manual de Implementación del Servidor del Entorno**, que da una visión global y completa del mismo. Se recomienda revisar para ampliar los contenidos aquí presentados.

10.1.- Introducción

El *Servidor del Entorno* es el eje fundamental del presente Proyecto Final de Carrera, y puede ser apodado como el "Motor de la Simulación".

Se encarga de simular un entorno submarino virtual donde evolucionan agentes móviles equipados con sensores y actuadores. Como se adelantó en el **Apartado 9.1**, también puede funcionar como Servidor Temporal, con lo que tendría la misión de sincronizar a todos los subsistemas de la arquitectura ***SUBES***.

El resto de subsistemas de la arquitectura ***SUBES*** (que a partir de aquí serán llamados *Clientes Externos*) usarán los recursos del mismo, como puede verse en la Figura 123.

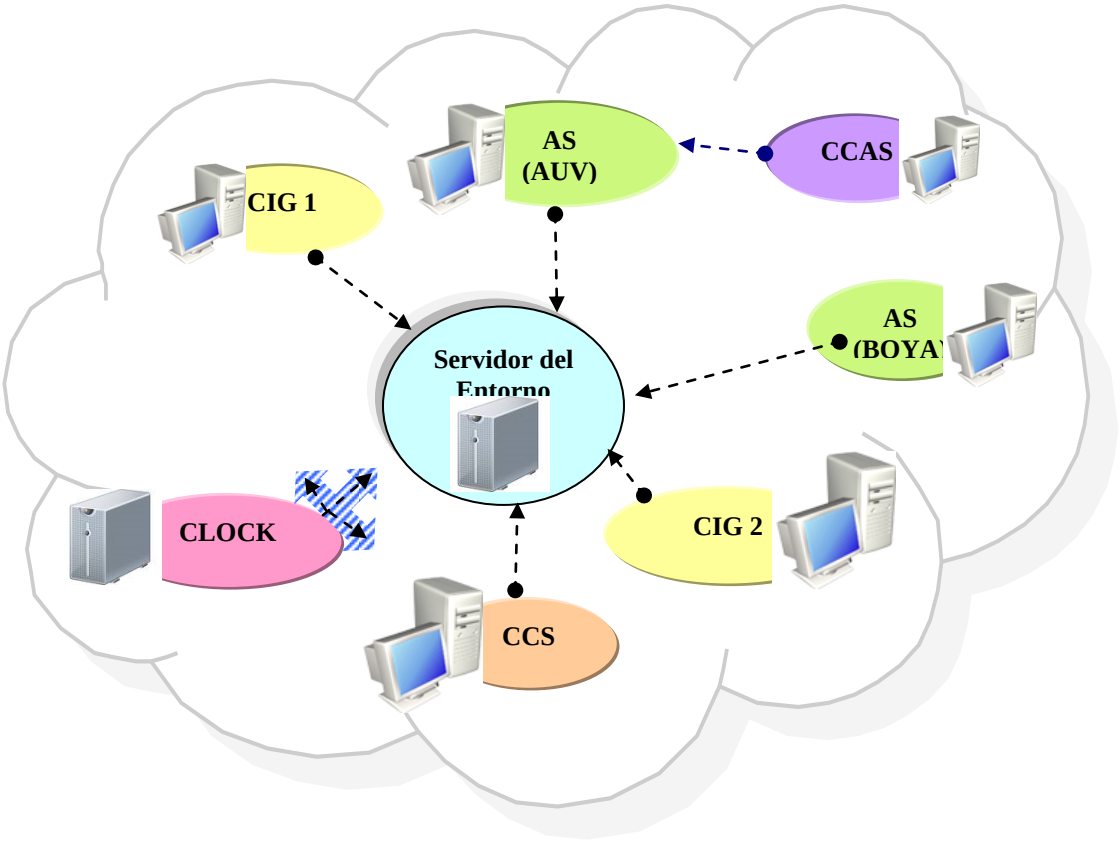


Figura 123: Subsistemas de la Arquitectura SUBES.

Para estos subsistemas de ***SUBES***, el servidor ofrece un catálogo de Servicios accesibles desde los Clientes Externos descritos en el **Apartado 8.3.2**. Cada servicio debe entenderse como un grupo de Comandos que, accesibles a través de peticiones, orientados todos a un determinado fin o recurso del servidor. Cada comando realiza una acción concreta sobre el servidor o piden una respuesta concreta del mismo.

Por ejemplo: todos los comandos relacionados con el modelo batimétrico pertenecen al *Servicio Batimétrico*. Un comando concreto dentro del servicio podría ser *obtener la profundidad en una determinada latitud y longitud*, u *obtener la latitud y longitud máximos* contemplados en el mapa batimétrico.

Todos los servicios ofertados por el Servidor del Entorno se pueden clasificar según objetivo y tipo de subsistema de la arquitectura ***SUBES*** que puede acceder al mismo, en nueve categorías, como se muestra en la Tabla 28:

Simulación del Entorno	Gestión de Parámetros Fisicoquímicos.
	Gestión Batimétrica
	Gestión Temporal[9]
Agentes Móviles	Gestión de Sensores
	Gestión de Simulación
	Gestión de Dispositivos de Comunicación
Gestión del Servidor	Controlador Central
	Gestión de Logs
	Gestión de Comunicaciones Externas

Tabla 28: Grupos de Servicios del SE.

- **Simulación del Entorno**: Grupo de recursos destinados a representar y actualizar el entorno submarino en cada instante. Mantiene un conjunto de modelos de diversas magnitudes (salinidad, corrientes marinas, temperatura, conductividad, batimetría, etc.), los actualiza en cada ciclo de simulación, y suministra a los clientes del servidor que lo soliciten el valor de la magnitud en un determinado punto o rango.

  A los servicios ofrecidos por este grupo acceden principalmente otros comandos y servicios del propio simulador. Por ejemplo, el comando que debe actualizar la posición de los agentes simulados (dentro de los comandos de *Gestión de la Simulación*) accede a la batimetría para comprobar que no colisiona con el fondo (dentro de la *Gestión Batimétrica*).

  El servicio y comando que se encarguen de "tomar una muestra" de una magnitud del medio submarino como puede ser la Temperatura (de la categoría de *Gestión de Sensores*) harán uso de los comandos y servicios de la categoría *Gestión de Parámetros Fisicoquímicos* para conocer Temperatura en un punto submarino.

[9] Como ya se ha descrito en el Capítulo, este tipo de servicios son gestionados por el Servidor del Entorno sólo en el caso del que el Servidor Temporal esté incluido dentro del Servidor del Entorno y no de manera centralizada.

También pueden acceder ciertos Clientes Externos, como el **CIG** si la interfaz gráfica requiere - por ejemplo – de la batimetría para proporcionar una representación gráfica.

Si el Servidor Temporal está embebido en el Servidor del Entorno, tanto los Clientes Externos como los distintos módulos internos que conforman el Servidor del Entorno tendrán que sincronizarse.

- **Agentes Móviles**: ofrece servicios para simular la actividad de los vehículos simulados, así como de los subsistemas internos de los mismos (sean AUV, ROV, BOYAS, etc). Así, permiten simular la navegación, la evolución del estado, etc. (con comandos y servicios de la *Gestión de Simulación*), y sus sensores (*Gestión de Medidas*) y dispositivos de comunicación (*Gestión de Dispositivos de Comunicación*). Acceden principalmente a estos recursos los **Clientes Agentes Simulados (AS)**.

- **Gestión del Servidor**: usado para supervisar y coordinar al resto de servicios del servidor, es utilizado básicamente por los propios componentes del servidor , aunque puntualmente algún cliente externo puede tener acceso, normalmente de tipo **Cliente Configurador del Servidor (CCS)**, para comenzar la simulación, pararla, cambiar el valor de los parámetros globales del sistema, etc.

  Ofrecen servicios para auditoria y depuración del sistema (*Gestión de Logs*), para gestionar las peticiones externas y sus respuestas (*Controlador Central*) y para la comunicación con Clientes Externos (*Gestión de Comunicaciones Externas*).

10.2.- Servidor del Entorno: organización modular en componentes.

Siguiendo la clasificación de servicios, el servidor se va a segmentar en nueve componentes **Gestores** o módulos(que se pueden ver en la Figura 125)**,** cada uno focalizado en la gestión de uno o varios servicios de cada categoría, y cada servicio con su propio elenco de comandos, como se muestra en la Figura 124.

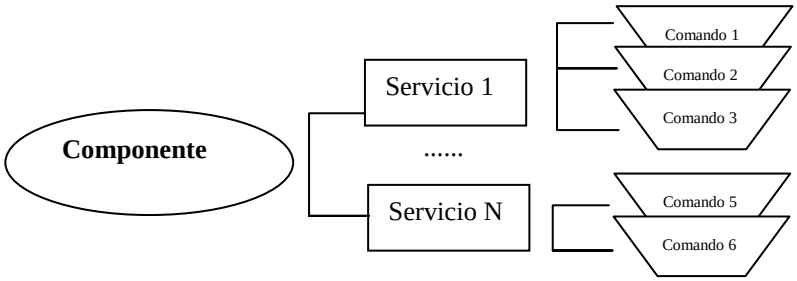


Figura 124: Organización de cada componente Gestor.

La división lógica permite, partiendo de una base común que comparten todos los componentes (que se explicará en la **sección 10.2.2.**), especializar a cada uno en función de las necesidades generadas por los servicios que ofrece. Facilita el mantenimiento y escalabilidad del servidor.

A niveles operativos, como veremos en profundidad más adelante, permitirá optimizar los tiempos de respuesta: siendo la carga computacional de cada tipo de servicio diferente en cada categoría, la división permite separar en hilos diferentes aquellos servicios de tipo más pesados, no siendo un cuello de botella para el resto de servicios.

En la Figura 125 se pueden observar los 9 componentes gestores en los que queda organizado lógicamente el Servidor del Entorno:

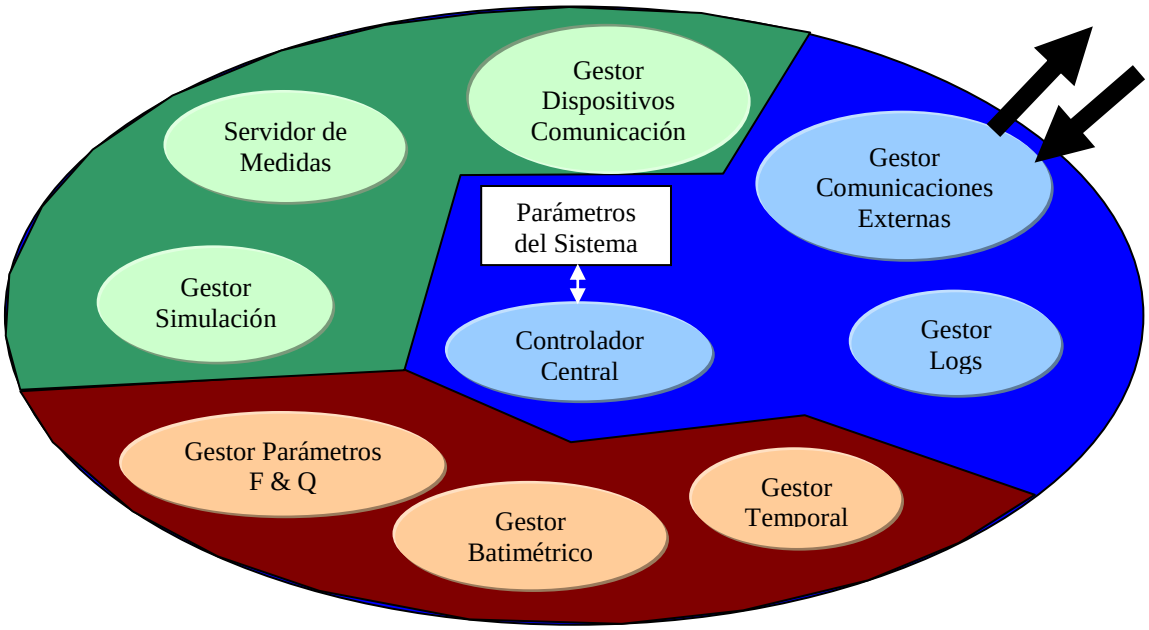


Figura 125: Componentes Gestores del Servidor del Entorno.

Componentes del Servidor del Entorno.

El componente **Gestor de Comunicaciones Externas** es el único punto de conexión del servidor que permite la comunicación del mismo con todos los Clientes Externos. Tiene función de "middleware", pues debe traducir los mensajes en un formato de serialización externa (XDR es el que se ha utilizado, acorde al **Apartado 8.4**) a los mensajes internos usados para comunicación entre componentes Gestores.

El **Controlador Central** gestiona el registro de todos los Clientes Externos - los cuales deben registrarse en el servidor mediante clave antes de usar sus recursos -, el enrutamiento de las peticiones externas llegadas a través del **Gestor de Comunicaciones Externas (GCE)** al componente Gestor correspondiente o la respuesta al GCE y la gestión de los parámetros globales del servidor.

El componente **Gestor de Logs** gestiona todos los datos de auditoria del sistema y su almacenamiento, no habiéndose implementado en el presente proyecto debido a su sencillez y porque no es un componente clave en el presente proyecto.

El componente **Gestor Batimétrico** gestiona el acceso a mapas batimétricos del sistema (en la implementación, mapas en formato netCDF), proporcionando comandos para acceder a información de los mismos.

El componente **Gestor de Parámetros Fisicoquímico** actualiza en cada ciclo de simulación los modelos de datos de magnitudes submarinas (como temperatura, salinidad, etc) y proporciona comandos para acceder al valor de las mismas.

El componente **Gestor Temporal** equivale al "Servidor Temporal" descrito en el **Capítulo 9**. De hecho, en el presente proyecto se ha optado por que este servidor se encuentre embebido en el Servidor del Entorno. Como se comentó en el mismo capítulo, este componente tiene la misión de registrar tanto a los Clientes Externos como a los componentes Gestores internos al servidor que necesitan sincronizarse temporalmente, y gestiona esa sincronización.

El componente **Gestor de la Simulación** registra a todos los Agentes Simulados externos (AS), y se encarga de actualizar su posición en cada ciclo de simulación, además de ofrecer comandos para gestionar la navegación de los AS.

El componente **Servidor de Medidas** gestiona el registro de los sensores de cada AS registrado a su vez en el sistema, y ofrece comandos para hacer mediciones usando los modelos de sensores asociados a los sensores registrados.

Finalmente, el componente **Servidor de Dispositivos de Comunicación** gestiona el registro de dispositivos de comunicación inter-agentes simulados de cada AS registrado en el sistema. También actualiza en cada ciclo de simulación el estado de los paquetes de datos enviados de un AS a otro subsistema.

En el presente proyecto no se ha implementado el soporte para simular misiones donde múltiples AUVs deban cooperar durante el transcurso de la misión, por considerarlo un objetivo que entraña un nivel más avanzado de complejidad.

10.2.1.- Protocolo de Comunicación entre componentes del servidor.

La comunicación entre los componentes Gestores del servidor se realizará mediante el intercambio de mensajes_de forma completamente distribuida: cualquier componente puede enviar un mensaje para petición o respuesta a cualquier otro.

Es por ello que cada componente tendrá un tampón o buffer de entrada en su interfaz en el que encolar los mensajes que deban ser procesados por el mismo. Incluso las respuestas devueltas a cada componente serán encoladas en el mismo buffer.

Comunicación basada en Mensajes:

Las unidades básicas de comunicación son los mensajes, que son encolados en un tampón o buffer de entrada que tiene la interfaz de cada componente Gestor, tal como se muestra en la Figura 126. Este esquema permite trabajar a los distintos componentes de forma independiente y desacoplada.

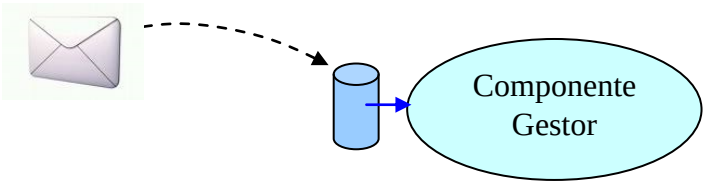


Figura 126: Comunicación mediante cola de mensajes en tampón de entrada.

En la implementación, los mensajes se corresponden con la clase *<MsgInterno.class>*, por lo que usaremos indistintamente los términos mensaje o MsgInterno.

Los mensajes tienen un identificador unívoco, y pueden ser de distinto tipo, según el objetivo del mensaje y tal como se muestra en la Tabla 29:

PETICION	RESPUESTA	INFORME
Para solicitar la ejecución de un comando perteneciente a cualquier servicio del catálogo y de cualquier componente Gestor.	Mensaje de respuesta relacionado con una petición.	Mensaje de respuesta relacionado con una petición. Sólo se genera cuando la petición original no tiene respuesta Así, el componente origen puede saber en que momento su acción fue realizada.
FINALIZE_THREAD	**TIME_CLOCK**	**REFRESH_DATA**
Mensaje de finalización. Indica al componente Gestor que debe finalizar su operación. Tiene la prioridad máxima frente al resto de mensajes.	Mensaje para indicar a los componentes que acaba de comenzar un nuevo tiempo de ciclo, y además enviarles en los parámetros adjuntos en que ciclo de ejecución se encuentra la simulación.	Mensaje que avisa a todo componente Gestor interesado cuando un parámetro global del sistema cambia de valor.

Tabla 29: Tipos de Mensajes entre componentes Gestores.

Dentro contendrán el Servicio, Comando y parámetros asociados al mensaje (parámetros de entrada, valores de respuesta, tiempo, etc).

Reglas de Comunicación:

Dentro del Servidor del Entorno, la comunicación es completamente distribuida. Cada componente Gestor puede enviar peticiones a cualquier otro (comunicación muchos a muchos) siguiendo el conjunto de normas que se presentan a continuación y que garantizan el éxito y robustez de la comunicación:

- *Gestión de Peticiones de un* componente *Gestor a otro dentro del Servidor:*

  "Todo mensaje que no sea RESPUESTA o INFORME que llega a un componente *Gestor, genera el envío de un mensaje a modo de respuesta al origen del mensaje".*

  En mensajes que no generan respuesta de por sí, se envía un INFORME, lo que permite que el origen siempre sepa cuando sus peticiones son tratadas.

- *Conectividad entre componentes Gestores del Servidor*:

"Los servicios deben ser unidireccionales, tanto directa como indirectamente".

Es decir, no puede suceder que:

- Para ejecutar un comando de un servicio S1 perteneciente a un componente CG1 solicite un comando del servicio S2 a un CG2
- Para resolver ese comando enviado, a su vez, el comando de S2 invoque un comando de S1.
- Es decir, un bucle de mutua petición (S1 a S2 y S2 a S1).

Esto pudiera producir un interbloqueo o **deadlock,** ya que el hilo de CG1 está bloqueado esperando respuesta de CG2.

Protocolo ante Peticiones provenientes de Clientes Externos

Cuando llega una petición desde cualquier cliente externo de los descritos en el **Capítulo 8** en formato XDR, el proceso que se desencadena en el servidor es el mostrado en la Figura 127:

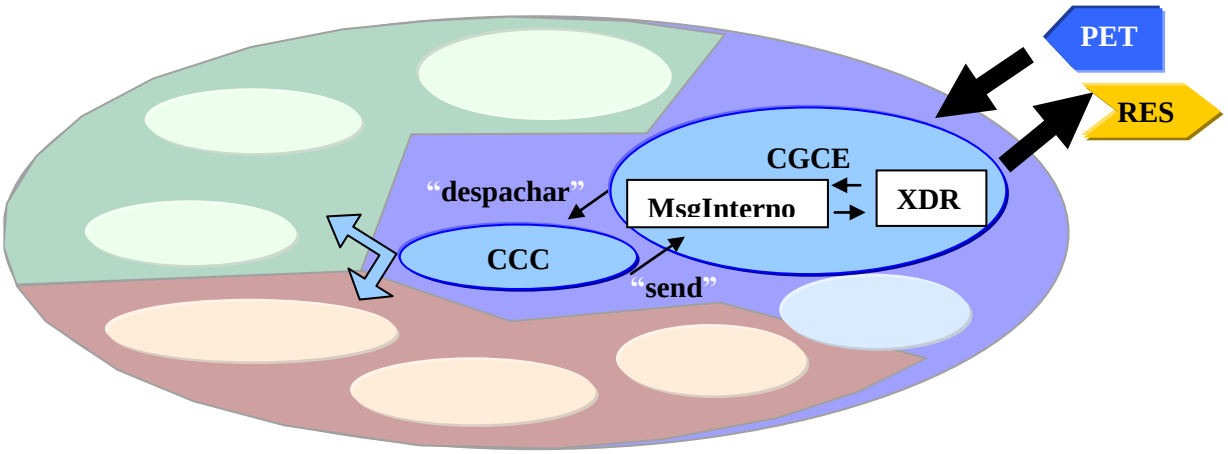


Figura 127:Traza de comunicación con Clientes Externos

Llegada la petición al componente Gestor de Comunicaciones Externas (CGCE), es traducido a un mensaje interno, para ser enviado al componente Controlador Central (CCC) que lo despachará, siempre y cuando esté registrado el cliente externo que hace la petición. Acto seguido distribuirá el mensaje al componente Gestor que lo tenga que tratar y atender.

Cuando hay una respuesta, se produce el proceso inverso: del CCC al CGCE (**send**) y finalmente envío al cliente externo.

10.2.2.- Abstracción de los componentes Gestores del Servidor del Entorno

Todos los componentes son diseñados e implementados en base a una misma estructura abstracta, que a modo de plantilla permite:

- Simplificar el mantenimiento: trabaja sobre un grupo reducido de clases.
- Simplificar la implementación: replicar la plantilla y especializar.
- Facilitar una rápida y sencilla escalabilidad del sistema: tomar la plantilla y especializarla según el componente.

A continuación en la Figura 128 se muestra la estructura base de todo componente Gestor:

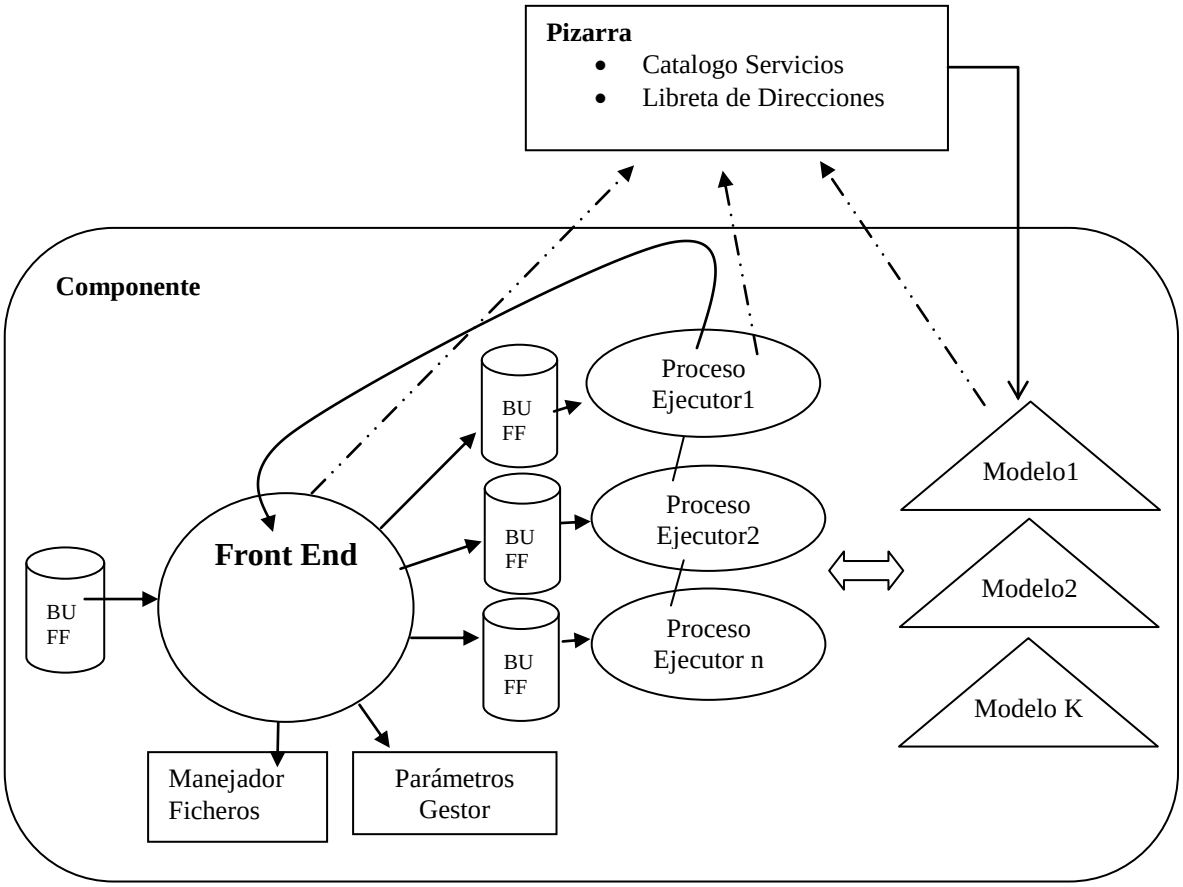


Figura 128: Organización abstracta interna de cada componente Gestor.

❖ Pizarra de Datos Compartidos

Se corresponde con la clase <*Pizarra.class*> implementada. Consiste en una estructura compartida por todos los componentes, que da información general sobre todos los servicios activos en el sistema y sobre cómo acceder a cualquier otro componente, ya que la comunicación es distribuida todos con todos. Se ha implementado mediante el patrón de diseño "Singleton", que imposibilita que haya más de una instancia en el Servidor del Entorno.

La Pizarra ofrece y comparte dos elementos con todos los componentes:

- **Catálogo de Servicios** (*clase <Catalogo.class>)*: es un directorio que indexa todos los servicios del servidor e información relevante de cada uno mediante una tabla Hash. Cada registro del catálogo representa a un servicio con la clase <*eltoCatalogo.class*> con una relación de todos los modelos usados por los comandos pertenecientes al Servicio.

- **Libreta de Direcciones** (*clase <LibretaDir.class>*): lista indexada que, para cada componente Gestor, tiene un registro de la clase <*eltoLibretaDir.class*> que lo asocia a un buffer de entrada, al que tiene que conectarse cualquier otro componente Gestor para comunicarse con el componente concreto.

❖ 

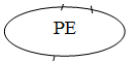
Buffer de Entrada

Se encarga de encolar los mensajes entrantes a un hilo de ejecución (que puede ser **Front End** del componente Gestor o **Proceso Ejecutor** y permanecerá dormido), así como notificar al correspondiente hilo la recepción de un nuevo mensaje, ante lo cual se despertará el mismo. Es, por tanto, la puerta de entrada de peticiones, respuestas, etc. a cualquier hilo en ejecución.

La política para extraer datos es principalmente FIFO, si bien:

- Siempre un mensaje de finalización (FINALIZE_THREAD) tiene mayor prioridad.

- Se puede solicitar al buffer una respuesta, con lo que sólo se sirven respuestas encoladas en la cola FIFO.

❖ 

Procesos Ejecutores:

Son el núcleo del componente Gestor, y la función principal es consumir mensajes encolados en su Buffer de Entrada (cada uno tiene el propio) y ejecutar los diversos comandos ofertados por el componente y asociados a los mensajes consumidos, dependiendo del tipo de mensaje y del comando que pretende ejecutar.

Cada componente Gestor tiene su propia clase Java que implementa todos los Procesos Ejecutores del componente (cada uno instancia de la misma). De hecho, veremos en el fichero de configuración del servidor que esta clase se carga en tiempo de ejecución.

Dependiendo del componente Gestor, el Proceso Ejecutor deberá cumplir ciertas interfaces (especificadas en el fichero de configuración del servidor) con los comandos de los distintos servicios que debe servir el componente.

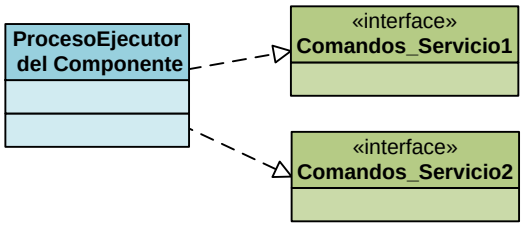


Figura 129: Diagrama UML del "ProcesoEjecutor".

La acción relacionada puede requerir ejecutar un comando secundario en cualquier otro componente Gestor, para lo que el Proceso Ejecutor hará uso de la Pizarra para obtener el buffer de entrada del front-end o interfaz del componente destino, y finalmente hacer el envío. Si se espera una respuesta, quedará a la espera.

Multiplicidad de Procesos Ejecutores en un mismo componente Gestor

Si bien todos los Procesos Ejecutores dentro de un componente son exactamente iguales (la misma clase), se puede especializar cada Proceso Ejecutor en la ejecución de comandos de un sólo Servicio.

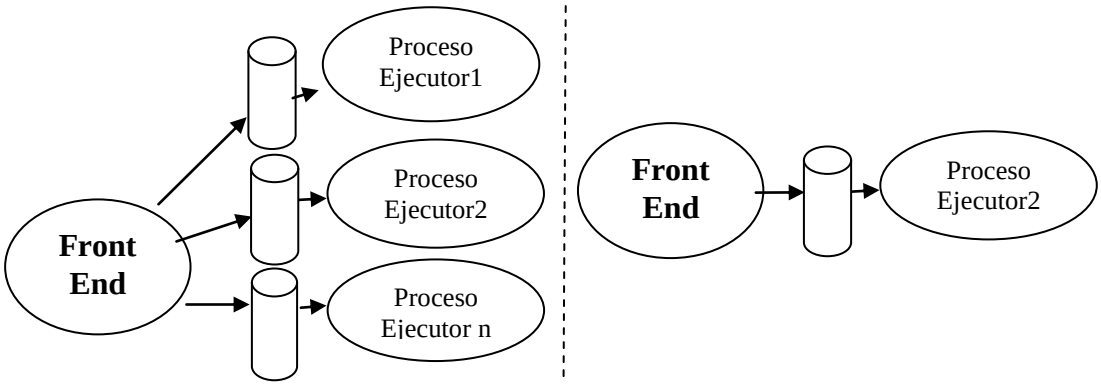


Figura 130: Ejemplo de multiplicidad de Procesos Ejecutores en Componentes.

Por tanto, la definición del componente Gestor es **completamente versátil,** como se muestran dos ejemplos en la Figura 130, y como veremos más adelante, en el fichero de configuración del servidor del entorno el usuario de la aplicación puede elegir:

- Cuántos Procesos Ejecutores tendrá el componente Gestor.
- La asociación de cada uno a un servicio / varios, o si el Proceso Ejecutor puede ejecutar cualquier comando de cualquier servicio.
- Incluso ciertos servicios pueden requerir la creación de un Proceso Ejecutor exclusivo (que en el proyecto es llamado "asterisco"), que al final de la ejecución del comando muera (para comandos pesados).

❖ 

Modelos

Los modelos de Datos son compartidos y utilizados por todos los Procesos Ejecutores dentro del componente Gestor. Todas las instancias cuelgan del **Catalogo** dentro de la **Pizarra**, para ser compartidas para todo el componente Gestor.

Jerárquicamente, en el fichero de configuración del servidor se especificará que estos modelos tienen que cumplir cierta interfaz de modelo (con las funciones a ofrecer), y de cada modelo se podrán instanciar varios modelos en el mismo fichero. Se muestra un ejemplo en la Figura 131.

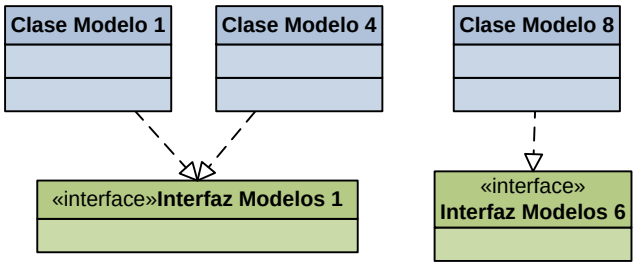
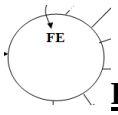


Figura 131: Definición de modelos, implementando su correspondiente interfaz.

Como veremos en el fichero de configuración del Servidor del Entorno, las interfaces van a dar mucha versatilidad al sistema: sin cambiar la misma Interfaz de Modelos, el usuario va a poder cambiar las Clases de Modelo en el fichero de configuración para dotar al sistema de nuevos métodos y recursos necesarios.



❖ Front-End

El Front-End es el hilo de ejecución de entrada al componente Gestor: tanto peticiones, como respuestas dirigidas al componente son encoladas en su buffer de entrada (única entrada del componente), para ser enrutados al Proceso Ejecutor correspondiente. En la implementación se corresponde con la clase <*FrontEnd.class*>.

Su principal misión es enrutar los mensajes dirigidos al componente Gestor al Proceso ejecutor correspondiente, en función de los servicios atendidos por cada Proceso Ejecutor y la carga de los mismos. También gestiona la creación de los mensajes de forma centralizada, controlando en el proceso la identificación unívoca de los mensajes: cada componente tiene un identificador único que se añade a la identificación del mensaje, más uno autonumérico, como se muestra en la Figura 132.

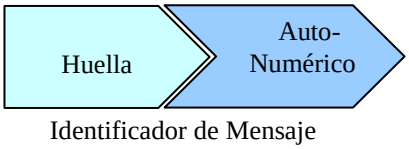


Figura 132: Formación de Identificador de Mensaje.

Se hace mediante el patrón de diseño de clonación: a partir de unos mensajes guardados de cada tipo, se clona y adapta, para finalmente devolverlo al Proceso Ejecutor que hace la petición al Front-End de crear un nuevo mensaje.

10.2.3.-Gestor de Comunicaciones Externas

Modelos de Datos Implementados

Interfaz de Modelos	Modelos Relacionados	Descripción
I_ExternalSerialization	M_XDRCommunication	Maneja el puerto del Servidor para el envío y recepción de datos con Clientes Externos. **M_XDRCommunication** es una implementación concreta, que realiza serialización de datos en XDR.

Tabla 30: Modelos de datos del Gestor de Comunicaciones Externas.

Véase la versatilidad que ofrece el sistema: el Proceso Ejecutor de este componente utiliza este modelo para enviar y recibir al y del exterior. Esto significa que, simplemente cambiando el modelo **M_XDRCommunication** por otro modelo (que también implemente la interfaz I_ExternalSerialization), podemos cambiar:

- El formato de serialización de datos de comunicación del Servidor con cualquier entidad Externa, eligiendo uno distinto de XDR.
- Manteniendo el mismo formato, cambiar los atributos u orden de los datos serializados.

De cara al servidor, hay dos funciones importantes del modelo.

- **Recibir mensajes de Clientes Externos:** hace una lectura de mensajes en XDR de un buffer asociado a un socket, evitando así que se bloquee el proceso durante la lectura. Traduce el mensaje a un mensaje interno que invoca el método **despachar** del **Controlador Central** para que enrute el mensaje al componente Gestor adecuado.
- **Enviar mensajes a Clientes Externos:** hace la traducción del formato de mensajes interno del servidor al formato XDR, para enviarlo finalmente al destino externo.

Servicios y Comandos disponibles

Servicio	Descripción	Clase Interfaz Relacionada	Comandos que se pueden Solicitar
"Send"	Comandos para el envío de mensaje hacia Clientes Externos del servidor	**Com_Send**	**Send**(MsgInterno msg)
"Read"	Si bien la recepción de mensajes externos es automática, estos comandos posibilitan conectar el servidor a un puerto de escucha.	**Com_Read**	**ConectServer**(int timeout, int ReviewerThreadLatency) **ConectServer**(int port, int timeout, int ReviewerThreadLatency)

Tabla 31: Servicios y Comandos del Gestor de Comunicaciones Externas.

Están implementados todos estos servicios en la clase de Proceso Ejecutor **PE_ComExternas2.** Como se observa en la Tabla 31, los comandos que ofrece el componente Gestor son **Send** para el envío de mensajes a Clientes Externos, y **ConectServer** que sirve para que el servidor abra un socket para posibilitar comunicación externa.

Para recibir mensajes, el componente Gestor utilizará un hilo especial con un Proceso Ejecutor al frente para permanecer a la escucha de los distintos sockets que abran los clientes. El resto de Procesos Ejecutores por lo general se encargarán del servicio "Send" con los comandos que conlleva.

Se recomienda leer el **Manual de Implementación del Servidor del Entorno** para ver los detalles de implementación y flujo de datos del componente Gestor.

10.2.4.-Controlador Central.

Modelos de Datos Implementados

Interfaz de Modelos	Modelos Relacionados	Descripción
I_ClientRegister	M_ClientRegister	Usado principalmente por el servicio "Register", mantiene actualizada información de todos los Clientes Externos que se han registrado en el servidor.
I_MessageRegister	M_MessageRegister	Usado básicamente por el servicio "InternalDispatcher", mantiene una lista de todos los mensajes que han sido recibidos de Clientes Externos para el posterior tratamiento de la respuesta.

Tabla 32: Modelos de datos del Controlador Central.

Estos modelos implementados en la Tabla 32 no tienen nada en especial: son listas que permiten añadir, eliminar y consultar información, como se muestra en la Figura 133.

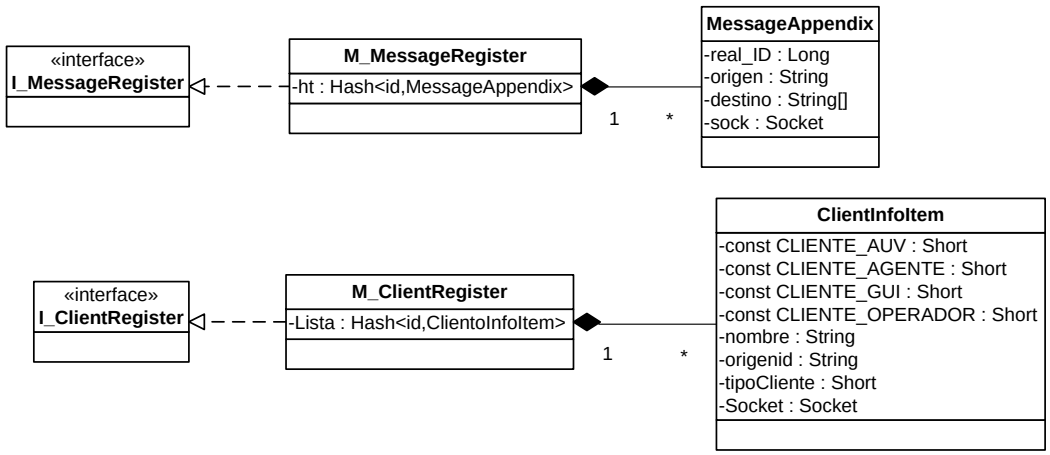


Figura 133: Diagrama UML de modelos de datos del Controlador Central.

En el caso de **M_ClientRegister**, por cada cliente se almacena un elemento de la clase del sistema **ClientInfoItem**, que contiene información sobre el Cliente Externo (nombre, tipo, socket, etc).

Los tipos de Clientes Externos que se pueden registrar son: CLIENTE_AUV (tipo **AS**), CLIENTE_AGENTE (boyas, ROV, animal marino, todos de tipo **AS**), CLIENTE_GUI (tipo **CIG**) o CLIENTE_OPERADOR (tipo **CCS**).

En el caso de **M_MessageRegister**, se guarda en una tabla información sobre la petición recibida de un cliente, para poder preparar después la respuesta a enviar al cliente tras obtener el resultado del servidor.

Servicios y Comandos disponibles

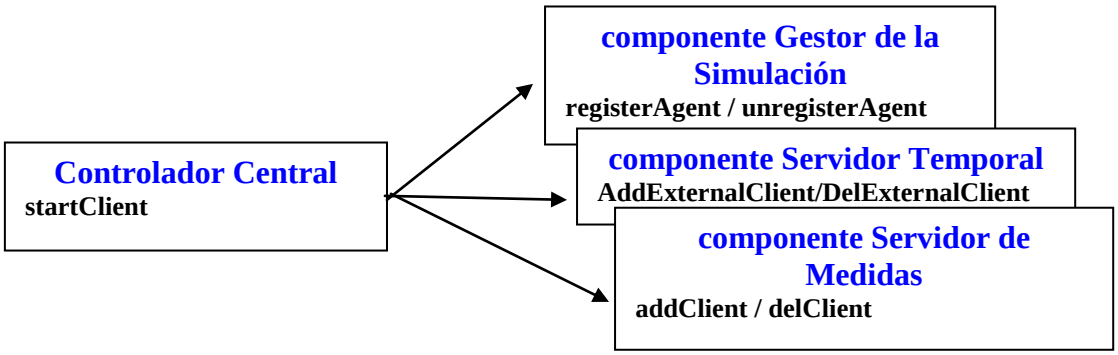
Servicio	Descripción	Clase Interfaz Relacionada	Comandos que se pueden Solicitar
"InternalDispatcher"	Comandos para gestionar los parámetros globales del Servidor del Entorno, apagarlo y para enrutar mensajes provenientes de Clientes Externos	**Com_InternalDispatcher**	**dispatch()** String **getEnvParm**(String variable) **setEnvParms (**Hashtable ht, String direction) **setEnvParm(**String variable, Object value**)** **closeServer()**
"ExternalDispatcher"	Permite gestionar algunos aspectos del componente desde un Cliente Externo.	**Com_ExternalDispatcher**	**ActivateAnswersCollector()** **ConsumeAnswer()**
"Register"	Gestiona el registro y desregistro de clientes en el Servidor del Entorno.	**Com_Register**	Boolean **startClient(** Short type, String name, String password) Boolean **finaliceClient(** String name) eltoInfoCliente **findClient(** String IdOrigin)

Tabla 33: Servicios y Comandos del Controlador Central.

Se trata de un componente Gestor no dependiente del tiempo (no tiene que sincronizarse con el Sevidor Temporal) e implementado en la clase heredera de ProcesoEjecutor **PE_CCentral.** Se recomienda un hilo ejecutor para cada servicio.

Servicio "Register":

Es bastante sencillo: sólo ofrece métodos para que un cliente externo pueda registrarse, eliminarse o buscar información de si mismo o de otro cliente del sistema. Destaca el comando **startClient** (su **simétrico es finaliceClient)** que registra el cliente en el **C.G. Controlador Central, y** automáticamente también registra al Cliente externos en los siguientes componentes Gestores del Servidor del Entorno: componente Gestor de la Simulación, componente Servidor Temporal, y componente Servidor de Medidas.



Servicio "InternalDispatcher"

El comando **apagarServidor** envía en broadcast a todos los componentes un mensaje de FINALIZE_THREAD, incluyéndose a sí mismo, con lo que el servidor se apaga.

Respecto al manejo de los parámetros globales del sistema, ofrece principalmente el comando **getEnvParm**, para obtener su valor. Todo componente Gestor que tome un parámetro global automáticamente se subscribe a este servicio, con lo que cualquier cambio externo de un parámetro (mediante **setEnvParm**) es notificado automáticamente (mediante mensajes de tipo **REFRESH_DATA**) a todos los subscriptores usando el patrón de diseño "Observador Observable".

El último punto importante del Controlador Central es el comando **dispatch**. Es invocado únicamente por el componente Gestor de Comunicaciones Externas. Cuando llega un mensaje de un Cliente Externo, éste se traduce a mensaje interno del servidor y se

envía al Controlador Central usando el comando **dispatch**. El **Controlador Central** tendrá que analizar el comando del mensaje y enrutarlo al componente Gestor concreto usando la Libreta de Direcciones.

Servicio "ExternalDispatcher"

Si bien en el servicio anterior el componente era capaz de "**dispatch**"**,** este servicio permite que, una vez se haya ejecutado el comando, y al Controlador Central sea devuelta la respuesta pedida por el cliente:

- El componente Controlador Central sea capaz de recibir respuestas, activándose como un Colector de Respuestas mediante **ActivateAnswersCollector**.
- Con cada respuesta recibida, usar **ConsumeAnswer** para empaquetarla y hacer la petición **send** al C. G. de Comunicaciones Externas.

10.2.5.-Gestor de Log

Modelos de Datos

Si bien no se ha implementado, el modelo de datos debería gestionar el almacenamiento (**añadir** y **recuperar**) de las entradas de log del sistema de forma segmentada: tipo de mensaje de log (error, warning, etc), código de mensaje, mensaje adicional del log, origen que genera el log, instante en que es generado (fecha y hora), etc.

Servicios y Comandos

No es un componente que dependa del tiempo (no necesita sincronización temporal). El servicio principal sería "Log"**,** donde se ofrezcan los comandos:

- **AddRegister**: invocado sólo por los componentes Gestores del Servidor del Entorno, añade una nueva entrada al log.
- **GetLog**: comando versátil, para obtener una porción del fichero de log según especificación de ciertos parámetros. Dado que el resultado puede ser bastante grande, el resultado podría generar un fichero que es enviado al cliente, en formato XML.

10.2.6.- Gestor Batimétrico

Modelos de Datos Implementados

Interfaz de Modelos	Modelos Relacionados	Descripción
I_DataVolume	M_DataVolume	Usado para representar la distribución de un parámetro bidimensional en un cierto volumen. **M_DataVolume** implementa este modelo haciendo uso como base de un fichero **NetCDF.**

Tabla 34: Modelos de datos del Gestor Batimétrico.

Se van a usar referencias al formato **netCDF** y cdl (metadatos del netCDF). Por ello es recomendado estudiar el **Apartado 6.3**, en la parte que se describe **netCDF**.

El modelo implementado en la Tabla 34 (**M_DataVolume**) proporciona información batimétrica mediante la lectura de un fichero netCDF. Véase que cambiando este modelo por otro que también cumpla la interfaz I_DataVolume se pueden implementar otros modelos no basados en netCDF (usando cualquiera de los métodos de representación de volúmenes de datos como los expuestos en el **Capítulo 6**).

Formato de Ficheros netCDF

El formato del fichero netCDF que puede ser interpretado por este modelo sigue estas directrices:

- 2 dimensiones: latitud y longitud
- 3 variables: Latitud, Longitud y Magnitud (bidimensional, por cada latitud / longitud da valor de magnitud).
- Cada variable debe poseer una serie de atributos obligatorios: unidades, escala, etc. Se utilizan nombres de atributos estándar, definidos por la convención de CORADS

Esta Figura 134 sería la declaración CDL del formato admitido (un ejemplo concreto, usando como magnitud la etiqueta ROSE, para batimetría).

```
netcdf DataVolume {
dimensions:
      longitude = ... ;
      latitude = ... ;
variables:
      float longitude(longitude) ;
            longitude:units = "UnidadDeLongitud" ;
            longitude:ipositive = 1 ;
      float latitude(latitude) ;
            latitude:units = "UnidadDeLatitud" ;
            latitude:ipositive = 1 ;
      short ROSE(latitude, longitude) ;
            ROSE:units = "meters" ;

// global attributes:
            :title ="Smith & Sandwell v. 8.2: 1/30-degree topography
and bathymetry";
```

Figura 134: Formato de fichero netCDF aceptado por el Modelo de Datos.

En este caso concreto (y en el presente proyecto) se está usando el fichero de batimetría de *Smith & Sandwell*, que es una batimetría y topología mundial con 1/30 grados de resolución.

*Modelo de datos **M_DataVolume***

En cuanto al modelo **M_DataVolume**, que es capaz de interpretar esta clase de ficheros, hay se puede destacar:

- Es un modelo general: en tanto y cuanto que permite leer magnitudes situadas en un eje de coordenada latitud /Longitud (no profundidad, por tanto, sólo magnitudes bidimensionales).
- Permite tomar una sección del fichero.
    - Para ello se especifica la latitud mínima, máxima y el incremento que se desea que tenga la latitud y longitud. Se muestra en la Figura 135

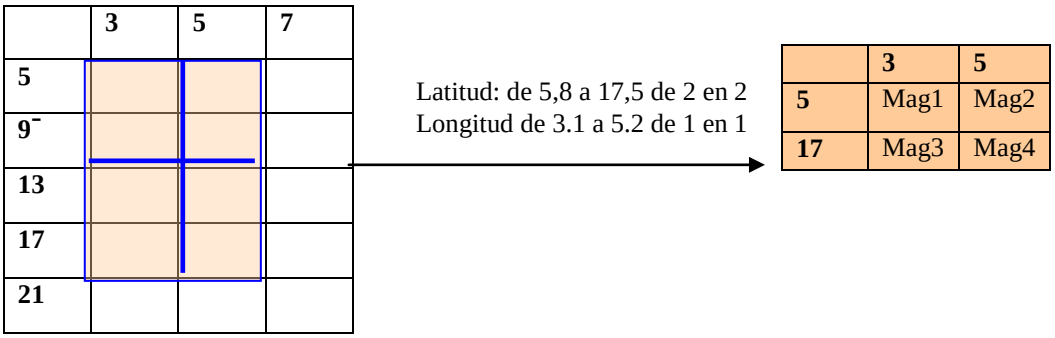


Figura 135: Porción obtenido de un fichero netCDF.

- También se puede obtener el valor de la magnitud en un punto concreto.

Servicios y Comandos disponibles

Servicio	Descripción	Clase Interfaz Relacionada	Comandos que se pueden Solicitar
"Bathymetry"	Ofrece comandos para poder solicitar el valor batimétrico en una determinada geo-localización submarina (o rango).	**Com_Bathymetric**	File **getBathymetricSlice** (Float Minlon Float Maxlon, Float LonInc, Float Minlat, Float Maxlat, Float LatInc) Muestra **getBathymetricValue** (Float lon, Float lat) Muestra **MaximunLatitude ()** Muestra **MaximunLongitude ()** Muestra **MaximunDepth** () Muestra **MiniumLatitude ()** Muestra **MinimunLongitude ()** Muestra **MiniumDepth ()**

Tabla 35: Servicios y Comandos del Gestor Batimétrico.

Está implementado en la clase heredera de ProcesoEjecutor **PE_Batimetrico**. No depende del tiempo.

La relación entre los comandos de los servicios del componente y su modelo de datos es bastante directa. La única diferencia es que en este caso los comandos sí que especifican una magnitud concreta: Profundidad o ROSE siguiendo nombre estándar.

Finalmente, mencionar que la implementación hecha contempla la integración de un único fichero batimétrico. Si bien, la arquitectura del Servidor del Entorno permite fácilmente que este componente pueda devolver la batimetría siguiendo varios ficheros batimétricos, incluyendo varios modelos en el fichero de configuración XML y mediante un algoritmo de integración de datos de varios ficheros (media, etc).

10.2.7- Gestor de Parámetros Fisicoquímicos

Modelos de Datos Implementados

Interfaz de Modelos	Modelos Relacionados	Descripción
I_FocusData	M_TempFocus	Se usa para representar la distribución de una determinada magnitud a lo largo de un medio submarino, basado en focos circulares con un valor umbral en el centro que tiende a un valor umbral en los extremos. **M_TempFocus** es una implementación concreta para una distribución de temperatura.
I_FunctionData	M_SalinityFunction	Se usa para representar la distribución de una determinada magnitud a lo largo de un medio submarino, basado en una función o ecuación que, dados ciertos parámetros de entrada, devuelve el valor. **M_SalinityFunction** es una implementación concreta para una distribución de salinidad, donde la salinidad depende de otras magnitudes o parámetros fisicoquímicos.

Tabla 36: Modelos de datos del Gestor de Parámetros Fisicoquímicos

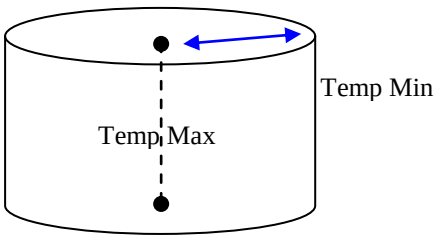
Este componente Gestor es un paradigma de la potencia que puede llegar a tener el Servidor del Entorno simplemente cambiando el Modelo de Datos en el fichero XML, en este caso, la distribución de un determinado parámetro en el medio marino.

El modelo de datos puede ser:

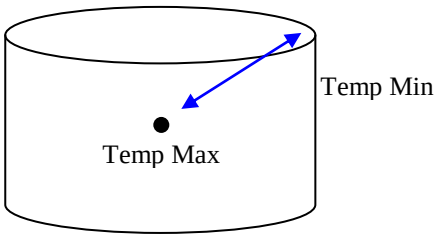
- **Fichero de Datos:** similar a M_DataVolume del **Apartado 10.2.6**, pero tridimensional (latitud, longitud y profundidad). Puede ser pesado, y no es ideal para magnitudes variables en el tiempo.
- **Ecuación**: una simple función nos puede dar el valor de la magnitud en una determinada latitud, longitud y profundidad.

  En este proyecto se ha implementado como puede verse en la Tabla 36 y a modo de ejemplo **M_TempFocus** (y en general la interfaz I_FocusData), para la magnitud de temperatura. Consiste en definir un cilindro, con un centro y un valor de la magnitud o temperatura en el centro, que en el caso de temperatura se puede entender como un "punto caliente", y finalmente el radio de acción, así como el valor de la magnitud fuera del radio de acción. El valor de la magnitud será proporcional a distancia al centro.

  Respecto a la profundidad hay varias alternativas: considerar que es independiente de la misma (sólo depende de distancia a eje central) o dependiente de profundidad (no todo el eje es "caliente", sino un punto concreto). En el modelo implementado se considera todo el eje "caliente".



Eje de Temperatura (independiente de profundidad)



Centro de Temperatura (dependiente de profundidad)

Estos modelos sí que se prestan más a magnitudes que sean dependientes del tiempo: basta con cambiar el valor de alguno de los atributos que configuran el modelo en función del tiempo.

En el ejemplo dado (**M_TempFocus**) bien podría ser la variación de la magnitud máxima en el centro en función del tiempo, en el caso de que el foco de temperatura se fuera extinguiendo con el tiempo.

Los modelos de tipo Ecuación también pueden obtener el valor de la magnitud representada en función de otras magnitudes del sistema, como es el caso de **M_SalinityFunction** en la Tabla 36, que depende de magnitudes como temperatura, presión y conductividad.

- **Autómata celular**: el modelo puede ser mucho más complejo que los modelos ya descritos, y representar – por ejemplo - un autómata celular:
  - Basta con tener un modelo de datos que represente una matriz.
  - También configurar el modelo, mediante variables como la vecindad, razón del cambio, peso de las celdas (en cuyo caso habría que definir otra matriz para representar los pesos).
  - En cada ciclo de simulación, el Proceso Ejecutor que maneja los modelos tendrá que actualizar la matriz según atributos.

  Es perfecto para representar magnitudes cuya distribución dependa del tiempo, del histórico de evolución de la magnitud, y de los valores que adoptan "la vecindad", como las "corrientes marinas".

  En el presente proyecto no se ha implementado si bien queda propuesto para desarrollos futuros.

Servicios y Comandos disponibles

Servicio	Descripción	Clase Interfaz Relacionada	Comandos que se pueden Solicitar
"Temperature"	Ofrece comandos para poder solicitar el valor de la temperatura en una determinada geo-localización submarina	**Com_Temperature**	MedidaSimple **getTemp**(Double lat, Double lon, Double depth) MedidaCompuesta **getTempFocuses ()**

“Salinity”	Ofrece comandos para poder solicitar el valor de la salinidad en una determinada geo-localización submarina	**Com_Salinity**	MedidaSimple **getSal**(Double lat, Double lon, Double depth)

Tabla 37: Servicios y Comandos del Gestor de Parámetros Fisicoquímicos.

Está implementado en la clase heredera de ProcesoEjecutor **PE_PFyQ** y si que tiene que sincronizarse con el componente Servidor Temporal. En cuando a la configuración del componente en sí, es recomendable que haya un Proceso Ejecutor dedicado a atender cada Servicio para un funcionamiento óptimo.

En el presente proyecto se suministran dos magnitudes significativas que se muestran en la Tabla 37, como son la Temperatura y Salinidad. Su implementación es ad-hoc, y no corresponden con modelos físicos reales. Sin embargo permiten probar los distintos servicios que pueda ofrecer este componente. Serán en futuros proyectos basados en los resultados del presente en los que se pueda desarrollar modelos reales.

Cada magnitud es un servicio, con sus propios modelos.

En cuanto al servicio de **Temperature**, se conforma a partir de un conjunto de focos. Cuando se invoca el comando **getTemp**, se comprueba la temperatura en función de la distancia a los distintos focos. Si el punto en el que se quiere medir la temperatura está dentro de 2 focos de acción, se suma la temperatura equivalente a la distancias a sendos centros, como se muestra en la Figura 136.

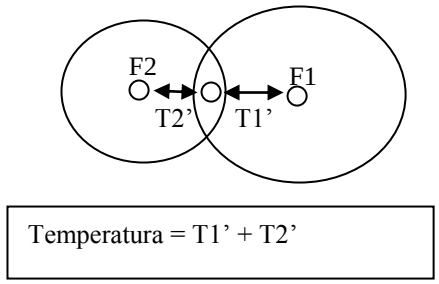


Figura 136: Focos de Temperatura colindantes.

El servicio de **Salinity** depende de otras magnitudes (como ya se ha descrito) para obtener su valor, por lo que genera peticiones al propio componente Gestor de Parámetros

Fisicoquímicos, cuyas respuestas (valores de otras magnitudes) son usadas por el modelo para obtener el valor final de salinidad.

10.2.8.-Gestor Temporal

Se recomienda leer el **Capítulo 9,** pues ésta no es más que una implementación concreta del Servidor Temporal descrito en ese capítulo.

Modelos de Datos Implementados

Interfaz de Modelos	Modelos Relacionados	Descripción
I_TimeRegister	M_TimeRegister	Gestiona los parámetros temporales del simulador y gestiona todo elemento del sistema que tenga que sincronizarse temporalmente. **M_TimeRegister** es una implementación concreta bastante sencilla

Tabla 38:Modelo de datos del Gestor Temporal.

El **M_TimeRegister** es el soporte que utiliza el gestor para poder llevar a cabo su operación. Por tanto, almacena:

- Una copia de todos los parámetros globales de configuración que fueron descritos en el **Apartado 9.2**. Tendrá que refrescarse al variar cualquiera de los parámetros.
- Información dinámica (tiempo de ciclo actual del simulador y estado del mismo, si está corriendo o parado).
- Gestión de dos clases de vectores con los:
    - **Registro de Clientes Externos,** nombrado en el **Capítulo 9**: Conjunto de Clientes Externos sincronizados temporalmente.
    - Se añade el **Registro de componentes Gestores**: todos y cada uno de los Componentes Gestores internos del Servidor del Entorno que deben sincronizarse.

Servicios y Comandos disponibles

Servicio	Descripción	Clase Interfaz Relacionada	Comandos que se pueden Solicitar
"Synchronization"	Comandos que permiten, por una parte la gestión de la simulación (comenzarla, finalizarla, saber estado, etc), y la gestión de los Clientes Externos subscritos a la sincronización.	**Com_TimeManager**	**ConfigureTimeManager ()** Boolean **startSimulation()** Boolean **stopSimulation()** **ChangeTime**() Boolean **SimulationRunning**() **AddExternalClient** (ClientInfoItem id) **DelExternalClient** (String id) **ExternalReport** (String ClientName, Integer Time) **BroadCast**()

Tabla 39: Servicios y Comandos del Gestor Temporal.

Los servicios y comandos son los mismos descritos en el **Apartado 9.4**, si bien vamos a ver cómo son implementados por la clase que define el componente Gestor, **PE_GestorTemporal**. Para una descripción más detalladas, se recomienda leer el **Manual de Implementación del Servidor del Entorno**.

Se proporcionan comandos para gestión de clientes registrados (**AddExternalClient** y **DelExternalClient**), para comenzar y finalizar la simulación (**start** y **stop/simulation**) y para comprobar el estado de la simulación **(SimulationRunning**).

El lazo de control contempla el envío de un mensaje de tipo TIME_CLOCK tanto a componentes Gestores del servidor como a Clientes Externos, y esperar la respuesta de los mismos (dependiendo de los parámetros globales, se esperará por los externos o no), aparte de atender las peticiones de comandos desde Clientes Externos, y resolver contingencias de Clientes Externos no sincronizados. Por ejemplo en la implementación hecha, se desregistran del servidor los clientes no sincronizados para que pueda avanzar el ciclo de simulación, con lo que ya no participan de la misma.

10.2.9.-Gestor de Simulación

Modelos de Datos Implementados

Interfaz de Modelos	Modelos Relacionados	Descripción
I_AgentRegister	M_AgentRegister	Usado en el servicio "Dynamic", hace una gestión integral del registro y des-registro de los agentes simulados en este componente. El modelo implementado **M_AgentRegister**.
I_Coordinates	M_SimpleCoordinates	Modelos usados en el servicio "Dynamic" para la predicción de posición de los agentes simulados. El modelo implementado es muy sencillo, haciendo una interpolación.

Tabla 40: Modelos de Datos del Gestor de Simulación.

El modelo **M_AgentRegister** gestiona todos los Agentes móviles que están siendo simulados en el servidor. Cada uno es representado mediante un modelo (**M_SimulatedAgent**) que almacena información dinámica del Agente móvil, como se muestra en la Figura 137.

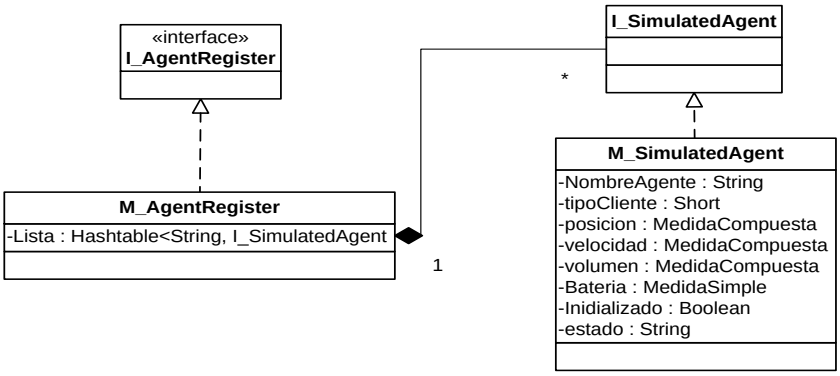


Figura 137: Diagrama UML del modelo de datos "M_AgentRegister".

Por otra parte, la interfaz **I_Coordinates** junto al **M_SimpleCoordinates** que lo implementa, proporciona el método para calcular la nueva posición de un agente simulado, en función de la vieja posición, la velocidad y el tiempo de ciclo.

Servicios y Comandos disponibles

Servicio	Descripción	Clase Interfaz Relacionada	Comandos que se pueden Solicitar
"Dynamic"	Conjunto de comandos para la gestión de los Clientes Externos que simulan la actividad de un Agente Simulado, entre lo que se encuentra registro, desregistro y modificación de la información dinámica (posición, velocidad, etc) del Agente.	**Com_Dynamic**	**ConfigureSimulationManager** () Boolean **registerAgent (** String name, Short type, MedidaCompuesta initialVel, MedidaCompuesta initialPos, MedidaCompuesta initialVol, MedidaSimple initialBat**)** Boolean **initAgent** (String name, MedidaCompuesta initialVel, MedidaCompuesta initialPos, MedidaCompuesta initialVol, MedidaSimple initialBat) Boolean **unregisterAgent** (

			String name) MedidaCompuesta **getPositionParameter (** String name) MedidaCompuesta **getVelocityParameter (** String name) MedidaCompuesta **getParametroVolumen(** **getVolumeParameter** String name) MedidaSimple **getBatteryParameter** (String name) Short **getAgentType** (String name) String **getState** (String name) Boolean **setPositionParameter** (String name, MedidaCompuesta value) Boolean **setVelocityParameter**(String name, MedidaCompuesta value) Boolean **setVolumeParameter**(String name, MedidaCompuesta value) Boolean **setBatteryParameter**(String name, MedidaSimple value) String[] **getAllAgents** ()

Tabla 41: Servicios y Comandos del Gestor de Simulación.

Todos los comandos de la Tabla 41 son implementados en la clase **PE_GSimulacion.** Necesita sincronizarse con el componente Servidor Temporal, y ofrece dos tipos de comandos dentro del mismo servicio:

- Comandos para registrar Agentes: **registerAgent** y **unregisterAgent**. Todos los Clientes Externos de tipo **AS** que pretendan simularse en el Servidor del Entorno deben estar registrados.

- Comandos para obtener y cambiar el valor de la Posición, Velocidad, Volumen (éste último no tiene por qué ser un parámetro dinámico, y sólo necesitará instanciarse inicialmente) y Batería. Es típico cambiar el vector de Velocidad desde un Cliente Externo para cambiar el plan de navegación del **AS**.

De la implementación, es relevante que en cada ciclo recorre los modelos de datos para actualizar la posición de todos los **AS** registrados.

10.2.10.-Servidor de Medidas

Modelos de Datos Implementados

Interfaz de Modelos	Modelos Relacionados	Descripción
I_SensorReg	M_SensorReg	Usado y gestionado por el servicio "**SensorReg**". Es la estructura de datos que permite mantener los sensores registrados de cada **AS**.
I_SensorTemp	M_SensorTemp1 M_SensorTemp2	Usados por el servicio "**Measurement**". Por cada tipo de sensor (especializado en medir una magnitud), debe existir una clase interfaz para todos los sensores del mismo. Cada interfaz a su vez puede tener varios modelos implementados, según se simulen sensores concretos para la misma magnitud (como es el caso de la temperatura, que hay dos sensores distintos simulados, M_SensorTemp1, y 2, que bien podrían ser dos sensores de temperatura de dos fabricantes distintos, para comparar).
I_SensorSal	M_SensorSal1	

Tabla 42: Modelos de datos del componente Servidor de Medidas.

De todos los componentes Gestores del presente Servidor del Entorno, es el que puede presentar mayor número de Modelos: esto es debido a que cada sensor distinto que se quiera simular debe tener un modelo asociado, como se ve en la Tabla 42 y Figura 138.

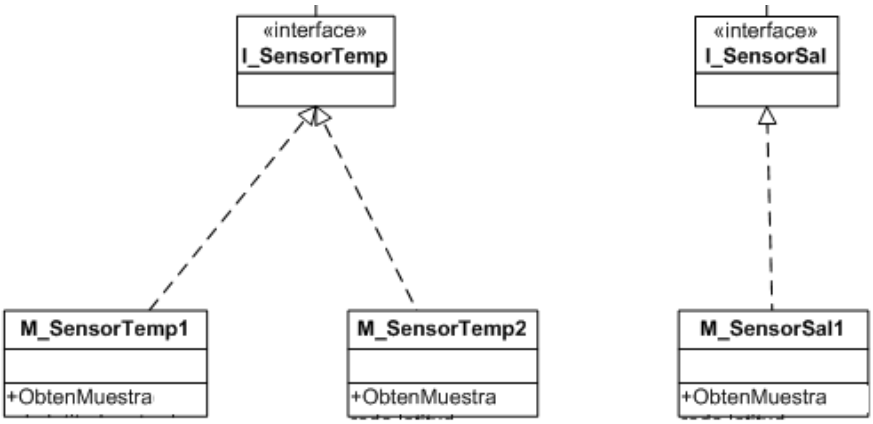


Figura 138: Diagrama UML de las distintas interfaces / modelos de cada sensor simulado.

En cuanto a la interfaz I_SensorReg y su modelo implementado M_SensorReg, consiste en una tabla que permite registrar para cada Cliente Externo de tipo **Agente Simulado** (**AS**) un conjunto de sensores (como se ve en la Figura 139), de los que durante la simulación podrá solicitar medidas.

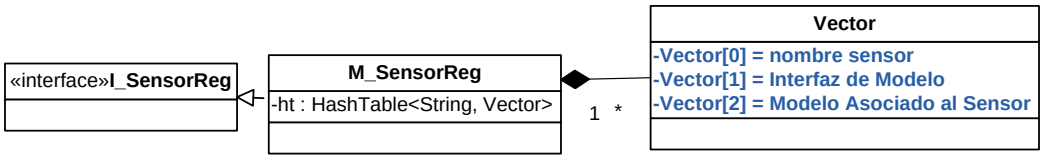


Figura 139: Diagrama UML del registro de sensores "M_SensorReg".

Como se observa en la Figura 139, el modelo es una tabla Hash que relaciona el nombre del **AS**, con un vector que contiene tantos elementos como sensores tenga registrado el AS en el Servidor de Medidas. Por cada sensor existe a su vez, un vector con información del modelo del sistema que representa el sensor.

Servicios y Comandos disponibles

Servicio	Descripción	Clase Interfaz Relacionada	Comandos que se pueden Solicitar
"SensorReg"	Comandos para gestionar el registro y desregistro de clientes externos y sus sensores, aparte de obtener información de cada sensor asociado a un cliente.	**Com_SensorReg**	Boolean **addClient** (String client) Boolean **delClient** (String client) Boolean **addSensor** (String client, String name, String service, String model) Boolean **delSensor** (String client, String name) String[]**getSensors** (String client) String[] **getSensorFeatures** (String client, String sensorName) String[] **getAllSensors** ()
"Measurement"	Ofrece comandos para realizar medidas de un sensor concreto.	**Com_Measurement**	Medida **measure** (String client, String sensorName)

Tabla 43: Servicios y Comandos del componente Servidor de Medidas.

El Proceso Ejecutor que implementa el componente es **PE_SMedidas** y no es dependiente del tiempo de la simulación. Habilita comandos para gestionar clientes y sus sensores (basado en el modelo de datos que implementa la interfaz I_RegSensor, **M_RegSensor**), todos agrupados en el servicio "**SensorReg"**. Será el **AS** externo el que tenga que registrar sus propios sensores

En cuanto al servicio "**Measurement"**, tiene un único comando que es **measure**, que permitirá especificar qué cliente desea medir y de cuál de sus sensores registrados. Para medir usará el modelo asociado al sensor especificado.

10.2.11.-Servidor de Dispositivos de Comunicación.

Modelos de Datos

No se ha implementado este componente Gestor, si bien presentará dos tipos de modelos de datos muy similares a los expuestos para el Servidor Temporal:

- Por una parte, tantas interfaces distintas como tipos de dispositivos de comunicación, y dentro de cada interfaz un grupo de modelos implementados para un dispositivo de comunicación concreto. Se muestra un ejemplo en la Figura 140

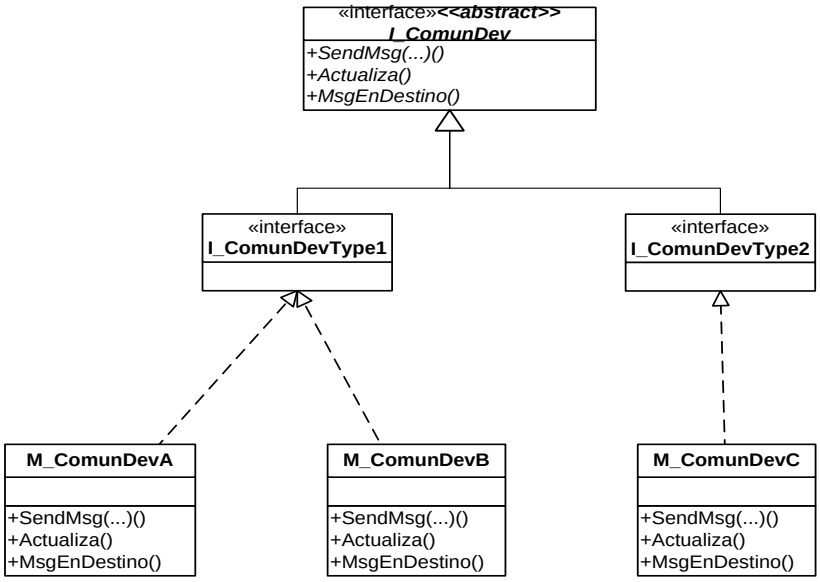


Figura 140: Diagrama UML de las interfaces / modelos de cada dispositivo de comunicación.

- Además, un registro que asocie a cada cliente, con sus dispositivos de comunicación, como está en la Figura 141. En este caso, para cada Dispositivo de Comunicación de cada Cliente Externo, almacena información dinámica: una cola de mensajes que está simulando, con el mensaje, destino, dirección en que se envía el mensaje y la posición en la que ahora mismo se "encuentra" el mensaje.

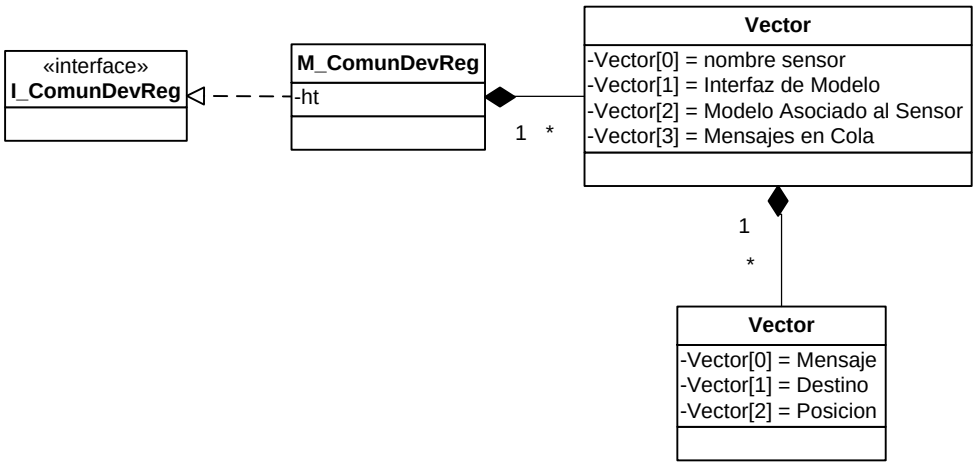


Figura 141: Diagrama UML de registro de dispositivos de comunicación.

Servicios y Comandos

Servicio	Descripción	Clase Interfaz Relacionada	Comandos que se pueden Solicitar
"ComunDevReg"	Comandos para gestionar todos y cada uno de los clientes y sus dispositivos de comunicación.	**Com_ComDev**	Boolean **addClient**(String client) Boolean **delClient**(String client) Boolean **addComunDisp** (String client, String name, String service, String model) Boolean **delComunDisp** (String client, String name) String[] **getAllComunDisps** (String client) String[] **getComunDispFeatures** (String client, String dispName) String[] **getAllComunDisp**()

"MsgSending"	Ofrece comandos para realizar medidas de un sensor concreto.	**Com_MsgSending**	Medida **Send_Msg** (String client, String ComunDisp, String msg, String destination)

Tabla 44: Servicios y Comandos del Servidor de Dispositivos de Comunicación.

Tendrá que ser un componente sincronizado con el Servidor Temporal, y por tanto actualizar en cada ciclo la posición de los mensajes encolados, y comprobar para cada uno si han llegado a su posición final de destino, en cuyo caso enviar al Cliente Externo usando comandos del componente Gestor de Comunicaciones Externas.

10.3.- Configuración y carga dinámica del Servidor del Entorno.

Una de las mayores ventajas del diseño del presente Servidor del Entorno es que es altamente configurable, siendo el usuario el que decide la estructura, los componentes Gestores, los hilos de ejecución de cada componente, etc). Hay dos posibilidades a la hora de configurar el Servidor del Entorno:

- La parametrización mediante la configuración de un fichero XML con toda la configuración del Servidor del Entorno.
- El uso de un conjunto de métodos que ofrece Java para permitir la reflexión: la carga de clases, instanciación de objetos de las mismas, y uso de sus métodos en tiempo de ejecución, en función de un fichero XML.

10.3.1.- Parametrización del Servidor del Entorno: configuracionServidor.xml

Este fichero se encuentra en la carpeta **Configuración** de la raíz del proyecto. La etiqueta raíz es <**Servidor**> y lo forman dos partes principales:

- <**ParametrosConfiguracion**>: se forma de conjunto de parámetros globales (<**param**>) del Servidor del Entorno, que serán usados por todos los componentes Gestores del mismo y gestionados por el componente **Controlador Central**. Éste

también se encarga de actualizar el fichero XML cuando se modifican los parámetros. Se muestran a continuación en la Figura 142.

```
<ParametrosConfiguracion>
   <Param nombre="ExternalSynchronization" valor="true" />
   <Param nombre="EcuatorialEarthRadius" valor="6378137" />
   <Param nombre="PolarEarthRadius" valor="6356752.3" />
   <Param nombre="port" valor="1983" />
   <Param nombre="timeout" valor="500" />
   <Param nombre="Wait" valor="false" />
   <Param nombre="tSimulated" valor="3000" />
   <Param nombre="pauseHRev" valor="20" />
   <Param nombre="password" valor="uno" />
   <Param nombre="MaxCicles" valor="6" />
   <Param nombre="tCicle" valor="3" />
   <Param nombre="host" valor="192.168.168.25\serverHome" />
</ParametrosConfiguracion>
```

Figura 142: Parámetros de Configuración del SE.

Los parámetros relacionados con la gestión temporal fueron descritos en el **Apartado 9.2**. El resto son usados en el resto de componentes Gestores del servidor.

- **Parametrización de cada componente Gestor del Servidor del Entorno**:

Cada componente Gestor tiene su propia etiqueta. Veamos como ejemplo el componente **Gestor de Parámetros Fisicoquímicos**. En el fichero de configuración de la Figura 143:

```
<GParamFisicoQuimicos>
    <Parametros>
      <direccion ruta="local" />
      <dirDefecto ruta="C:\Proyecto\" />
      <pEjecutor nombre="PE_PFyQ" />
      <temporal valor="si" />
    </Parametros>
    <Procesos>
      <Proceso nombre="p1" />
      <Proceso nombre="p2" />
    </Procesos>
    <Servicios>
      <Servicio nombre="Temperature" activo="true" proceso="p1" comandos="Com_Temperature">
        <InterfazModelos interfaz="I_FocusData">
          <Modelo fichero="M_TempFocus">
            <Parametro id="1" valor="28.027" />
            <Parametro id="2" valor="-16.788" />
            <Parametro id="3" valor="0" />
            <Parametro id="4" valor="20" />
            <Parametro id="5" valor="100000" />
            <Parametro id="6" valor="5" />
          </Modelo>
          <Modelo fichero="M_TempFocus">
        </InterfazModelos>
      </Servicio>
      <Servicio nombre="Salinity" activo="false" proceso="p2" comandos="Com_Salinity">
    </Servicios>
</GParamFisicoQuimicos>
```

Figura 143: Ejemplo de configuración del Gestor de Parámetros Fisicoquímicos.

- <**Parametros**>: los parámetros globales del componente, que son la dirección en caso de que esté en una máquina distinta a la del núcleo del servidor, **pEjecutor** que es la clase que implementa todos los Procesos Ejecutores del componente, y si es **temporal** o no.

- <**Procesos**>: Se definen los distintos Procesos Ejecutores que formarán parte del componente Gestor. Se les debe a cada uno asignar un nombre distinto.

  Aparte de los que aquí se especifiquen, siempre se creará un Proceso Ejecutor más que tendrá carácter global, y será el Proceso Ejecutor por defecto.

- <**Servicios**>: Se forma de una colección de elementos con la etiqueta **<Servicio>**, con todos y cada uno de los servicios que ofrece el componente Gestor. En el ejemplo mostrado, hay dos servicios: Temperatura y Salinidad.

  En cada <**Servicio**> se pueden especificar:

  - Comandos: clase interfaz Java que prototipa todos los comandos del servicio. Los Procesos Ejecutores tendrán que ejecutar todos los comandos de todos los servicios.

- El Proceso Ejecutor que atiende el servicio: * si debe ser un Proceso Ejecutor de tipo asterisco, u omitir si lo puede llevar a cabo cualquiera.
- Interfaces de Modelos (de 0 a muchas), modelos (dentro de cada interfaz, de 1 a muchos), y parámetros para instanciar cada modelo en el componente Gestor.

```
<InterfazModelos interfaz="I_FocusData">
  <Modelo fichero="M_TempFocus">
    <Parametro id="1" valor="28.027" />
    <Parametro id="2" valor="-16.788" />
    <Parametro id="3" valor="0" />
    <Parametro id="4" valor="20" />
    <Parametro id="5" valor="100000" />
    <Parametro id="6" valor="5" />
  </Modelo>
  <Modelo fichero="M_TempFocus">
</InterfazModelos>
```

Figura 144: Interfaz I_FocusData seguida de modelo M_TempFocus.

En el ejemplo de la Figura 144 se declara una única interfaz, que es **I_FocusData**, y para la misma dos instancias (ambas del modelo **M_TempFocus**, aunque pudieran ser distintas en líneas generales

Hay que resaltar la gran versatilidad del sistema: el usuario es libre de, en cualquier momento, cambiar en el fichero de configuración el modelo por una clase distinta, siempre que respete la Interfaz de Modelos Asociada, o cambiar la interfaz, añadir o quitar servicios y cambiar la interfaz Java que prototipa los comandos del servicio, además de cambiar la clase del Proceso Ejecutor para ampliar la funcionalidad.

El cambio es sencillo porque, tanto modelos, como interfaces de modelos, como interfaces de comandos, como servicios y procesos ejecutores se cargan en tiempo de ejecución.

Por otra parte, es versátil porque el usuario decide en el fichero de configuración cuantos procesos ejecutores hay y cómo se reparten los servicios del componente.

10.3.2.- Inicialización de los componentes Gestores mediante Reflexión

Para mayores detalles, se recomienda al lector leer el **Manual de Implementación del Servidor del Entorno,** documento anexo al proyecto.

Al crear una instancia cada componente Gestor durante el arranque del servidor, se utiliza la clase <*InstaladorGestor.class*>. Su objetivo principal es:

- Leer el segmento del fichero XML que se corresponde con el componente Gestor.
- Analizar la corrección léxica-sintáctica del fichero de configuración XML, usando en Java la librería JDOM para lectura de ficheros XML).
- A medida que analiza el fichero, instancia mediante reflexión el Front-End, los Procesos Ejecutores de la clase especificada en el fichero de configuración y tantos como declaró el usuario, y los modelos de la clase especificada que parametrizó el usuario en las cláusulas en el mismo fichero.

Para la reflexión se usa el **ClassLoader** de Java, que permite cargar una clase previamente compilada (*.class) durante la ejecución. Esto dota de gran versatilidad al sistema: cambiando el fichero XML para reflejar una nueva clase (de Proceso Ejecutor, Modelos, etc) se puede incorporar ésta al engranaje del Servidor del Entorno.

10.4.- Protocolo de Comunicación del Servidor del Entorno con Clientes Externos.

Cualquier subsistema de la Arquitectura ***SUBES*** que necesite ejecutar un **Comando** de cualquiera de los **Servicios** que ofrece el Servidor del Entorno, deberá hacerlo enviando un mensaje con el **Formato de Intercambio de Paquetes** en **XDR** descrito en el **Apartado 8.4**.

Unidades de Comunicación

En concreto, cualquier subsistema de la arquitectura ***SUBES*** sólo podrá enviar mensajes de tipo **PETICION**. (**PET**).

Tal como se describió en el **Apartado 8.4.3**, los mensajes de tipo **PET** son "Mensajes para TeleComandos" cuyo cuerpo es de tipo **CuerpoSolicitud**. En dicho cuerpo, el nombre del cuerpo debe ser el **Servicio** al que pertenece el comando a ejecutar en el servidor. En el vector de **parámetros**, el primer elemento debe ser el comando mientras el resto son parámetros de entrada al comando.

Del Servidor, pueden llegar al Cliente Externo dos tipos de mensaje:

- El cuerpo del mensaje de tipo **RES**  (como respuesta a **PET** anterior) tendrá un cuerpo de tipo **CuerpoInforme**, con el identificador de la solicitud a la que responde, y entre su vector de variables, un único elemento que es la respuesta.

- El cuerpo del mensaje de tipo **TIC**  tendrá un cuerpo de tipo **CuerpoTiempo**, con el tiempo de ciclo del simulador.

Protocolo de Comunicación

Descritas las unidades básicas de comunicación con el Servidor del Entorno, todo Cliente Externo que necesite ejecutar un comando de algún servicio del Servidor (sea **AS**, **CIG** o **CCS**), debe seguir el siguiente protocolo:

1) Cualquier Cliente con intenciones de usar los servicios y comandos del Servidor debe estar registrado:

   a. Abrir un **Socket** con el Servidor del Entorno.

   b. A través del mismo debe enviar una **PET**  **,** ejecutando el comando **startClient** del servicio **"Registro"** del servidor y con la clave correcta.

   c. Deberá quedar a la espera de la respuesta (a través del mismo socket) **RES**  , con el valor **verdadero** o **falso** según se haya conseguido el registro (continuar) o no (volver a 1).

2) El socket usado para el registro del cliente (**sR**), no debe ser usado para enviar nuevas **PET** al Servidor del Entorno.

a. El motivo es que ese socket será usado para la sincronización con el Servidor del Entorno. El cliente deberá quedar a la espera del socket de recibir un mensaje de tipo **TIC**  , ante lo cual ejecutar todas las acciones del ciclo,

b. Tras recibir dicho mensaje, ejecutar finalmente el comando **InformeExterno** por el mismo socket para indicar al servidor que finalizó, y volver a a).

Todo cliente que sea de tipo CLIENTE_AUV, CLIENTE_AGENTE, CLIENTE_CIG deberá sincronizarse o si no será expulsado del servidor.

3) Para enviar cualquier **PET**  al Servidor del Entorno, el Cliente Externo debe tener habilitado al menos otro socket abierto con el Servidor (**sP**), distinto del socket de registro. También es posible abrir tantos sockets como se deseen con el servidor para enviar peticiones.

a) En función del tipo de petición hecha, el cliente deberá quedarse esperando en el Socket abierto a la espera de respuesta **RES**  .

4) Finalmente, hay una especialización del punto 3). Es el caso de Clientes Externos de tipo Agente Simulado (**AS**) que necesiten simular sensores o dispositivos de comunicación:

a. Enviar una **PET**  para ejecutar

i. El comando **addSensor** del servicio **RegSensor** con los datos del sensor (Interfaz, Modelo del Sensor, nombre, etc). En cuanto al nombre del Sensor, el mismo AS no puede registrar dos sensores con el mismo nombre.

ii. Idem con el comando **addDispComun** del servicio **RegDispComun**.

b. Debe quedar a la espera de recibir la confirmación con un mensaje de tipo **RES**  de que ha sido registrado correctamente el Sensor o Dispositivo.

c. A partir de este punto, el **AS** podrá enviar peticiones del comando **medir** del servicio **Medidas** para medir de un sensor registrado o **Send_Msg** del servicio "**MsgSending**" para hacer uso de dispositivos de comunicación.

10.5.- Guía de Implementación

Se recomienda enérgicamente leer el **Manual de Implementación del Servidor del Entorno**, el apartado correspondiente, para ampliar la información aquí mostrada. Este apartado sólo dará pinceladas.

El presente proyecto proporciona herramientas para permitir el rápido escalado del sistema, gracias a la reflexión explicada en el **Apartado 10.3**.

Todo desarrollador debe seguir el fichero de configuración XML del Servidor del Entorno para ampliar la funcionalidad del sistema:

- Crear una nueva clase para el **Proceso Ejecutor** del componente, incorporándola al fichero XML.
    i. Debe heredar de la clase *ProcesoEjecutor.class*, e implementar todas las interfaces de **comandos** de todos los servicios del componente Gestor para el que se crea el ProcesoEjecutor.
- Crear nuevos servicios, haciendo las modificaciones pertinentes en la clase del **Proceso Ejecutor** del componente Gestor.
- Configurar cuantos procesos ejecutores tendrá el componente Gestor, y a qué servicios estarán asociados.
- Crear nuevos **Modelos**, que deben implementar la **Interfaz de Modelos** a la que da implementación y heredar de la clase *Modelo.class*.
- Modificar la **Interfaz de Modelos,** modificando consecuente los Modelos relacionados.

En el **Manual de Implementación del Servidor del Entorno** se encontrarán pautas, consejos, y trozos de código de utilidad para el desarrollador. Así se abre el camino a los desarrolladores que tengan interés en ampliar la funcionalidad del simulador.

10.6.- Caso de Estudio: Distribución en Grid del Servidor del Entorno

Se van a usar continuas referencias a la tecnología RMI de Java. Revisar el **Apartado 4.2.3** y Descripción Técnica de Formatos de Datos Utilizados, entre los documentos anexos al proyecto. En el **Manual de Implementación del Servidor del Entorno** se amplia la información aquí explicada con trozos de código incluidos.

Hasta este punto, se ha explicado el diseño e implementación de los componentes Gestores del Servidor del Entorno considerando que todos los componentes Gestores se encontrasen en la misma máquina:

- Comparten Pizarra con Catálogo de Servicios y Libreta de Direcciones.
- En la Libreta de Direcciones, se encuentra el Buffer de Entrada del Front-End de cada componente necesario para poder encolar mensajes).

El diseño de la aplicación se ha hecho contemplando la posibilidad de ejecutar, de la manera más sencilla y con un conjunto reducido de cambios posibles, un Servidor del Entorno **distribuido** de manera que los componentes Gestores se ejecuten en distintas máquinas. Esto puede tener utilidad en los siguientes supuestos:

- Para optimizar los tiempos de respuesta del Servidor, que estaría montado sobre un **grid** de ordenadores trabajando simultáneamente (frente a una única máquina atendiendo todos los servicios). Evidentemente esta posibilidad tiene sentido siempre y cuanto los tiempos de respuesta en la comunicación de los datos no sea elevado, en cuyo caso se pierde la ventaja de la computación grid.
- Permite la colaboración entre equipos de desarrollo, incluso estando situados en distinta situación geográfica.
- Elimina el cuello de botella que puede suponer que un componente Gestor ofrezca servicios pesados:
- Incluso en el grid, se pueden destinar máquinas adecuadas para cada componente Gestor. Ver el gráfico de la Figura 145.

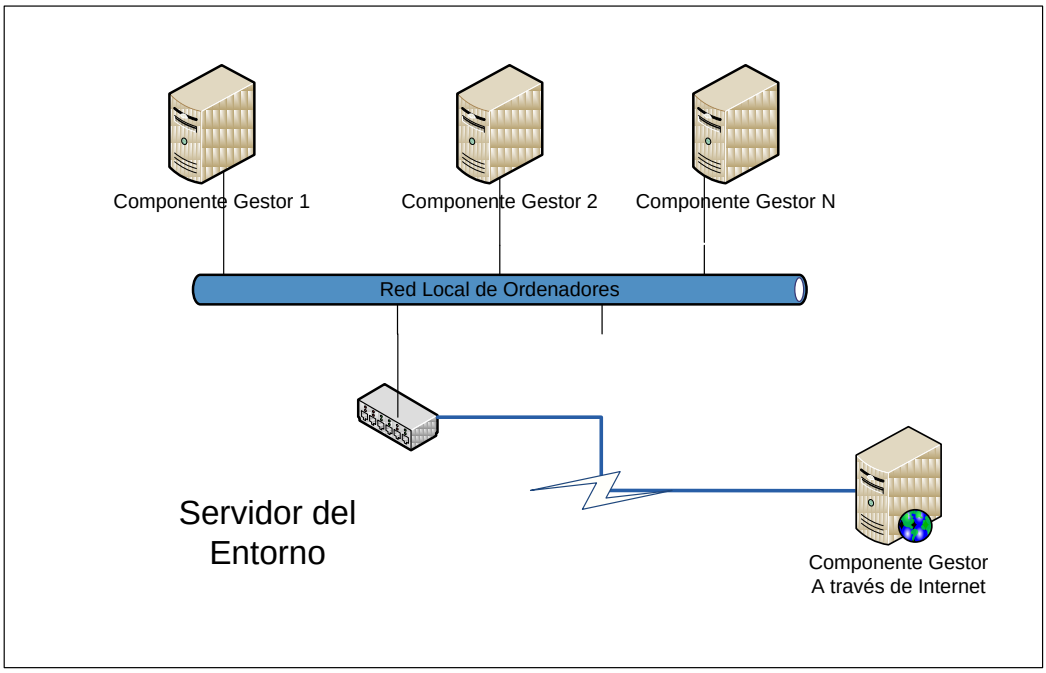


Figura 145: Ejemplo de Servidor del Entorno distribuido.

Al diseño abstracto ya expuesto en el **Apartado 10.2.2**, habría que realizar ciertos cambios para posibilitar esta comunicación entre el grid de computadores.

10.6.1.- Multiplicidad de Pizarra

En el esquema explicado, la pizarra era común a todos los componentes Gestores. En este caso el concepto cambia: cada máquina o nodo del grid tiene que tener su propia Pizarra la cual será compartida por todos los componentes Gestores que se ejecuten en esa máquina.

En cuanto al Catálogo, teniendo acceso al XML el componente Gestor es capaz de cargarlo sin problema.

El aspecto más problemático del diseño del Servidor del Entorno es el que tiene que ver con la Librega de Direcciones, donde se acumulan para cada componente Gestor el buffer de entrada del Front – End o interfaz donde encolar mensajes, que en este caso pueden estar en máquinas remotas. En este caso, los componentes Gestores remotos, en vez de almacenar el buffer de entrada, deberán almacenar el stub compartido por el componente Gestor remoto mediante RMI.

Pongamos el ejemplo de dos nodos remotos en la Figura 146:

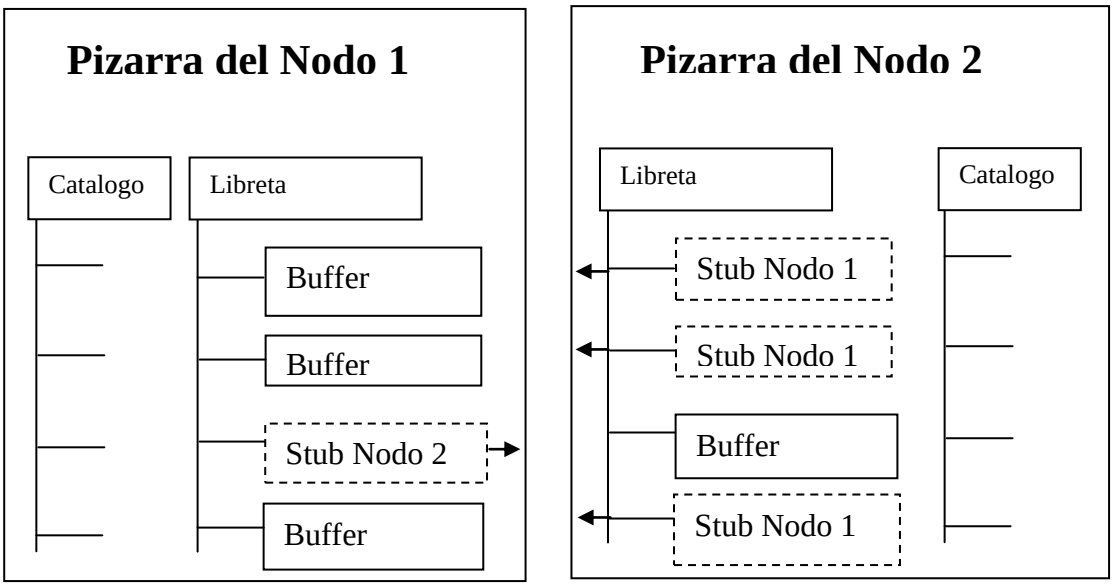


Figura 146: Comparación de dos componentes Gestores remotos.

Se observa la estructura de la pizarra en dos nodos distintos. En el nodo 1, existen tres componentes Gestores, con lo que la libreta tiene almacenado en la libreta el **buffer** del Front-End o interfaz de estos tres componentes.

Sin embargo, existe un componente gestor remoto, en el Nodo 2, por lo que en la libreta del Nodo 1 sólo se puede almacenar el **stub** que apunta al nodo remoto.

La situación es simétrica en el Nodo 2: sólo hay un componente Gestor local, del que almacena el buffer de entrada del Front-end o interfaz, frente a 3 componentes que se dan en un nodo remoto (el nodo 1), y del que sólo se puede almacenar el stub.

CAPÍTULO 11.- *Cliente Simulador de Agentes Simulados*

Si bien no es objetivo del presente proyecto diseñar el sistema de control de un AUV, sí se hace necesario analizar ciertos aspectos relevantes del mismo para poder encajar el simulador.

Por tanto, veremos en este capítulo en qué consiste la misión de un AUV y cómo representarla. Seguiremos hablando de los subsistemas que forman un AUV tipo, y analizaremos el sistema de control de uno de ellos (el *Sick***AUV).**

Finalmente de describirá el prototipo que se ha diseñado e implementado en el marco de este proyecto para evaluar la utilidad del simulador.

11.1.- Especificación de Misión

Un AUV es diseñado para desempeñar una misión de forma autónoma. La misión al final se traduce en un complejo algoritmo de control del AUV. Es por ello que la misión debe expresar de forma clara e inequívoca:

- Los objetivos explícitos a perseguir por el AUV: aquellos que la entidad (normalmente oceanográfica) persigue para realizar estudios.
- Los medios para conseguir los objetivos.
- Otros objetivos implícitos a perseguir por el AUV: aquellos destinados a la supervivencia del AUV, como afrontar situaciones extremas, etc.
- Expresar qué parámetros de la misión son configurables, y cuáles no.

En el entorno de los AUVs, hay dos formas de afrontar el problema de diseñar las misiones:

- **Plantear Misiones Individuales**: La misión describe los objetivos y medios de un único AUV, sin considerarlo en Sociedad. No significa que una misión individual no pueda implicar comunicaciones con puesto de control y con otros AUVs. Pero el flujo de la misión no depende de la comunicación con otros AUVs.
- **Plantear Misiones en Grupo**: el AUV forma parte de una sociedad, que puede estar jerarquizada o no, pero donde la información intercambiada con otros AUVs influye directamente en el comportamiento y el flujo de la misión de cada AUV individualmente.

Las misiones en Grupo son un híbrido: cada AUV tiene una misión individual, pero parte de esta misión depende de información intercambiada con otros AUVs.

El diseño de este proyecto final de carrera va a contemplar la posibilidad de que en un futuro se puedan simular AUVs con misiones en Grupo, si bien se va a profundizar en el desarrollo de misiones individuales de AUVs. En este marco, un plan, es una parte de la misión que se encarga de un objetivo concreto. En su conjunto, forman el algoritmo de control global del AUV.

11.1.1.- Misiones en Grupos: Ejemplo CODA

CODA es un simulador orientado a simulaciones multi AUVs, con técnicas avanzadas de control de misiones conjuntas de grupos de AUVs. Se puede obtener una descripción más detallada en la referencia **[ALB03]**.

Para afrontar este punto, CODA propone una organización jerarquizada de los AUVs en dos niveles: (Ver Figura 147).

- MLO (*meta level organization)*: Fase Deliberativa. Los individuos lo suficientemente inteligentes forman este colectivo. Diseñan la organización del grupo TLO en función de situación inicial, o alguna situación adversa que se dé. Sólo se presenta en la situación inicial o situaciones adversas por tanto. El resto del tiempo se da la fase operativa.

- TLO (*task level organization)*: Fase Operativa. Cada individuo realiza la misión en esta fase diseñada por la MLO.

Existen diversos esquemas configurables para pasar de una a otra fase. También se dan un conjunto de ***protocolos de Cooperación*** para interacción entre los agentes del mismo nivel jerárquico y entre distintos niveles.

El sistema para elegir a los miembros del MLO es por ***heurísticas de restricciones***: individuos que cumplen ciertas características entran a formar parte. Otro esquema que pudiera plantearse es hacer la elección de formas distribuidas, como pueden ser técnicas de subastas, etc.

En el presente proyecto no se trabajado con misiones cooperativas, si bien se han habilitado en el Servidor del Entorno herramientas para que en el futuro se puedan usar estas capacidades, como es el componente Gestor de Dispositivos de Comunicación.

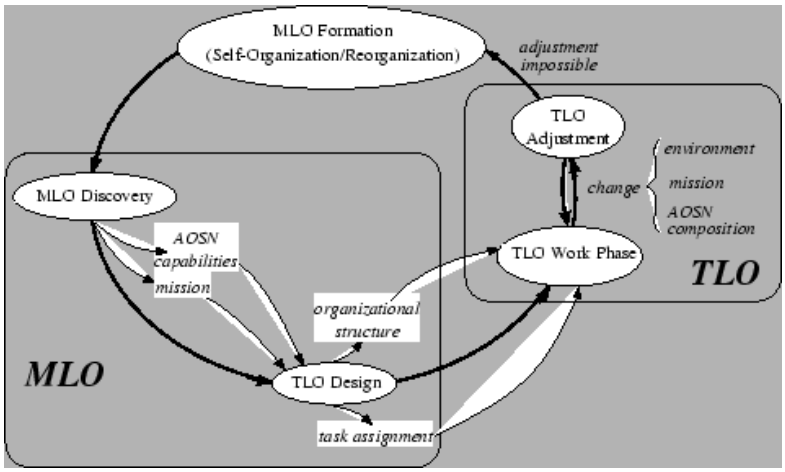


Figura 147: Ejemplo del simulador CODA de misiones en grupo.

11.1.2.- Misiones Individuales

Para mayores detalles, se recomienda al lector leer la **Especificación de Misiones AUV_XML_y_XSD**, documento anexo al proyecto, ya que aquí se expone un breve extracto del mismo.

El caso más sencillo es considerar a un AUV con su propia misión. De una forma sencilla, el uso más común que se le va a dar, o lo que es lo mismo, los objetivos explícitos son:

- Moverse a un determinado punto o zona (**Navegación)**.
- Llegado a esa zona, realizar una serie de mediciones, y almacenarlas (**Medición y Almacenamiento**).
- Completadas las medidas, volver a un punto de encuentro (**Navegación**).

Además, durante el desarrollo de la misión, se hace necesario de forma implícita las siguientes capacidades:

- El AUV debe ser capaz de enviar a un centro de control el estado del mismo (baterías, posición, problemas que se hayan encontrado) para permitir tomar medidas (**Medición** y **Comunicación**).

- El controlador puede cambiar la misión del AUV para adaptarse (**Comunicación** y **Supervisión**).
- El AUV debería ser capaz de cambiar su misión automáticamente en casos de emergencia (**Supervisión**).

Este sencillo ejemplo de uso de un AUV permite identificar los planes que van a hacer falta para describir una misión. Una misión (**mision.xml**) está formada por los siguientes planes (uno o ninguno de cada):

- **Plan de Navegación (pdn):** definir la ruta que debe seguir el AUV, desde funcionar como una boya, sin movimiento alguno, a esquemas combinados de recorrer áreas. También debe implicar definir la velocidad del AUV y pose en cada punto de la ruta.
- **Plan de Medidas (pdm)**: definir qué medidas se deben hacer: tanto la medida como el sensor del que se debe de medir.
- **Plan de Almacenamiento (pda)**: definir que datos deben almacenarse. Desde las medidas del plan de medidas hasta parámetros internos del AUV como logs, batería, etc.
- **Plan de Comunicación (pdc)**: definir cuando debe el AUV enviar y recibir datos de otros AUV o de un controlador, además de definir qué debe comunicarse.
- **Plan de Supervisión (pds)**: definir planes de contingencia, además de cambios automáticos en los planes anteriores ante casos especiales concretos.

Estos planes son ejecutados concurrentemente por el AUV durante su misión. Dentro del grupo de proyectos de la Universidad de Las Palmas de Gran Canaria (ver **ANEXO I**) destinados al desarrollo de sistemas AUVs, se ha trabajado conjuntamente para definir estos planes con un formato común, siguiendo el estándar XML. También se han usado ficheros XSD para definir la estructura de los ficheros XML con los planes de la misión.

Es por ello que vamos a profundizar en el diseño hecho en común, y describir lo que un oceanógrafo puede esperar de cada plan. En la descripción de cada uno se

nombrará el fichero XML desarrollado y que se entrega con el proyecto, para el lector que desee hacer consultas.

Previamente trataremos una nota común entre todos los planes: **en todos se realizan Tareas.**

Tareas

Para cada tarea que tiene que realizar un AUV dentro de un plan el científico debe especificar

- Condiciones: disparan la acción.
- Acciones: tarea en sí a realizar sólo cuando se cumpla la acción. Dependen de cada plan, y, por tanto, serán explicadas en cada uno.
- Periodo de Repetición: una vez cumplida la condición, tiempo que pasará hasta que se pueda volver a disparar. Se muestra en la Figura 148.

```
<tarea id="1" nombre="AlmacenarTemperaturaInterna">
    <disparadores:disparadores>
    ....
   </disparadores:disparadores>
     <acciones>
    .....
    </acciones>
    <periodoRepeticion valor="10" unidad="minuto" />
</tarea>
```

Figura 148: Código XML para definir una tarea de misión.

Las Condiciones provocan o disparan el comienzo de la tarea, usando tipos de reglas básicas combinadas entre sí mediante AND y OR lógicos.

En realidad esta forma de especificar condiciones sirve para cualquier plan en cuyo fichero XML se encuentre la etiqueta **disparadores**. Propondremos diversos ejemplos con el plan de medidas (si bien es aplicable a otros planes, como veremos), para que sea más cercano al lector:

- **Condición "siempre"**: esta tarea de medición siempre está activa, independiente de cualquier condición. Esta regla no es combinable con ninguna de las otras, como se ve en la Figura 149.

```
<disparadores:disparadores/>
```

Figura 149: condición que siempre se dispara.

- **Condición "booleana"**: La tarea se dispara en función del valor de una determinada magnitud, de forma que si es igual, menor, menor o igual, mayor, mayor o igual, distinto que un umbral, se dispara la tarea.

Ejemplo: *Se desea que el AUV tome la temperatura a 200 Hz durante 1 minuto y, una vez disparado, que tarde 10 minutos en volver a poderse disparar como mínimo (periodo de repetición):*

- **Si el AUV está en el waypoint 5 (waypoint=5).**

```
<tarea id="1" nombre="AlmacenarTemperaturaInterna">
    <disparadores:disparadores>
        <disparadores:condicion id="1" medida="waypoint" operador="eq"
            valor="5" unidad="waypoint"/>
    </disparadores:disparadores>
     <acciones>
       // Tomar temperatura a 200Hz durante 1 minuto
    </acciones>
    <periodoRepeticion valor="10" unidad="minuto" />
</tarea>
```

Figura 150: Ejemplo de condición booleana.

- **Si la presión es superior a 2 atmósferas (presión >2 atm).**

```
<disparadores:condicion id="1" medida="presion" operador="mt" valor="2" unidad="atm"/>
```

Figura 151: Ejemplo de condición booleana.

- **Si son las 14:00 horas (tiempo=14:00)**

```
<disparadores:condicion id="1" medida="tiempo" operador="eq" valor="14" unidad="h"/>
```

Figura 152: Ejemplo de condición booleana.

- **Si el AUV se encuentra en la latitud 10 (latitud = 10)**

```
<disparadores:condicion id="1" medida="latitud" operador="eq" valor="10" unidad="°N"/>
```

Figura 153: Ejemplo de condición booleana.

- **Condición "intervalo"**: la tarea se dispara si una determinada magnitud se encuentra entre 2 umbrales con un paso predeterminado e incluso alguno de los umbrales pudiendo ser infinito. Sobre todo está pensado para usar con la magnitud tiempo y waypoint, que avanzan creciendo (no pueden decrecer) y con un paso constante: los waypoints avanzan de uno en uno, y el tiempo cada minuto).

Ejemplo: *Se desea que el AUV tome la temperatura a 200 Hz durante 1 minuto*:

- **Si el AUV está entre el waypoint 3 y 9 de dos en dos**. Los waypoints son una magnitud que crece linealmente y por tanto el AUV recorre en orden. Por ello, medirá en el 3,5,7,9.

```
<disparadores:intervalo id = "2" medida="waypoint" inicio = "3" fin="9" periodo="2" unidad="waypoint"/>
```

Figura 154: Condición "intervalo".

- **Condición "excepción"**: sobre todo está pensado para casos de contingencia. Por ejemplo "cuando **Batería bajas** entonces…".

```
<disparadores:excepcion id="3" nombre="agotamientoBaterias"/>
```

Figura 155: Condición Excepción.

- Finalmente, se pueden combinar condiciones de distinta clase, aumentando así la potencia del plan.

Ejemplo: *"Se desea que el AUV tome la temperatura a 200Hz durante 1 minuto, pudiendo repetirse la tarea sólo 10 minutos después de un disparo anterior."*

- **Si el AUV está entre el waypoint 10 y 15 cada 2 waypoints, y además son más de las 14:00**

```
<tarea id="1" nombre="AlmacenarTemperaturaInterna">
    <disparadores:disparadores>
       <disparadores:intervalo id="4" medida="waypoint" inicio="10" fin="15"
          periodo="2" unidad="waypoint"/>
       <disparadores:condicion id="5" medida="tiempo" operador="mt"
          valor="14" unidad = "h"/>
    </disparadores:disparadores>
     <acciones>
       // Tomar temperatura a 200Hz durante 1 minuto
    </acciones>
    <periodoRepeticion valor="10" unidad="minuto" />
</tarea>
```

Figura 156: Ejemplo de combinación de condiciones de distinto tipo.

Para crear un OR lógico, basta con crear dos tareas diferentes, con la misma acción pero con condiciones de disparo diferentes.

Plan de Navegación (pdn.xml)

Es el único plan que no se define en base a tareas concurrentes (las descritas en el apartado anterior).

En primera instancia, puede haber 3 formas básicas de definir la navegación Una vez vistas explicaremos como pueden ser combinadas para formar rutas más complejas

- **Rutas**: se define una serie de waypoints o puntos consecutivos por los que debe pasar el AUV, y en cada uno de ellos la pose con la que debe de pasar, la velocidad y el momento en el que debe de pasar, tal como se muestra en la Figura 157 y Figura 158.

  Llamando transecto al segmento delimitado por dos waypoints contiguos, debe definirse la velocidad "de crucero" a la que debe navegar el AUV durante el recorrido del transecto.

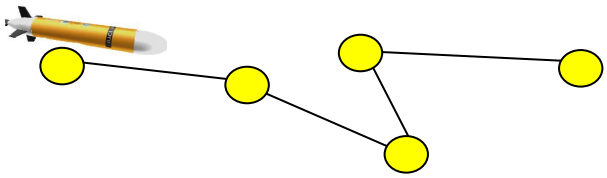


Figura 157: Plan de navegación mediante Ruta.

```
<ruta id="1" nombre="salida">
    <waypoint id="1">
    <pose>
      <posicion x="30" y="20" z="-10" unidad="m" />
      <orientacion roll="0" pitch="0" yaw="1" unidad="rad" />
      <incertidumbre valor="2" unidad="m" />
    </pose>
    <velocidad>
      <posicion x="1" y="0.2" z="0" unidad="m*s^-1" />
      <orientacion roll="0" pitch="0" yaw="0.1" unidad="rad*s^-1" />
    </velocidad>
    <interpolacion valor="10" unidad="m" />
    </waypoint>

    <waypoint id="2">
    <pose>
      <posicion x="30" y="20" z="-10" unidad="m" />
      <incertidumbre valor="2" unidad="m" />
    </pose>
    <interpolacion valor="10" unidad="m" />
    </waypoint>

    <transecto id="1" inicio="1" fin="2">
    <velocidad>
      <posicion x="1" y="0.2" z="0" unidad="m*s^-1" />
      <orientacion roll="0" pitch="0" yaw="0" unidad="rad*s^-1" />
    </velocidad>
    </transecto>
</ruta>
```

Figura 158: Ejemplo de ruta en fichero XML del plan de navegación.

- **Área**: viene delimitada por un conjunto de vértices tridimensionales, y una profundidad mínima y máxima. Esto define un volumen de tipo prisma en el que puede moverse el AUV. Además, puede especificarse un límite temporal, lo que equivale a establecer un tiempo máximo para recorrer el área.

  Ya delimitada temporal y espacialmente el área, existen diversas maneras de recorrerla. Las más interesantes son (y se muestran en la Figura 159):

  - **Deriva**: Sencillamente dejarse llevar el AUV hasta que se complete el tiempo máximo. Puede verse un ejemplo en la Figura 160.
  - **Seguimiento de Magnitud**: el AUV se desplaza en el área en función de una magnitud concreta, desde simplemente

mantenerse en la zona del área donde el valor de la magnitud esté entre dos límites, a seguir una función más compleja como puede ser un gradiente.

- **Transectos**: definir una forma prefijada de recorrer el área, una especie de "recorridos tipo". Por ejemplo, hacer un zigzag de límite a límite del área y descendiendo una determinada profundidad cada cierto tiempo.
- **Combinado de las anteriores**: dependiendo de ciertas **condiciones**, un AUV puede cambiar de un modo a otro de recorrer el Área. Por ejemplo:
  - Si la temperatura está entre 20ºC y 30ºC, hacer un seguimiento de la Temperatura usando una función gradiente.
  - En el caso que la temperatura sea superior a 30 ºC, ir a la deriva.
  - En el caso de que sea menor a 20ºC hacer un zigzag hasta que encuentre gradiente.

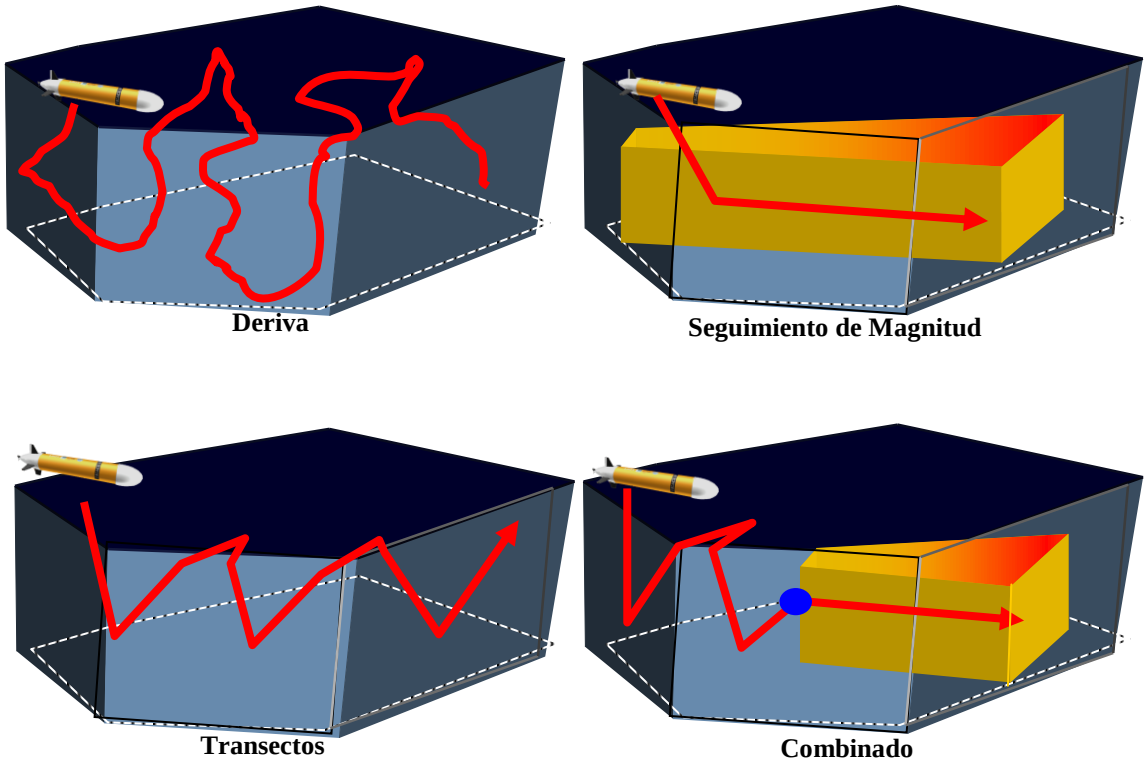


Figura 159: Plan de navegación: distintos métodos de recorrer un área.

```
<area id="2" nombre="deriva">

  <vertice id="1" x="40" y="20" z="10" unidad="m" />
  <vertice id="2" x="20" y="10" z="20" unidad="m" />
  <vertice id="3" x="30" y="20" z="10" unidad="m" />
  <vertice id="4" x="20" y="20" z="10" unidad="m" />

  <profundidad minima="10" maxima="100" unidad="m" />

  <tiempo valor="30" unidad="minuto" holgura="10" />

   <recorrido id="1">
  <disparadores:disparadores />
  <deriva />
  </recorrido>

</area>
```

Figura 160: Ejemplo de definición de área para recorrer a la deriva.

- Zona Prohibida: área que el AUV debe evitar navegar. Se podría definir de forma muy similar al volumen de las áreas.

Plan de Mediciones (pdm.xml)

En este caso, el investigador va a estar interesado en que el AUV haga un conjunto de mediciones, y en que las haga en un momento concreto. La primera cuestión es "qué medir": para cada medida que quiera realizar, se planificará la medición usando los parámetros:

- La magnitud a medir.
- La frecuencia de medida.
- El límite de muestras, bien especificando el número de muestras, o el tiempo que se mantiene $midiendo a la frecuencia indicada.
- La resolución con la que quiere hacer la medida.

Véase que sólo con especificar la magnitud, ya el AUV tiene la inteligencia suficiente para resolver de cuál de sus sensores debe medir. Incluso si tuviese varios para la misma magnitud, puede llevar a cabo distintas alternativas para la resolución de forma autónoma. Algunas estrategias de resolución serían:

- Seleccionar uno cualquiera.
- Seleccionar uno según lista de prioridades.

- Seleccionar uno según la resolución y frecuencia especificadas.
- Medir de todos y hacer una integración de los datos, por ejemplo con un filtro de Kalman.

Opcionalmente, el investigador puede requerir que la medida se haga de un sensor concreto. Por ejemplo, porque está sometiendo el sensor a una depuración o sencillamente porque el investigador confié en un sensor concreto.

También opcionalmente el investigador puede requerir que el AUV no lo resuelva de forma autónoma. Incluso especificando el sensor, puede opcionalmente querer una configuración concreta de ese sensor, o por el contrario usar la configuración por defecto.

Por ejemplo (Figura 161): *Se desea medir la temperatura a una frecuencia de 200Hz durante hasta un total de 500 muestras. Se hace la medida desde un sensor específico, en este caso el sensor de temperatura integrado en la brújula electrónica TCM2*.

```
<acciones>
    <medir id="1" magnitud="temperatura" cantidad_muestras="500">
      <frecuencia valor="200" unidad="Hz" />
      <resolucion valor="1" unidad="ºC" />
      <sensor id="1" nombre="brujulaTCM2" />
    </medir>
</acciones>
```

Figura 161: Ejemplo de plan de medidas.

Por ejemplo: *Se desea medir la temperatura a una frecuencia de 200 Hz durante 10 minutos, y que sea medida por el sensor TC400. Se hace la medida desde el sensor TC400 con su configuración por defecto*.

Por ejemplo: *Deseo medir la temperatura a una frecuencia de 200 Hz durante 10 minutos, y que sea medida por el sensor TC400 en bajo consumo. Se hace la medida desde el sensor TC400 con su configuración bajo consumo*.

En cuanto el momento de hacer las medidas, para cada medida que quiere planificar un científico puede especificar las **condiciones disparadoras** que provocan el comienzo de las mismas.

Considerando las condiciones, un ejemplo completo sería:

Por ejemplo: *Se desea medir la temperatura a una frecuencia de 200 Hz durante 10 minutos* ***si la temperatura es mayor a 20ºC*****.**

Plan de Almacenamiento (pda.xml)

En este caso el científico está interesado en especificar qué desea almacenar, y bajo qué condiciones.

En cuanto a "qué almacenar", se resuelve especificando qué magnitud desea medirse, y el número de muestras a almacenar.

En cuanto al momento, para cada almacenamiento que quiere planificar un científico puede especificar las **condiciones disparadoras** que provocan el comienzo de las mismas.

```
<acciones>
    <almacenar id="1" magnitud="temperatura"
        cantidad_muestras="100" />
</acciones>
```

Figura 162: Ejemplo plan de almacenamiento.

*Por ejemplo (*Figura 162*):* Se desea almacenar las 100 medidas de temperatura **entre las 14:00 y las 15:00 cada 5 minutos si la temperatura >6ºC.**

Plan de Comunicaciones (pdc.xml)

El científico desea especificar qué desea comunicar, y bajo qué condiciones. En cuanto a planificar lo que se va a enviar se puede hacer de cuatro maneras básicas diferentes, y de la combinación de las mismas (siempre de forma aditiva o AND lógico):

- Enviar un dato: el científico desea que el AUV, bajo ciertas condiciones, envíe un dato de los almacenados en el plan de almacenamiento. Por tanto, hay que especificar qué magnitud se desea comunicar y el destinatario.

  Por ejemplo (Figura 163): *Se desea que el AUV envíe todas las temperaturas que ha tomado al controlador de la misión en la ip 172.172.172.172.*

```
<acciones>
    <enviarDato id="1" dato="historicoTemp"
            destinatario="172.172.172.172" />
</acciones>
```

Figura 163: Ejemplo de Plan de Comunicaciones.

- Enviar una medida: similar al anterior, pero el dato no lo toma de los almacenados, sino que toma el dato directamente del sensor. Por tanto debe especificarse qué magnitud se desea comunicar, y el destinatario.

  Por ejemplo: *Se desea que el AUV envíe lo que está midiendo el sensor de temperatura justo en este momento al controlador de la misión en la ip 172.172.172.172.*

- Recibir un dato: el caso simétrico con enviar un dato. En este caso se tiene que especificar la fuente que tiene que mandar el dato:

  Por ejemplo: *Se desea recibir datos almacenados de temperatura del AUV4 en la ip 122.122.122.122.*

- Recibir una medida: el caso simétrico con enviar una medida.

  Por ejemplo: *Se desea recibir los datos del sensor de temperatura del AUV4 en la ip 122.122.122.122*

En cuanto a decidir el momento de hacer el envío, para cada tarea de comunicación que quiere planificar un científico puede especificar las **condiciones disparadoras** que provocan el comienzo de las mismas.

Vamos a poner un ejemplo combinando las formas básicas antes especificadas, y con una condición de disparo:

Por ejemplo: *Se desea que, si **la temperatura < 5 y se produce una excepción "batería baja"** el AUV envíe al controlador de la misión en la dirección ip 172.172.172.172 todas las medidas de temperatura que ha almacenado, y además envíe justo la medida que ahora da el sensor.*

Por ejemplo: *Se desea que, **si está en el waypoint final (el 35)**, el AUV envíe al AUV4 en la dirección ip 122.122.122.122 la medida de presión que da ahora mismo el sensor, y además que el AUV reciba la medida de temperatura del AUV7 con dirección ip 155.155.155.155.*

Plan de Supervisión (pds.xml)

El científico desea especificar qué tareas de supervisión realizar, y bajo qué condiciones realizarlas

En cuanto a planificar lo que se va a supervisar se puede hacer de dos maneras diferentes, y de la combinación de las mismas (siempre de forma aditiva, AND lógico).

- **Ejecutar un Comando**: sirve para realizar tareas concretas: cambiar la configuración de algún sensor, cambiar algún parámetro de cualquier otro plan, activar /desactivar dispositivos, etc.

  Se hace mediante **xpath**, que permite hacer modificaciones de los ficheros XML de los planes de misión en tiempo de ejecución.

  Por ejemplo (Figura 164):

  - Cambiar la configuración del termómetro TC400, y ponerle una configuración de bajo consumo.
  - Desactivar sensores de temperatura.
  - Cambiar la variable del plan de medida, de forma que ahora en vez de medir cada 5 minutos, mida cada 10 minutos.

```
<acciones>
    <ejecutarComando id="3" comando="CambiarVariableMision">
                    <parametro id="1" valor="@FrecuenciaMuestreo" />
                    <parametro id="2" valor="10" />
    </ejecutarComando>
</acciones>
```

Figura 164: Ejemplo de plan de supervisión. Ejecutar un Comando.

- **Ejecutar un Plan Completo**: sirve para realizar un plan de contingencia completo, y está pensado para excepciones. Puede verse como un conjunto de comandos, invocados de forma grupal por sencillez.
    - Emerger, lo cual implica muchos comandos sobre todo a nivel de plan de navegación, añadiendo una tarea con una ruta que lleve ala superficie. Véase la Figura 165.

```
<acciones>
    <ejecutarPlan id="1" plan="Emerger">
      <parametro id="1" valor="25" />
    </ejecutarPlan>
</acciones>
```

Figura 165: Ejemplo de plan de Supervisión. Plan Completo.

En a cuando ejecutar las tareas de supervisión, para cada tarea de supervisión que quiere planificar un científico puede especificar las **condiciones disparadoras** que provocan el comienzo de las mismas.

Vamos a poner un ejemplo combinando las formas básicas antes especificadas, y con una condición de disparo:

- **Si se produce la excepción "agotamiento de baterías"** entonces ejecutar el plan "emerger" y desactivar los sensores de temperatura, presión, salinidad y conductividad.
- **Si la temperatura está entre 20ºC y 30ºC** entonces cambiar la frecuencia de medición de temperatura del plan de medidas, de 200Hz a 300Hz.

11.2.- Especificación de AUV

Dado que el objetivo principal de este proyecto es simular un AUV, en este apartado vamos a especificar que aspectos físicos son necesarios tener en cuenta para describir a un AUV de la forma más completa posible.

Así, en un AUV podemos observar un núcleo central, y una serie de sistemas periféricos en la Figura 166:

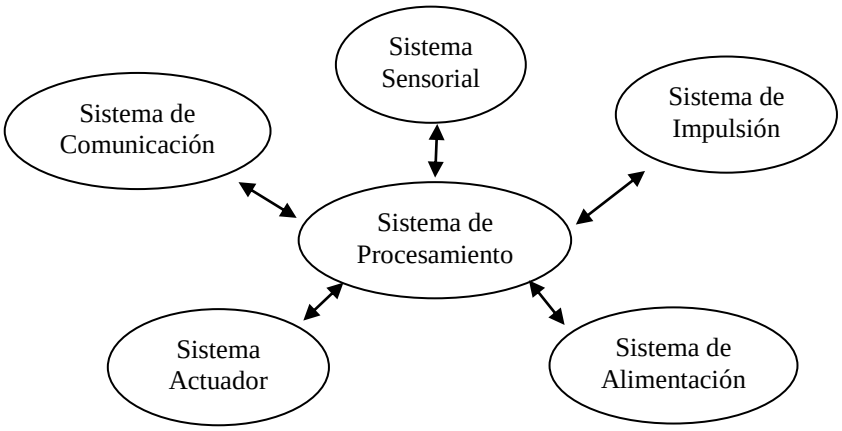


Figura 166: Sistemas de un AUV.

- **Núcleo central**: Integra todos sistemas periféricos además de llevar la lógica de control del sistema.
    - **Sistema de procesamiento**: Para describir el sistema de procesamiento, es necesario especificar:
        - Descripción de CPUs, con las características de operación frecuencia de procesamiento, número de núcleos, consumo.
        - Memorias, con las características, capacidad, frecuencia de acceso, consumo, etc.
        - Tarjetas, con la descripción de accesorios del sistema de procesamiento como pueden ser tarjetas de adquisición de señal, o conversores analógico-digital, etc.

- Sistemas Periféricos: Llevan el control de cierta parte del sistema:

    - **Sistema sensorial**: Hace referencia a los sensores: dispositivos para tomar medidas a distintos niveles:
        - Sensores de Intrumentación: sensores para la navegación instrumental del AUV, como son giróscopos, brújulas, velocímetro, etc.

- Sensores Internos: medir magnitudes internas del AUV, como son humedad, temperatura interna, etc.
- Sensores de la Misión: sirven para realizar las medidas objetivo de la misión, como son salinidad, conductividad, temperatura exterior, presión, etc.

Es interesante conocer las características, tanto modelo, como fabricante, magnitud que mide, resolución, frecuencia de muestreo, unidades, configuración por defecto, configuraciones que admite, etc. También conocer en que posición se encuentran en el AUV.

- **Sistema de impulsión**: conforma todo dispositivo empleado para la navegación del vehículo, entre ellos:
  - **Mandos de Control**: superficies de control de la pose para maniobrabilidad y navegación del AUV. Por ejemplo aletas, cola, etc. Es importante su consumo.
  - **Motores**: definición de los motores, potencia, velocidad, si son bidireccionales, consumo, etc.

Aparte, también hay que especificar ciertos parámetros del sistema de impulsión en general, como son:

- las dimensiones del AUV
- ubicación de motores y mandos de control
- parámetros de configuración de la hidrodinámica
- parámetros de configuración del vehículo, como són velocidad máxima, rozamiento, radio de giro mínimo, etc.
- parámetros de configuración del algoritmo de control.

Ambos subsistemas conforman los grados de libertad que tiene el AUV a la hora de la navegación.

- **Sistema de comunicación**: todo dispositivo empleado para comunicación de datos con otros AUV o sistemas de control. Por

ejemplo antenas GPS, antena satélite , comunicación vía serie , vía wifi con antenas wireless, etc.

Es interesante conocer la posición que ocupa en el AUV, configuración por defecto, configuraciones posibles, ancho de banda, consumo, etc.

- **Sistema de alimentación**: descripción de las baterías y otros dispositivos de alimentación con paneles solares, etc. Carga de las baterías, configuración de otros dispositivos, etc.

- **Sistema actuador**: Otros actuadores distintos a motores y controles orientados a navegación. Es interesante conocer la posición que ocupan en el AUV, configuración, configuración por defecto, el consumo, etc.

11.3.- Caso Práctico: Adaptación de SickAUV

11.3.1.- Sick AUV

Se pretende con este apartado describir, con un sistema de control real y concreto, que es el *Sick***AUV** (ver [**ENR08**] para más información), cómo conectarlo al Servidor del Entorno del presente Proyecto, para que pueda funcionar como un Cliente Externo de tipo Agente Simulado (**AS**) en el simulador ***SUBES***.

El sistema de control del *Sick***AUV** está integrado por los componentes que se muestran en la Figura 167:

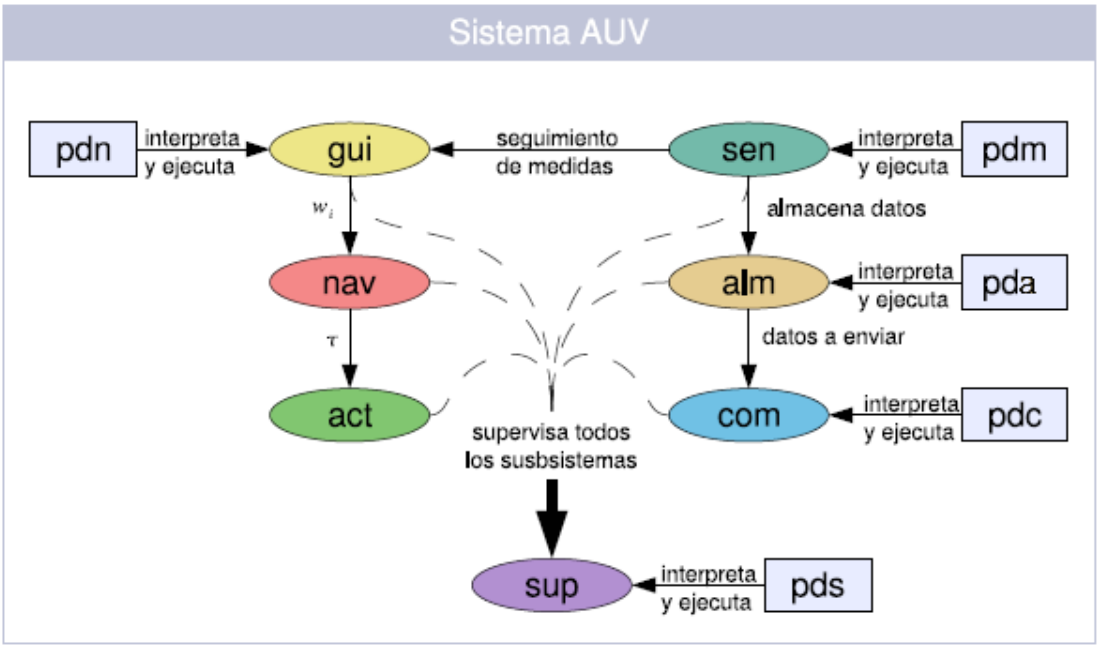


Figura 167: Sistema SickAUV con las relaciones entre sus subsistemas.[10]

Cada uno de los 6 subsistemas mostrados en la Figura 167 están diseñados como **Componentes Compuestos COOLBOT** interrelacionados.

- **gui** *"Subsistema de Guiado"*: interpreta el plan **pdn**, gestionando sus *condiciones disparadoras*, y acorde al mismo, genera los waypoints a los que el vehículo debe llegar para cumplir el plan. Estos waypoints serán enviados al componente **nav**.

- **nav** *"Subsistema de Navegacion"*: toma los waypoints objetivos en orden, y ejecuta los algoritmos de control y ecuaciones hidrodinámicas para deducir qué comandos y órdenes debe enviar a los actuadores (**act**) para conseguir el objetivo

- **act** *"Subsistema Actuador"*: gestiona las peticiones o comandos de control del **nav**, actuando directamente sobre los dispositivos actuadores (motores, aletas, etc). En principio sólo se consideran actuadores dedicados a la navegación y guiado del vehículo.

[10] En el diseño original presentado en [ENR08], no está presente el pda, dado que no fue implementado en el marco del proyecto, si bien se propone su implementación en las aclaraciones posteriores. Es por ello que para la presente explicación se incluye.

- **sen** *"Subsistema Sensorial"*: interpreta el **pdm,** gestionando sus *condiciones disparadoras* y acorde a las mismas, genera peticiones o comandos de control a los sensores del vehículo (para configurarlos, medir, etc). El resultado de las peticiones son enviadas al subsistema **alm**.

- **alm** *"Subsistema de Almacenamiento"*: interpreta el **pda**, gestionando sus *condiciones disparadoras* y acorde a las mismas, genera peticiones o comandos de control de lectura y escritura a los dispositivos de almacenamiento. Los datos almacenados son usados por el subsistema **com**.

- **com** *"Subsistema de Comunicación"*: interpreta el **pdc**, gestionando sus *condiciones disparadoras* y generando peticiones o comandos de envío y recepción a sus dispositivos de comunicación. También gestiona las peticiones de control remoto provenientes de un Subsistema Externo de tipo Controlador de Agente Simulado (**CCAS**)

- **sup** *"Subsistema de Supervision"*: interpreta del **pds**, gestionando sus *condiciones diaparadoras,* registrando qué se quiere supervisar (componentes o parámetros de otros subsistemas), y cuando se disparan se generan peticiones o comandos de control al resto de subsistemas para modificar su actividad.

11.3.2.-Caso de estudio: Simulación de Componentes

Descritos ya en el **Capítulo 10** los servicios y comandos ofertados por el Servidor del Entorno, el *Sick***AUV** real podría hacer uso de la simulación a los siguientes niveles:

- **gui / nav / act**:

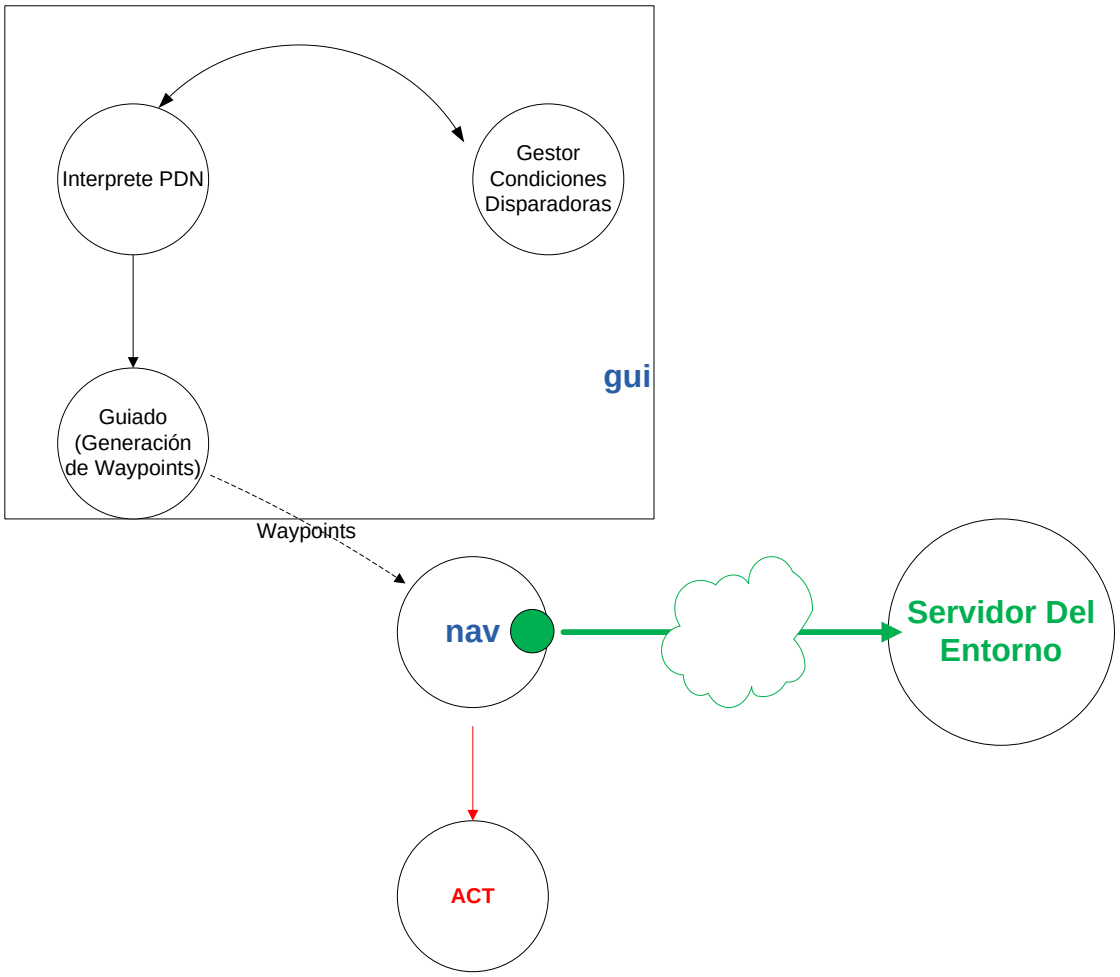


Figura 168: Adaptación de SickAUV para usar SUBES para navegación.

Según la Figura 168, en el caso de que *Sick***AUV** no dispusiera de actuadores, y quisiera simular su navegación e hidrodinámica con el Servidor del Entorno, simplemente habría que prescindir del Componentes compuesto **act**. También es precioso que el subsistema **nav**, tras ejecutar todos los algoritmos de control y ecuaciones hidrodinámicas, disponga de un **pequeño módulo** (marcado en verde) que:

- Traduzca estas conclusiones en peticiones al Servidor del Entorno (peticiones para cambiar el vector de velocidad del AUV en general, por ejemplo, ya que en la implementación hecha del Servidor del Entorno no se presentan actuadores en sí del vehículo.)
- Las envíe al mismo.
- Quede en espera de la respuesta.

Este módulo hace las veces de wrapper, o "enchufe" que permite la interfaz entre ambas entidades.

- **Sen:** en el caso de que el *Sick***AUV** no disponga de sensores reales, puede simularlos usando el Servidor del Entorno:

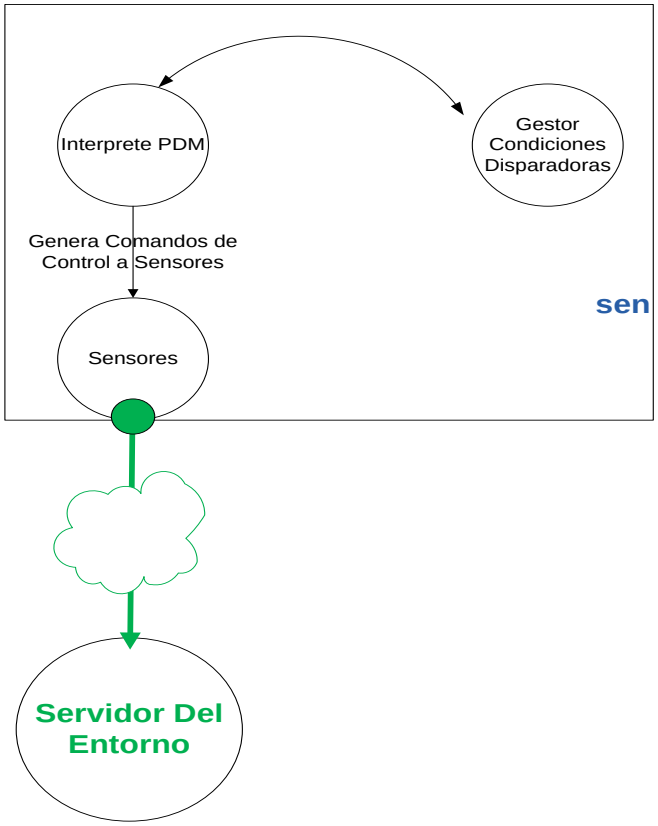


Figura 169: Adaptación de SickAUV para usar SUBES para simulación de sensores.

Como vemos en la Figura 169, dentro del Subsistema **sen**, el componente que recibe comandos de control y que los dirige al driver del sensor concreto (**sensores**), lo único que tiene que hacer es:

- Si el sensor no está disponible (puede que *Sick***AUV**, en su montaje real disponga de sensores reales para hacer sus mediciones, y sin embargo no disponga de otros que tengan que ser simulados).
- Redirigir la petición al **wrapper**.
- Éste lo traduce a una petición en el formato del Servidor del Entorno. Serán peticiones de tipo medir, configurar sensor, etc.
- Envía la petición.

- Queda a la espera de respuesta, para almacenarla en el subsistema **alm**.

- **Com:** en el caso de que el *Sick***AUV** no disponga de dispositivos de comunicación, puede simularlos usando el Servidor del Entorno:

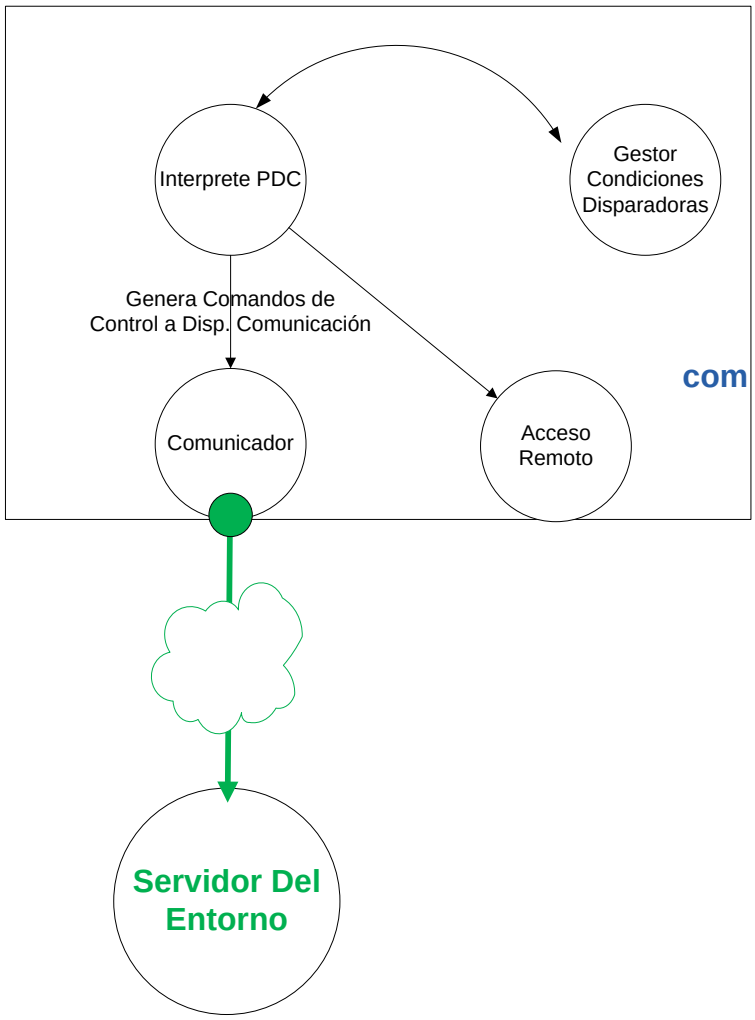


Figura 170: Adaptación de SickAUV para usar SUBES para simular dispositivos de comunicación.

Como vemos en la Figura 170, dentro del Subsistema **com**, el componente que recibe comandos de envío y recepción de datos y que los dirige al driver del dispositivo de comunicación concreto (**Comunicador)**, lo único que tiene que hacer es:

- Si el dispositivo de comunicación no está disponible (puede que *Sick***AUV** sí que disponga de ciertos dispositivos, y de otros no).

- Redirigir la petición al **wrapper**.

- Éste lo traduce a una petición en el formato del Servidor del Entorno.

- Envía la petición.

- Queda a la espera de respuesta si se da respuesta.

11.4.- Prototipo de Agente completamente Simulado

En el marco del presente proyecto, se diseñó e implementó un Subsistema de tipo **Agente Simulado (AS)**. El objetivo es probar las capacidades y rendimiento del Servidor del Entorno (**SE**) a la hora de simular múltiples clientes **AS** conectados, solicitando servicios y comandos simultáneamente en una simulación.

El **AS** es un hilo de ejecución que repite un lazo de control que contempla:

- **Un plan de navegación:** sigue de forma autónoma un vector tridimensional de velocidad constante y configurado por el usuario de antemano. Un cliente controlador externo (**CCAS**) puede cambiar este vector.
  - Las tres componentes del vector de velocidad son los mostrados en la Tabla 45:

Variación de Latitud **(m/s)**	Variación de Longitud **(m/s)**	Variación de Profundidad **(m/s)**

Tabla 45: Componentes del Vector de velocidad del Agente Simulado.

  - Se puede activar en el agente de forma opcional un plan de barrido en profundidad: cuando el AUV llega a la superficie o profundidad 0, cambia el componente de **variación de profundidad** de sentido y comienza a sumergirse. De forma similar cuando llega a un umbral de distancia con el fondo o profundidad máxima sin riesgo de choque, cambia su vector de velocidad emergiendo. Todo esto sucede de forma autónoma. Se muestra este tipo de misión en la Figura 171.

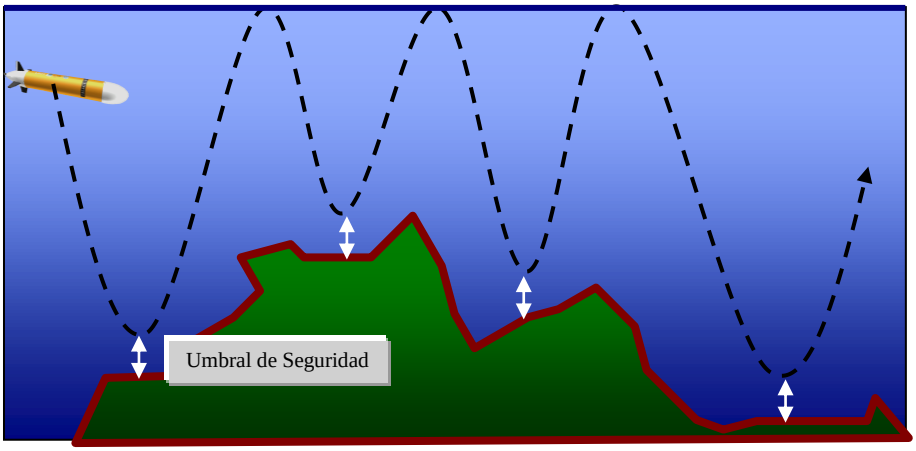


Figura 171: plan de barrido en profundidad.

- **Un plan de medición** sencillo: periódicamente toma de forma autónoma una muestra de sus sensores (aquellos que un cliente controlador externo **CCAS** haya especificado**).**
- Finalmente, puede recibir **peticiones del cliente controlador externo CCAS**, como por ejemplo solicitar una medida de un sensor concreto (petición que se sale del plan de medida), un cambio en la velocidad del plan de navegación, pedir información del AUV, como son sus sensores activos, etc. En cada ciclo de simulación comprobará las peticiones pendientes y las ejecutará.

En conclusión este **AS** está diseñado para poder interaccionar con un **CCAS**. En la implementación final, como veremos en el **Capítulo 12**, el **CCAS** es la Interfaz Gráfica prototipo propuesta: de esta manera el usuario de la interfaz puede de forma gráfica y sencilla crear, gestionar y monitorizar tantos **AS** como estime para su simulación.

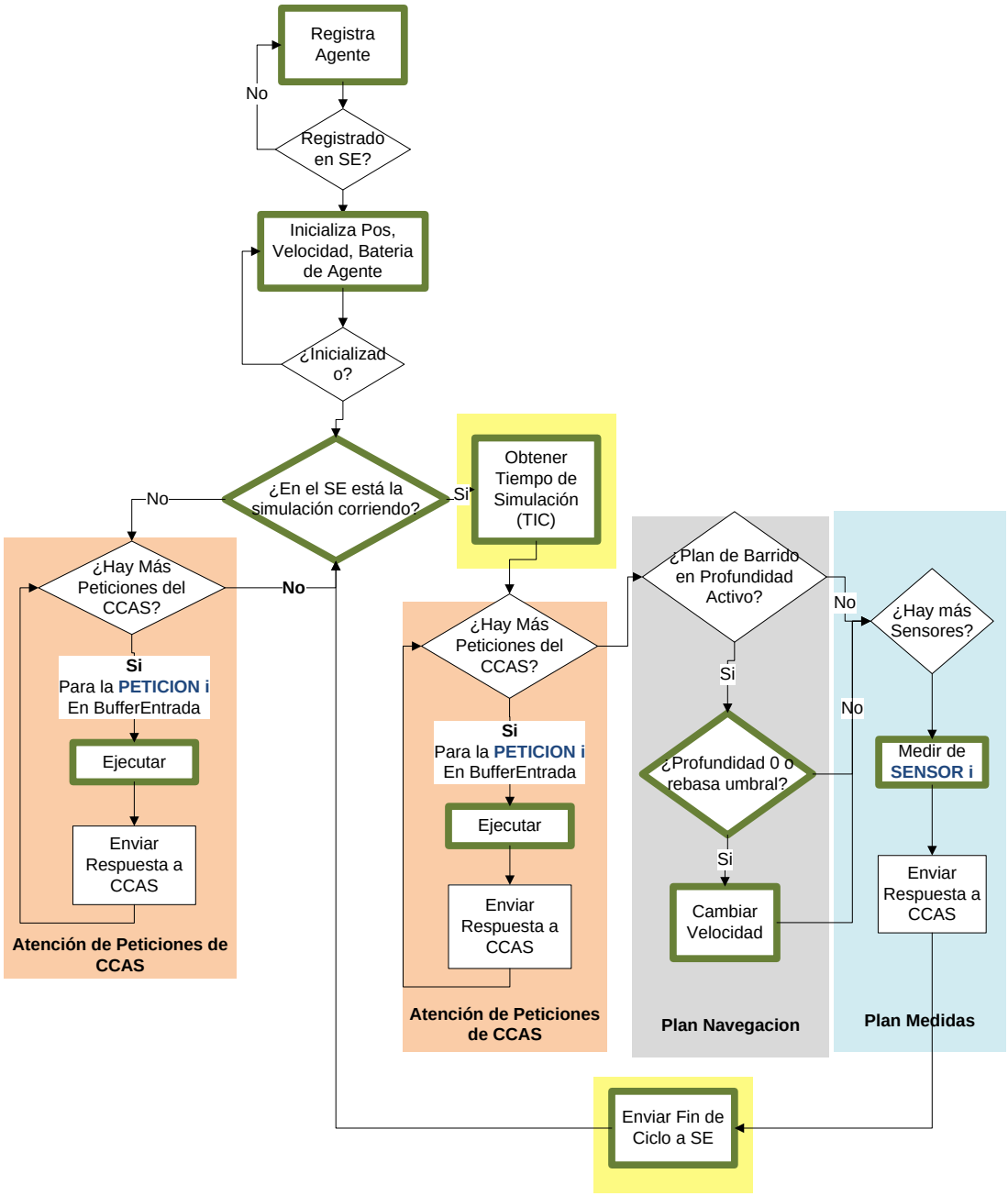


Figura 172: Diagrama de flujo de control del AS.

En la Figura 172 anterior se observa el flujo de control del **AS**, donde:

- Se marca en Verde todos los procesos que requieren petición al Servidor del Entorno (**SE**).
- En fondo amarillo se marca las acciones destinadas a sincronizar el **AS** con el **SE**: recibir el TIC o mensaje de sincronización, y enviar el mensaje de confirmación o fin de ciclo para permitir al SE que avance el ciclo de simulación.

- El resto de cuadros vienen explicados en el diagrama.

Para su operación, el **AS** utiliza 2 sockets (tal cual marca el protocolo de interacción con el **SE**):

- El Socket con el que se hace el registro del Agente, se utiliza en los procesos de sincronización amarillos.
- Se abre otro socket, para el resto de peticiones al **SE**. Estas peticiones son:
    - Registrar el **AS** en el **SE**.
    - Inicializar la posición velocidad y batería inicial del **AS** en el **SE**.
    - Peticiones del **CCAS**: de éstas destacan las peticiones de registro de nuevos sensores, necesario para llevar a cabo el plan de medidas.
    - Conocer si la simulación está corriendo o parada.
    - Peticiones para conocer en que profundidad se encuentra el AUV, y la batimetría en esa posición, para poder comprobar que el AUV no rebase la distancia de seguridad con el fondo.

Peticiones para medir desde cada uno de los sensores, tanto en el plan de medidas, como por orden expresa del **CCAS**.

CAPÍTULO 12.- *Interfaz Gráfica de Usuario*

Este capítulo ahonda en el diseño de la Interfaz Gráfica de Usuario. Dentro del mapa de los subsistemas que conforman el simulador, encaja en el perfil del subsistema de tipo **CIG** (**Cliente Interfaz Gráfica**), introducido en el **Apartado 8.2**.

Este subsistema aporta un programa gráfico sencillo que oculta la gran complejidad que pueda llevar el sistema. Así, ofrece a los usuarios:

- Una herramienta para la monitorización de la simulación en tiempo real: fiel reflejo gráfico de lo que sucede en el Servidor del Entorno.
- Una herramienta para configurar el servidor y comandarlo de forma sencilla: empezar, parar una simulación, etc.
- También, de forma accesoria, incorpora módulos para crear de forma gráfica Agentes Simulados (**AS**), como los prototipados en el **Apartado 11.4**. También módulos para comandar y gestionar estos **AS** desde la propia interfaz.

En el capítulo, se describe en forma de análisis el diseño ideal de la aplicación para cumplir con sus objetivos, así como se da una descripción modular de los componentes que conforman esta aplicación.

Finalmente se presenta el prototipo analizado, diseñado e implementado, que si bien no pretende ser la Interfaz final, sí permite, a modo de prototipo, reflejar el estado del Servidor del Entorno en cada momento.

12.1.- Diseño Funcional para la Interfaz Gráfica de Usuario.

En este apartado se describe la Interfaz Gráfica de Usuario que permite aprovechar todas las funcionalidades que ofrece el Simulador. Se muestran imágenes que, a modo de prototipo, permite validar la adecuación de la interfaz propuesta (pues no ha sido implementado finalmente).

La aplicación gráfica se compone de los elementos mostrados en la Figura 173:

- **Barra de Tareas Global**: Permite acceder a las distintas acciones y servicios que ofrece la interfaz gráfica.

- **Barra de Iconos Interna**: presenta algunos iconos para acceder directamente a las acciones y servicios antes nombrados.
- **Escritorio**: La interfaz va a orientarse a un **sistema de ventanas**: dado que la aplicación va a ofrecer distintas vistas de la simulación, centradas en distintos aspectos, el usuario puede abrir cuantas vistas (ventanas internas) desee, incluso varias del mismo tipo para focalizar la simulación en puntos distintos.
- **Ventanas internas al Escritorio**: Cada ventana es una vista, y es un subsistema completo, con su propia Barra de Tareas Global y Barra de Iconos para interaccionar con la Vista.
- **Barra Inferior de Estado**: Permite conocer cierta información de estado de la simulación, como puede ser el tiempo simulado, si la simulación está ejecutándose o no, si estamos conectados a un servidor del entorno (y a cuál), el usuario que se ha conectado a la aplicación, etc.

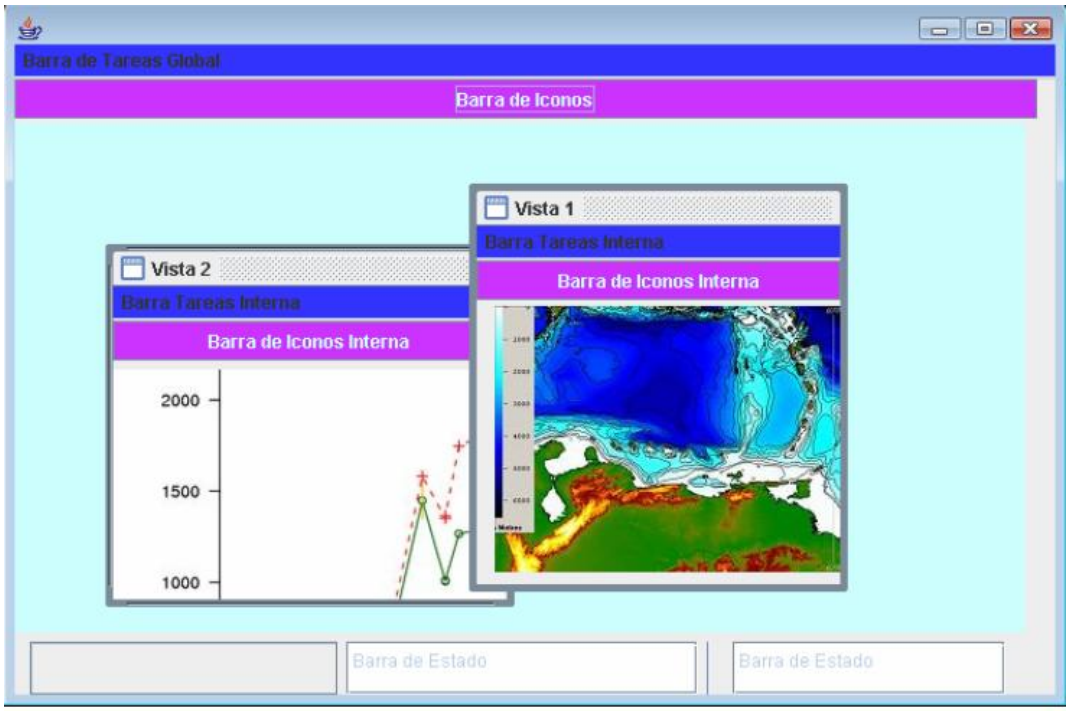


Figura 173: Interfaz Gráfica de Usuario. Pantalla principal.

Entrada de Usuario

Para comenzar a usar la aplicación, el usuario debe insertar un nombre de usuario y contraseña en la pantalla mostrada en la Figura 174.

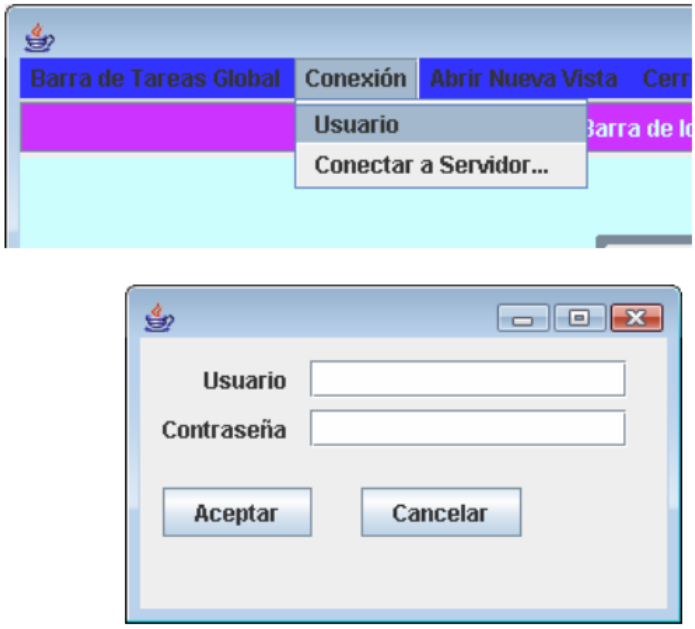


Figura 174: Pantalla de login.

Conexión de Servidor

Una vez registrado, el usuario debe conectarse a un servidor del Entorno en la pantalla mostrada en la Figura 175, para que pueda llevarse a cabo la simulación. Para ello debe especificar obligatoriamente la cadena de conexión al servidor, y el nombre de usuario. Opcionalmente, si el servidor requiere contraseña, también debe especificar la contraseña.

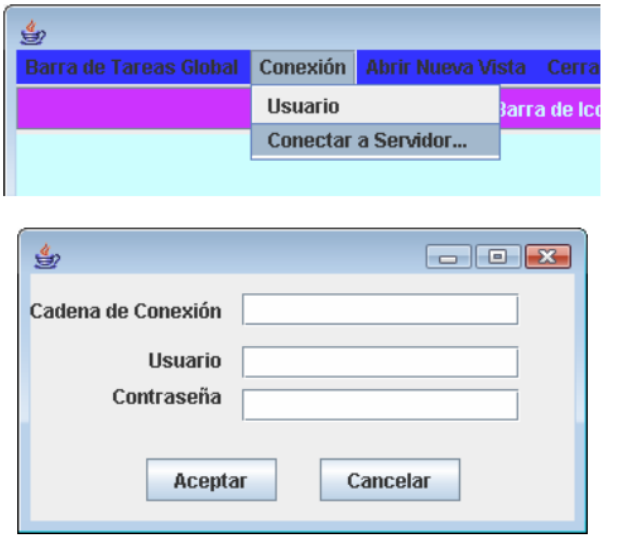


Figura 175: Pantalla de conexión con Servidor del Entorno.

Salir de la Aplicación

Un menú deja salir de la aplicación al usuario, mostrado en la Figura 176

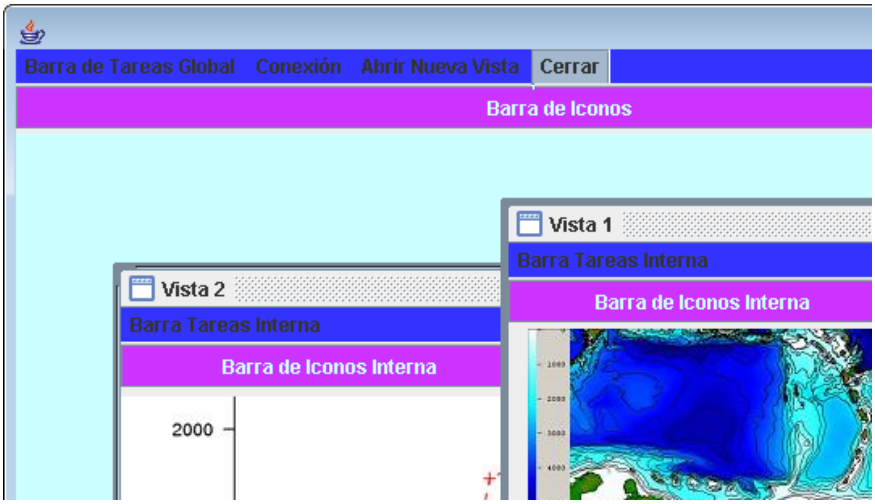


Figura 176: Menú para salir de la aplicación.

Abrir Nuevas Vistas

Desde el mismo menú (o barra de iconos) mostrado en la Figura 177 se listan los cinco tipos de vista que se pueden abrir. Al picar en cualquiera de ellas, se crea una nueva ventana en el escritorio con la información para esa vista. Las desglosamos a continuación.

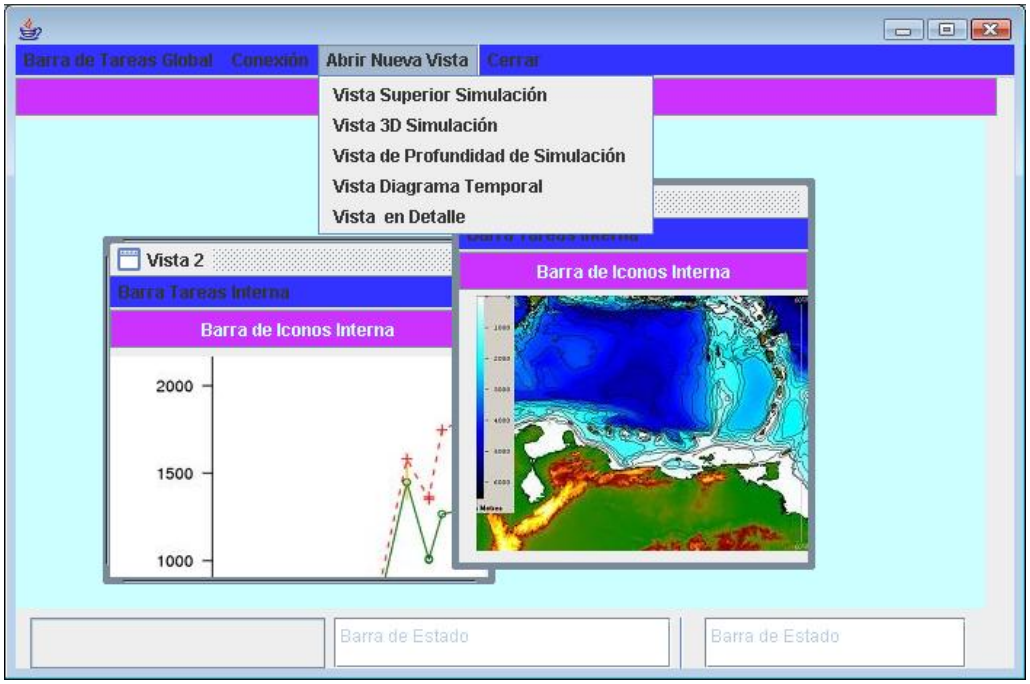


Figura 177: Distintas vistas que se pueden abrir de la simulación.

Las vistas **Diagrama Temporal** y **en Detalle** son vistas centradas en un único Agente Simulado. Es por ello que al seleccionar cualquiera de las dos, nos muestra una pantalla (como la de la Figura 178) donde tenemos que elegir al Agente Simulado en que queremos centrar la vista.

Figura 178: pantalla de selección de Agente Simulado.

Los Agentes Simulados son representados en base a un eje cartesiano en las distintas vistas, como se muestra en la Figura 179

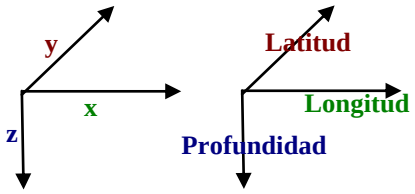


Figura 179:Eje cartesiano de representación de Agentes Simulados

Vista Superior de Simulación

Es una vista en 2 dimensiones ortográfica superior, centrada en el eje Y, X (es decir latitud-longitud), donde se representa básicamente:

- La **batimetría del terreno**: esto se muestra mediante líneas que definen los contornos de los picos de profundidad, y colores indicativos de la profundidad. En la leyenda se puede hacer una explicación de estos colores de profundidad.

- Los **Agentes** que se están simulando, representados de forma puntual, y con su vector de dirección, y pudiendo asignar colores para visualizarlos.

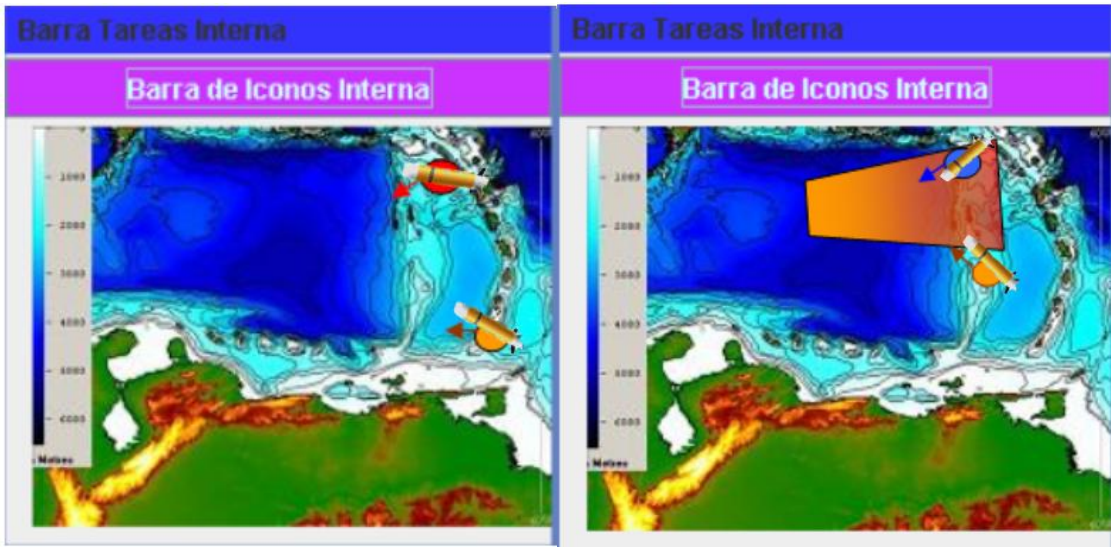


Figura 180: En la izquierda, sólo se muestra batimetría y agentes. En la derecha, además, gradiente de temperatura.

Se muestra un ejemplo en la Figura 180, donde cada AUV está rodeado con un círculo y una flecha que indica su dirección de movimiento.

En esta ventana básica se dan datos opcionales:

- Posibilidad de mostrar un **grupo de Agentes Simulados** seleccionados, incluso sólo uno, o ninguno (para ver sencillamente el terreno). Para esto la ventana tiene un menú superior desplegable local a la ventana, que permite abrir una pestaña donde se listan los Agentes Simulados, y el usuario pueda decidir cuales mostrar en esta ventana concreta, y cuales no.

- Posibilidad de quitar la **batimetría**, para que otros elementos superpuestos se vean más claros.

- Posibilidad de añadir a una ventana dada información de la distribución de algún parámetro fisicoquímico, como pueden ser corrientes, salinidad, temperatura, etc. También representación de distribuciones de suelos marinos, para detectar donde hay algas, rocas, etc, en misiones que lo requieran. Estas

distribuciones se muestran con códigos de colores y marcas, que son explicados en una leyenda global al simulador.

- Posibilidad de visualizar o no **obstáculos** que puedan haber, así como zonas prohibidas en el mapa.

- Posibilidad de representar la **traza** de cada Agente Simulado, mostrando información sobre la misión, como puntos por los que va pasando, waypoints venideros, puntos de comunicación, puntos de toma de muestras. Para cada uno se usa una marca especial explicada en una leyenda global al simulador. Por ejemplo, para un waypoint un aspa **X**, y para un punto de comunicación un asterisco *****. La traza se puede hacer con inclusión de errores, lo que implica representar la traza real y la esperada según la misión.

- Posibilidad de representar **parámetros específicos de la misión**: como área que tiene que recorrer el Agente Simulado, o área límite que tiene el Agente Simulado para seguir un gradiente, etc.

Para todos estos datos, el modo de activar o desactivar datos a visualizar es muy similar: acceder a la barra de menús de la ventana concreta que se quieren añadir parámetros o quitar parámetros de visualización, y seleccionar alguna opción, para activar desactivar.

Se puede seleccionar de una lista qué agentes simulados se quieren visualizar o no en cada vista.

Esta ventana es local y configura sólo parámetros de la ventana de vista superior desde la que fue activada. Por tanto, es **modal**: evita que se pueda picar en ninguna otra ventana mientras está esta ventana de elección de Agentes Simulados activada. Esto evita que el usuario confunda a qué ventana de visualización corresponde una ventana que configura los Agentes Simulados que se visualizan. Se muestra un ejemplo en la Figura 181.

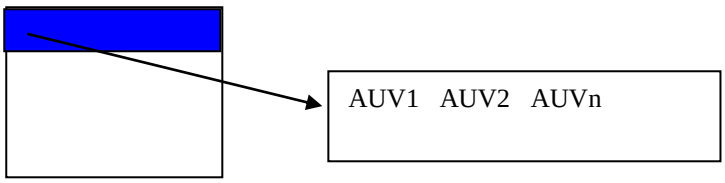


Figura 181: Lista de AUVs seleccionables en la vista.

Nótese que para los dos últimos puntos, cada Agente Simulado tiene su propia misión, y por tanto sus propios datos. Es probable que al usuario le interese mostrar la misión o traza de un Agente Simulado concreto de todos los que se muestran en el simulador, para que no se entremezclen.

Para activar o no la traza y parámetros el usuario sólo tiene que picar con el botón derecho sobre el Agente Simulado y seleccionar qué datos se quieren visualizar. Se puede ver un ejemplo en la Figura 182.

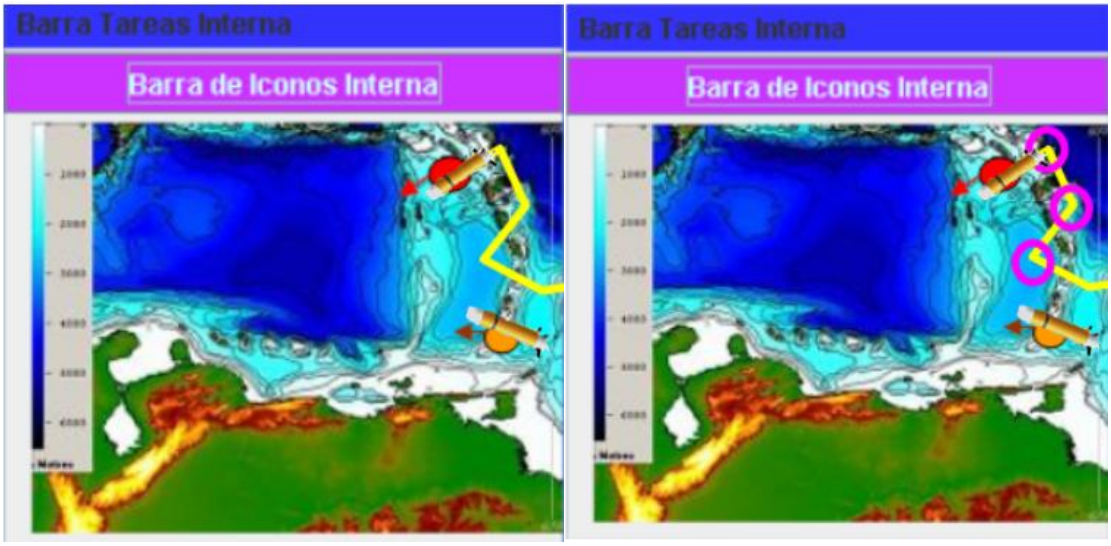


Figura 182: Traza de un sólo AUV a la izquierda, y traza con marcas a la derecha.

Al final se pueden tener varias ventanas para la misma simulación, una mostrando batimetría y Agentes Simulados, otra mostrando batimetría y un solo Agente Simulado, otra mostrando solo del Agente Simulado con su traza de la misión, y otra mostrando el Agente Simulado con su traza y con datos fisicoquímicos.

Internamente, en realidad hay una única situación en el simulador, pero que se visualiza de tantas formas como desea el usuario. Es imprescindible para esto por tanto el patrón de diseño **Controlador-Vista-Modelo** a la hora de implementar la Interfaz Gráfica.

Vista 3D de Simulación

Esta vista en principio es opcional, en función de las necesidades y capacidad de la máquina Java para sintetizar vistas tridimensionales. Utiliza los tres ejes X, Y, Z (longitud, latitud profundidad) para la representación. Representa varios Agentes Simulados a la vez y con una vista en perspectiva, que permita observar su proximidad y proximidad al terreno, además de ver la orografía del terreno de una manera más inteligible.

Básicamente permite lo mismo que en la Vista Superior. Además, facilita la modificación del punto de vista de la cámara virtual, que se traducen a cambios de coordenadas que se están viendo para el usuario, para poner la escena de la manera que mejor se entienda.

Debe tenerse en cuenta que también hay que aplicar un recorte de la escena, ya que mostrar simultáneamente todo el mapa batimétrico puede ser computacionalmente muy pesado. En este caso, el usuario puede recortar y mostrar sólo parte del terreno usando un menú específico. La vista 3D es la única, pues consume muchos recursos del ordenador y con una sola vista es suficiente para mostrar lo que queremos. Se puede ver un ejemplo en la Figura 183.

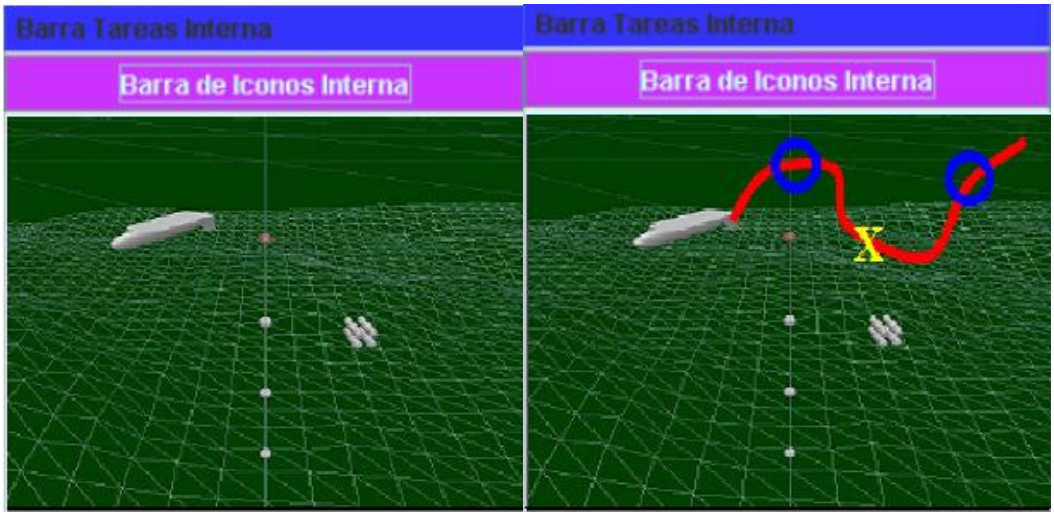


Figura 183: Vista en 3D, sin y con traza.

Vista en Profundidad de Simulación.

Se pueden elegir tantos Agentes Simulados u otros elementos de simulación (como obstáculos, ROVS, boyas, etc) como se desee para visualizar. Cada uno de ellos es representado por un icono del mismo rodeado por un color (a elección propia) y un vector de dirección. Se puede ver un ejemplo en la Figura 184.

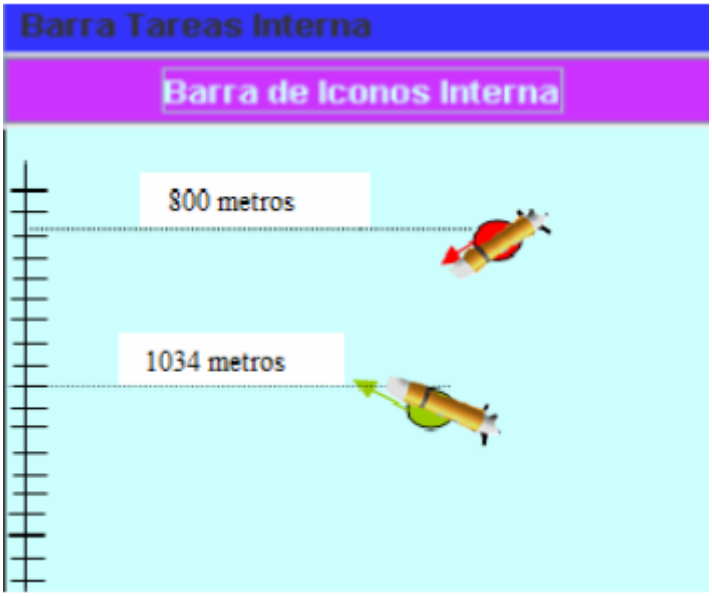


Figura 184: Vista en Profundidad de simulación.

Los ejes son T-Z (Tiempo: eje de abcisas-Profundidad: eje de ordenadas). La vista muestra una barra lateral con la escala de profundidades, desde profundidad absoluta que es el suelo, hasta profundidad 0 que es la superficie (configurable por el usuario la escala). También muestra la profundidad en números, y finalmente, el vector de dirección y velocidad, ahora no para conocer el rumbo al que se dirige, sino la tendencia del Agente Simulado, si es subir o bajar.

Opcionalmente, se puede añadir una traza de las profundidades alcanzadas por los Agentes Simulados, que son representadas superponiéndose a la imagen anterior. La traza puede incorporar puntos o marcas, como las profundidades a las que se alcanzan los waypoints, los puntos de comunicación, se toman muestras, etc.

También se pueden señalar marcas, de forma que muestre en cada altura que evento se ha producido (llegar a un waypoint, comunicación, etc), como se ve en la Figura 185.

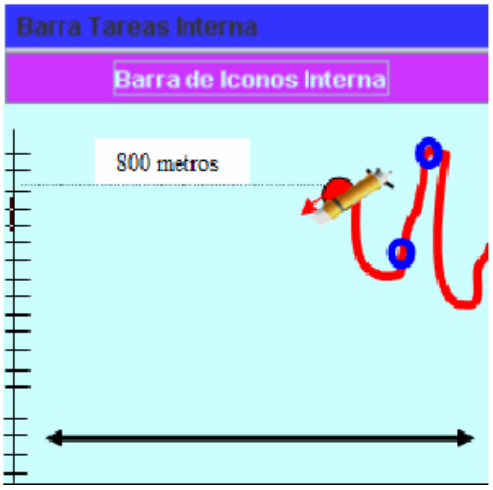


Figura 185: Traza de profundidades, con marcas temporales.

Es importante ver que la representación anterior era atemporal, mientras que al representar una traza, requiere que el eje de abscisas sea un eje temporal. Con incluir una barra inferior para el tiempo basta.

Vista en Detalle Agente Simulado

Permite ver datos concretos de un Agente Simulado, como se muestra en la Figura 186. Consiste en una tabla donde cada fila de la misma es un parámetro a visualizar del Agente, y las columnas son:

- Nombre del parámetro:
- Tipo de Parámetro:
- Orden del Parámetro: Orden que ocupa en la tabla
- Valor del Parámetro:
- Unidad del Parámetro:

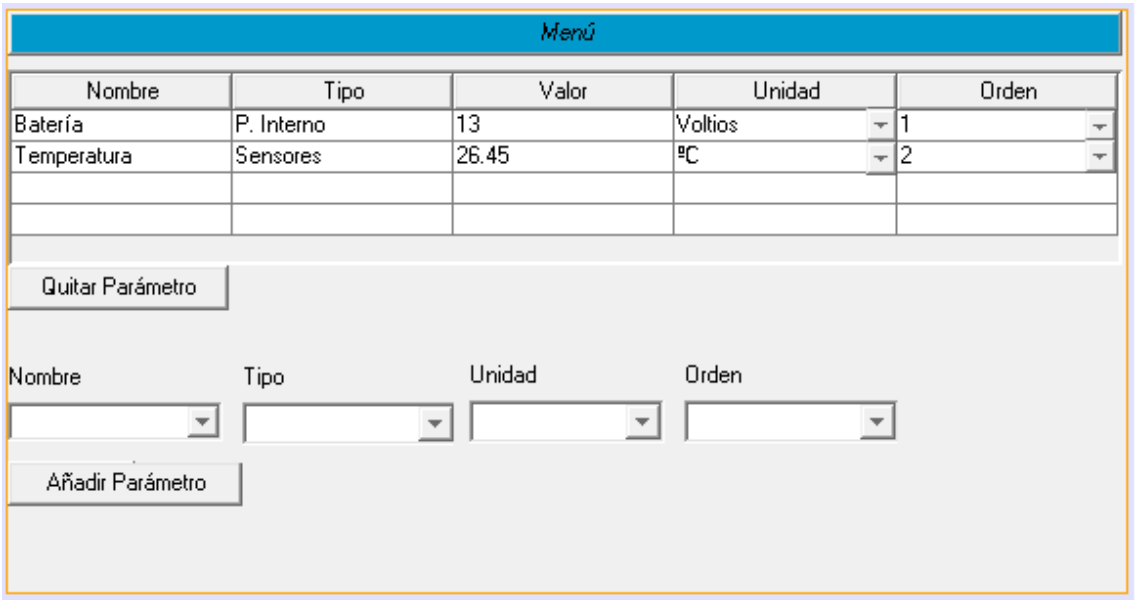


Figura 186: Vista en detalle del Agente Simulado.

Además, la vista permite cambiar la Unidad y el Orden de forma sencilla (por ejemplo con un combobox, como se aprecia en el prototipo) y añadir y eliminar parámetros de la vista de forma sencilla. Por ejemplo, en el prototipo vemos que la manera de añadir parámetros es rellenar los comboboxes y pulsar **Añadir**. Para quitarlo, hay que seleccionar el parámetro de la tabla, y pulsar en **Quitar**.

Vista en Diagrama Temporal

Esta vista está centrada en un único Agente Simulado. Permite ver una gráfica temporal de distintos parámetros, incluso superponerlas.

Lo que se va a representar es:

- Una línea por cada Parámetro del Agente Simulado que se quiera representar la línea temporal de su misión. El usuario, puede seleccionar desde la barra de menú de la ventana qué parámetros mostrar. Si se elige representar varios, se superponen las líneas, lo que permite comparar datos.
- El eje de abscisas es el tiempo (que puede ser configurado por el usuario el momento inicial y final), y el de ordenadas la magnitud a representar gráficamente.

- Adicionalmente, se pueden cambiar las características de la línea (color, tipo de línea, etc). Por ejemplo, la Figura 187.

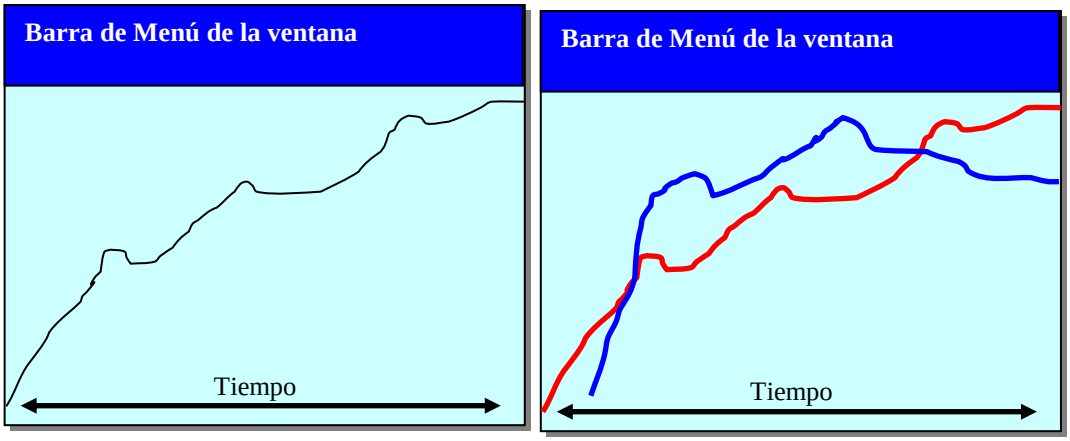
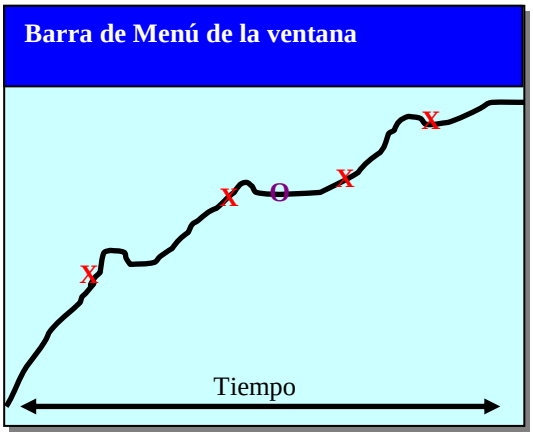


Figura 187: Vista de Diagrama Temporal, con uno (izquierda) o varios (derecha) parámetros

- Finalmente, a cada línea se le pueden añadir datos de la misión (opcionales en cada línea, por tanto picar en cada línea para poder añadirlos), como señalizar con marcas los instantes en los que el AUV alcanzó un waypoint, realizó una comunicación, o tomó ciertas muestras.



Vista de Leyenda

Esta vista se ejecuta desde cualquiera de las vistas anteriores que requiera leyenda (esto es, **Vista Superior**, **Vista 3D, Vista Profundidad** y **Diagrama Temporal**). No es más que una tabla con dos columnas (como se muestra en la Figura 188) donde:

- La primera columna señala el símbolo.

- La segunda columna, descripción del símbolo.

Cada fila es cada símbolo a describir en la leyenda.

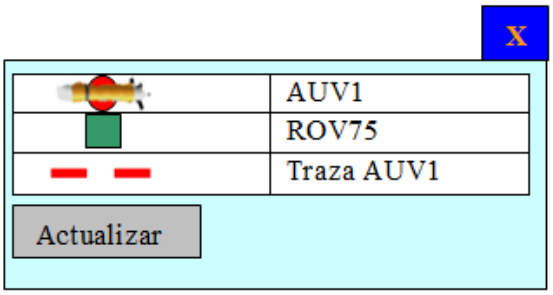


Figura 188: Ejemplo de leyenda.

Es una ventana muy sencilla, que tan sólo requiere de un botón para actualizar la vista.

12.2.- Diseño General del programa

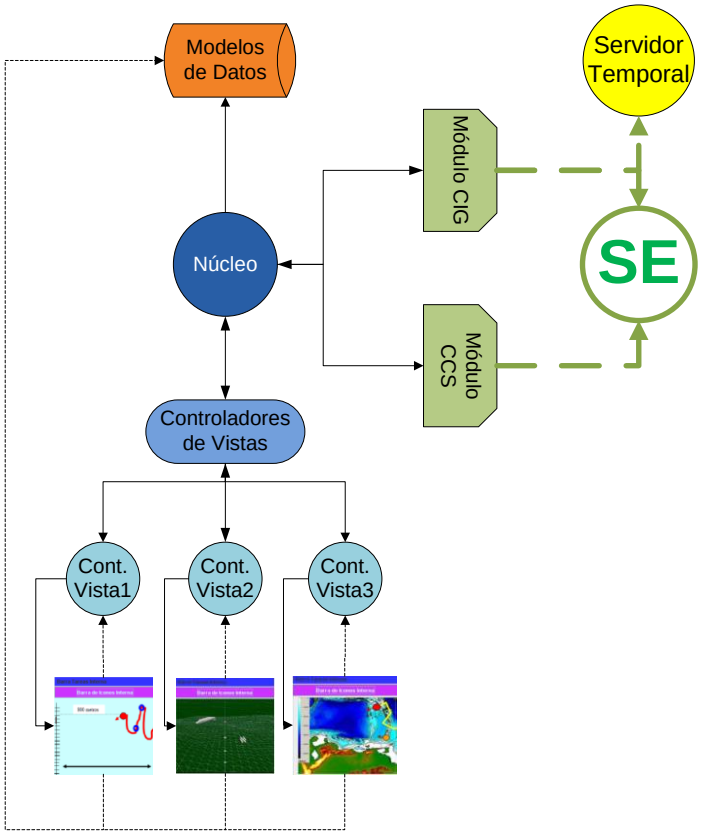


Figura 189: Módulos que conforman la Interfaz Gráfica de Usuarios.

Se muestra el diseño lógico de la Interfaz Gráfica de Usuario en la Figura 189, siguiendo la descripción del **apartado 12.1**. Los componentes básicos siguen el patrón de diseño Modelo Vista Controlador (MVC). La aplicación tiene:

- Un conjunto de **Modelos de Datos**, que sirven para representar los Agentes Simulados, los mapas y distribuciones visualizadas, las configuraciones realizadas, etc.

- Una zona central de **Controladores** que se organizan jerárquicamente:

    - **Núcleo**: tiene el lazo de control principal del programa. En cada ciclo de control es el encargado de tomar del servidor todas las actualizaciones de estado de los AS y de los modelos de parámetros fisicoquímicos, actualizar las vistas (a través de sus **Controladores de Vistas**). También tiene que atender a otras peticiones del usuario desde la interfaz.

    - **Controladores Vistas**: coordina que las órdenes desde el Núcleo lleguen al controlador de vista concreta, y viceversa.

    - **Controlador de Vista1, 2,..n:** cada uno es el Back-End de la interfaz gráfica correspondiente a cada vista. Gestiona los eventos del usuario en la vista, y lo notifica, al **Núcleo**, que indirectamente hace las peticiones al **Servidor del Entorno**, o modifica los **Modelos de Datos** Correspondientes.

- **Vistas**: son cada una de las ventanas que el usuario puede abrir dentro del escritorio, más la vista general del programa, como las barras de menús globales, etc.

Siguiendo el patrón MVC, cada vista es un *Observador* de uno o varios modelos *Observables*, con lo que cualquier cambio en los modelos es notificado a todas las vistas que toman ese modelo (por eso es marcado con línea discontinua). Así podemos tener varias vistas para ver a los Agentes Simulados, con distinto Zoom incluso, pero cuando se modifica la posición de un Agente Simulado se actualizan todas las vistas.

Cualquier acción del usuario en la vista es gestionada por el controlador de la vista, que a su vez puede pasar el control al núcleo si hace falta modificar algún modelo de datos

como consecuencia, o si hace falta conectar con el Servidor del Entorno en consecuencia, por ejemplo, para gestionar un cambio de configuración que el usuario pide desde la vista.

Por otra parte, el controlador **Núcleo** gestiona la sincronización con el Servidor Temporal, con lo que en cada ciclo exige una actualización de todas las vistas a través del controlador de cada vista.

Finalmente, pero no por ello menos importante, el **Núcleo** hace uso de dos módulos, que funcionan como subsistemas de la arquitectura del simulador ***SUBES***, explicada en el **Apartado 8.2**:

- **Módulo CIG (Cliente Interfaz Gráfica):** Es el módulo por excelencia de la interfaz para conectar tanto con el Servidor del Entorno (**SE**), como con el Servidor Temporal en el caso de que éste no fuera un módulo interno del **SE**. A través del mismo el núcleo podrá solicitar:
    - Gestionar la simulación.
    - Pedir el estado de los Agentes Simulados.
    - Pedir información batimétrica.
    - Pedir información de los parámetros fisicoquímicos modelados en el **SE**.
    - Realizar la sincronización temporal.
- **Módulo CCS (Cliente Configurador del Servidor):** módulo con el que el usuario podrá hacer cambios en la configuración del Servidor del Entorno desde la interfaz gráfica.

12.3.- Prototipo Implementado de Cliente Interfaz Gráfica (CIG)

Durante el desarrollo de este proyecto se ha implementado un prototipo que cumple con un subconjunto de las especificaciones detalladas en el **Apartado 12.1**. Esta versión simplificada de CIG ha servido para verificar y poner a prueba el Servidor del Entorno.

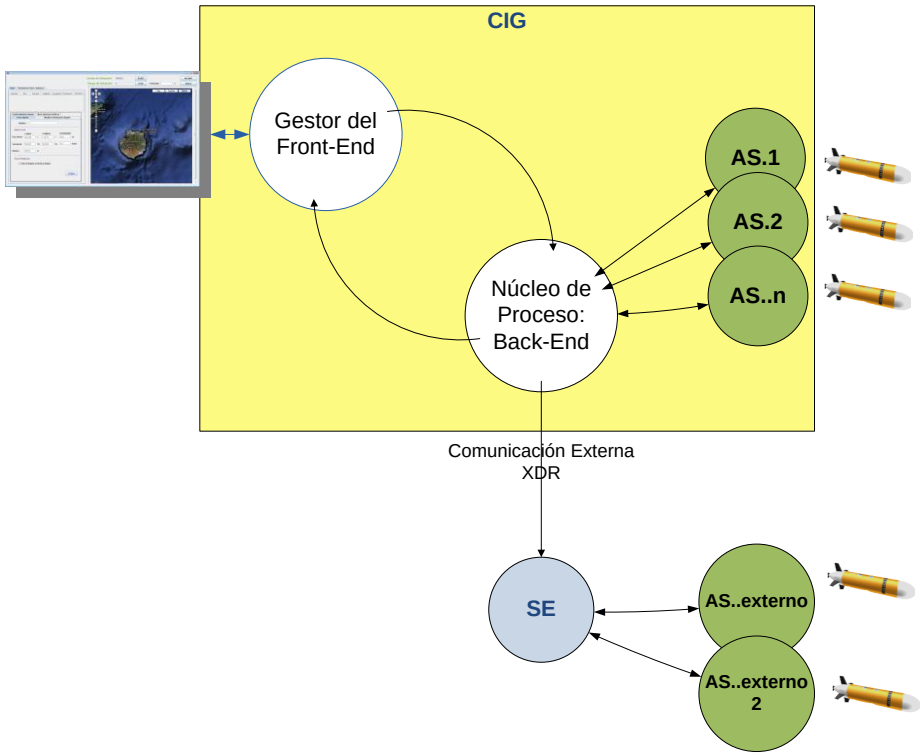


Figura 190: Diseño del CIG prototipo implementado.

Como vemos en la Figura 190, el CIG implementado tiene dos hilos de ejecución desacoplados principales (Gestor del Front-End y Núcleo de Proceso:Back-End), y tantos hilos como AS creados desde la propia interfaz:

- **Gestor del Front-End**: Se encarga de gestionar todos los eventos activados por el usuario en la interfaz gráfica de usuario. Todo evento que requiera realizar una petición al Servidor del Entorno o cierto proceso algorítmico pesado, es desviado al Núcleo de proceso.

  El objetivo es que ningún proceso pesado sea ejecutado en el propio hilo, ya que en tal caso el usuario notaría que la interfaz no responde durante el tiempo de proceso, disminuyendo la experiencia del usuario.

  Una vez el núcleo ha procesado el evento, devuelve la respuesta, para que el hilo del Gestor del Front-End gestione y monitorice el resultado adecuadamente (mostrarlo por pantalla). Se ve un esquema del funcionamiento en la Figura 191.

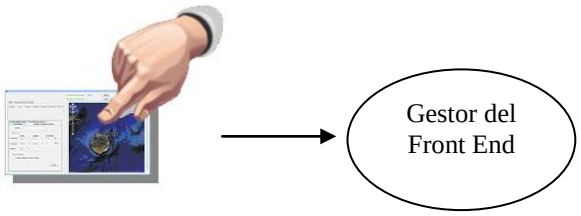


Figura 191

- **Núcleo de Proceso, Back-End**: este hilo tiene el lazo de control principal del programa, lo cual implica:
    - De forma automática, en cada ciclo de simulación, actualizar la información y enviar al **Gestor del Front-End** la información necesaria para que refresque la interfaz gráfica. La actualización de la información se lleva a cabo realizando las correspondientes peticiones al **Servidor del Entorno** (**SE**).
    - Atender todas las peticiones encoladas por el **Gestor del Front-End**, gestionando la respuesta adecuadamente y enrutándola al **Gestor del Front-End** para que refresque la interfaz en su caso. Las peticiones pueden ser de diversa índole:
        - Cálculo y proceso.
        - Peticiones al **SE**.
        - Comandos a **AS** internos al **CIG**.

- **Gestión de Agentes Simulados (AS) internos al CIG**: el usuario de la interfaz puede, desde la misma, crear tantos **AS** (descritos en el **Apartado 11.4**) como estime oportuno para su simulación.
    - Cada AS es un hilo independiente, que puede ser comandado desde el **Núcleo de Proceso** tras eventos activados por el usuario en la interfaz, como se ve en la Figura 192

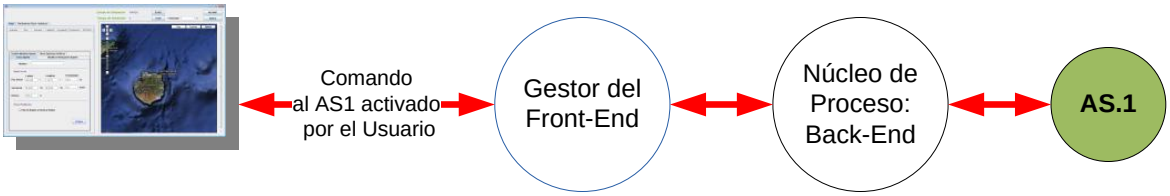


Figura 192: Traza de comando enviado al AS1 por el usuario.

- Cada **AS** tiene su propio Lazo de Control tal como se describió en el **Apartado 11.4**. Dentro del mismo, admite el encolamiento de peticiones de comandos, que es lo que en el **Apartado 11.4** llamamos "**peticiones del cliente controlador externo CCAS"**.

- Puede haber otros **AS** externos al entorno del **CIG** pero conectados al mismo Servidor del Entorno y por tanto participando de la misma simulación (por ejemplo un **AS** que en realidad sea un módulo dentro de un AUV real hardware-in-the-loop).

  El prototipo sólo contempla el control de los **AS** creados internamente mediante un sistema de mensajería interna, y no de **AS** externos (lo que requeriría un protocolo con formatos como el XDR como el descrito en el **Apartado 8.4)**. Sin embargo sí que puede pedir al **SE** información de los mismos para monitorizarlos.

  La comunicación de la interfaz con otros **AS** externos requeriría, por tanto, un módulo para intercambio de paquetes en XDR, que si bien es perfectamente factible dada la estructuración modular de la aplicación, no se ha abordado en el presente proyecto.

Sólo añadir que al iniciarse la aplicación, lee un fichero XML con información sobre el Servidor al que debe conectarse (ip, puerto), clave, nombre con el que registrarse, nombre completo, etc. Se puede apreciar en la Figura 193.

La interfaz de usuario no va a integrar un cuadro de control para gestionar esta conexión, por lo que el usuario tendrá que modificar el fichero XML asociado antes de arrancar la Interfaz Gráfica de Usuario.

```
<?xml version="1.0" encoding="UTF-8"?>
<Parameters>
  <Server name="localhost" port="1983" password="uno"/>
  <Gui name="GUIsynchl.0" localhost="GUIsynchl.0@synch.iusiani.ulpgc.es" />
</Parameters>
```

Figura 193: Configuración de conexión de la Interfaz Gráfica de Usuario.

12.3.1.- Descripción de la interfaz de usuario.

En el documento anexo al proyecto **Manuales de Usuario** se podrá encontrar un completo manual de usuario de la aplicación. Este apartado sólo pretende ser una breve introducción.

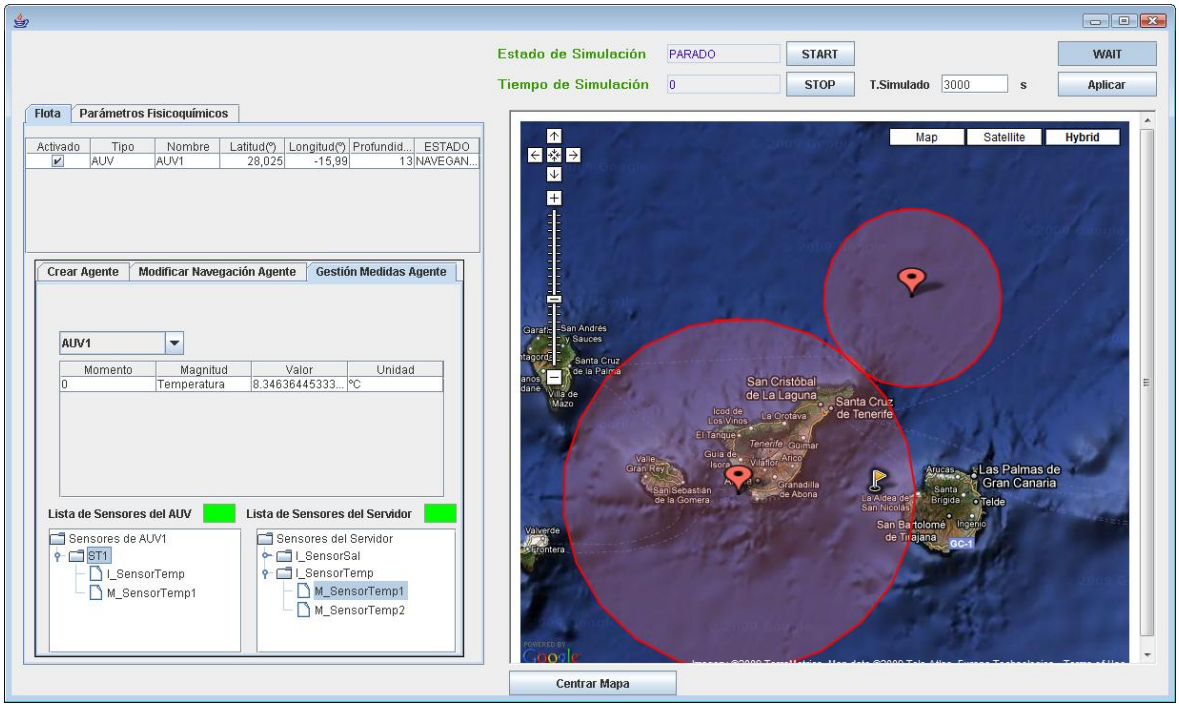


Figura 194: Interfaz Gráfica de Usuario implementada.

La interfaz mostrada en la Figura 194 pone a disposición del usuario las herramientas suficientes para poder gestionar y monitorizar la simulación. Ha sido programada en Java, acorde a las especificaciones detalladas en el **Capítulo 4**.

Destacan dos paneles:

- Panel *derecho* de monitorización y control general de la simulación:
    - Mediante un mapa integrado usando la librería **JDIC** para la integración de navegadores web en aplicaciones Java, se ha integrado un mapa *Google maps*, donde se monitorizará la simulación: los **AS**, los parámetros fisicoquímicos, etc.
    - En la parte superior, ciertos controles permiten gestionar la simulación:
        - Ver el estado de la simulación y el tiempo de simulación
        - Comenzar y parar la simulación.
        - Ver y modificar el tiempo de ciclo: cada ciclo de simulación a cuanto tiempo de simulación corresponde.
        - Gestionar con el botón "NO WAIT" si los ciclos del simulador tienen que tener una duración real constante, o puede ser variable dependiendo de la rapidez de respuesta de los clientes.
    - En la parte inferior, un botón para centrar el mapa en todos los AUVs que el usuario quiera visualizar.

- Panel *izquierdo* para la gestión de la flota y los parámetros Fisicoquímicos:

  De la plantilla de flota destaca, en la parte superior, una tabla donde se muestran todos los **AS** que tiene actualmente registrados el SE y por tanto, participan de la simulación. De cada uno se muestra información, como posición o el estado. En la implementación hecha, se manejan dos estados (escalables en futuros desarrollos):
    - Navegando: los sistemas del **AS** están correctos, y puede seguir su plan de Navegación y de Medición.
    - No Responde: el **AS** experimenta problemas, por ejemplo, cuando desciende lo suficiente como para chocar con el fondo. Quedan inactivos su Plan de Navegación y de Medición.

En la parte inferior, encontramos 3 cuadros de control:

En la Figura 195 se ve el cuadro para crear nuevos **AS**, donde podemos especificar el nombre, posición inicial, velocidad inicial, carga de la batería, y posibilidad de planes de navegación predefinidos.

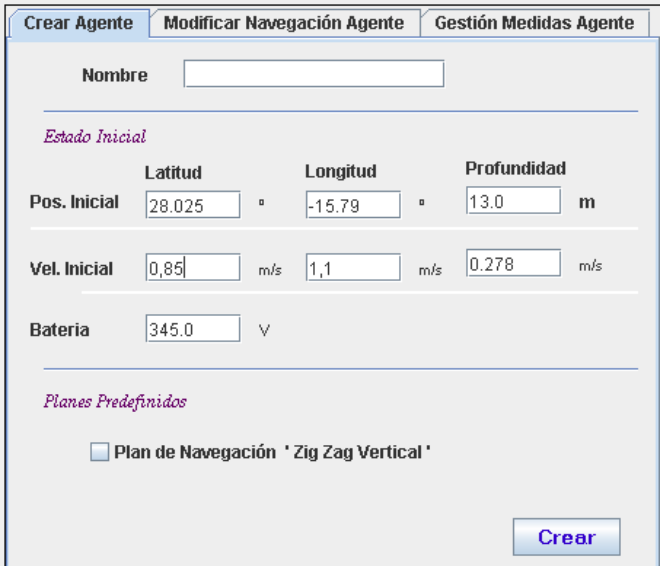


Figura 195: Panel "Crear Agente".

En la Figura 196 está el cuadro para modificar el vector de velocidad que sigue el **AS**, en sus tres coordenadas, lat / lon / Prof. También se puede eliminar el **AS**.

Figura 196: Panel para "Modificar Navegación del Agente".

En la Figura 197 la Cuadro para gestionar el sistema sensorial de cada **AS**. Se pueden seleccionar modelos en el cuadro inferior derecho, y asignarlo al **AS** dándoles un nombre.

En el cuadro inferior izquierdo podemos ver todos los sensores del **AS**. Se pueden seleccionar y ejecutar acciones sobre los mismos mediante un menú contextual (medir del sensor, o eliminarlo del **AS**, etc.).

En la tabla central se puede ver el histórico de medidas tomadas de todos los sensores del **AS**.

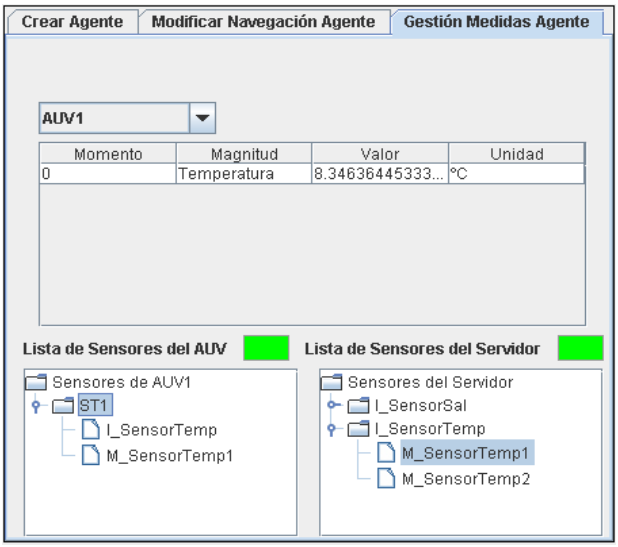


Figura 198: Panel para "Gestión de Medidas del Agente".

En cuanto a la plantilla de Parámetros Fisicoquímicos, como se ve en la Figura 199 en la tabla superior se muestran los distintos modelos de parámetros fisicoquímicos submarinos que están activos en el servidor.

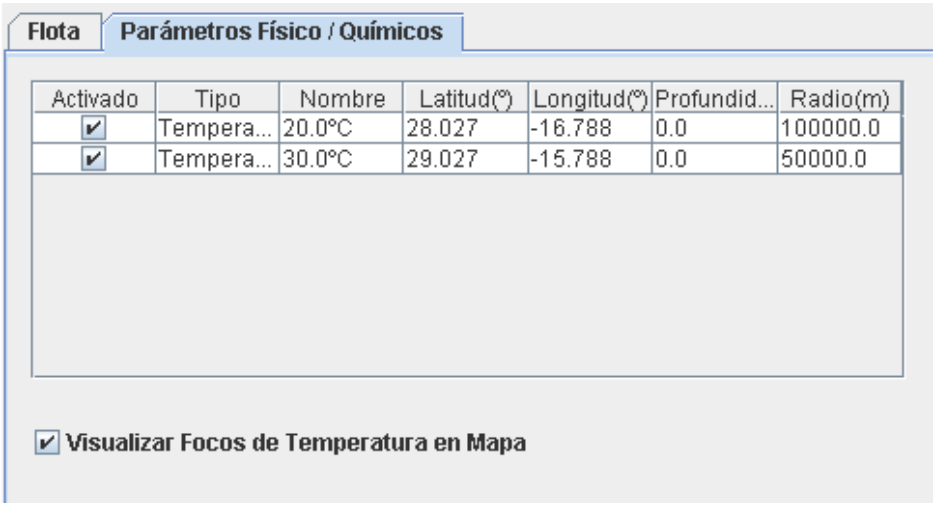


Figura 199: Plantilla de Parámetros fisicoquímicos.

En la parte inferior, permite al usuario controlar si quiere que se muestren los Focos de Temperatura sobreimpresos en el mapa *Google maps*, o no.

12.3.2.- Google Maps para monitorizar el Servidor del Entorno.

Si bien, Java proporciona muchas APIS para dibujar y representar mapas, al final para el presente prototipo se ha optado por *Google Maps* por diversos motivos.

Visual y estéticamente, su representación de imágenes de satélite e integración para permitir al usuario hacer zoom y navegar por el mapa, es insuperable. Esto ha ahorrado mucho trabajo en el presente proyecto, evitando programar la integración de los mapas batimétricos y geográficos representados con APIS de Java.

Por otra parte, se trata de una herramienta ligera. Haber programado mediante las APIS de Java la representación de los mapas sin duda habría dado como resultado un prototipo más pesado y con menos grado de detalle, aparte de que se hubiera prolongado el tiempo de implementación del prototipo.

Han facilitado también la decisión la fácil integración en aplicaciones Java, embebiendo un navegador en los paneles de Java, y la completa API que proporciona *Google* para poder incrustar en el mapa marcas, objetos, representación de zonas, trayectorias, incluso paneles de ayuda e información del estado de cada objeto o marca incrustado.

Sin embargo, introduce en el presente proyecto una importante limitación: para poder funcionar correctamente el prototipo, el ordenador debe tener acceso a Internet y por tanto al servidor de *Google Maps*.

Integración en el Prototipo CIG.

La integración de Google Maps se hizo mediante el patrón de diseño “Observador Observable”. A continuación se muestra gráficamente en la Figura 200 este patrón:

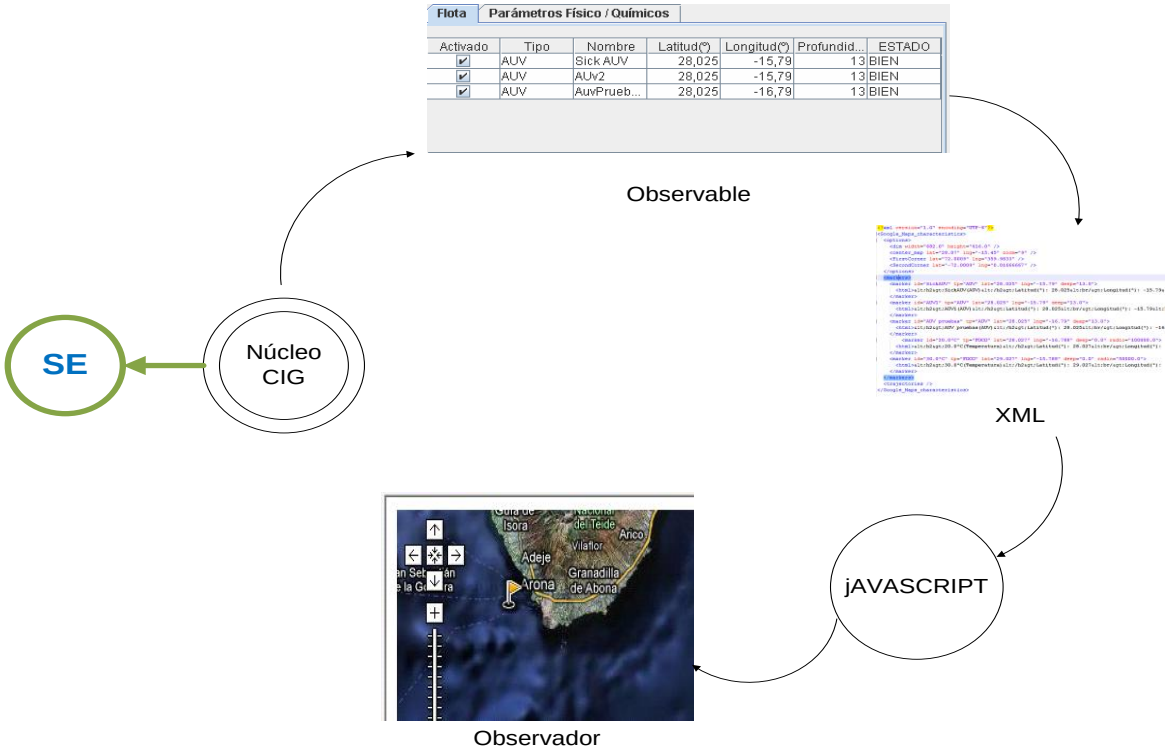


Figura 200: Flujo de control de funcionamiento de Google maps.

Para embeber el mapa *Google maps* en el prototipo, se utiliza la clase Java **PanelMapa.class**, que es de tipo Panel y utiliza la librería JDIC para incrustar aplicaciones (en este caso un navegador web) en paneles de Java. La página web que se embebe es un fichero html local que se alimenta de *Google maps*.

Este **PanelMapa.class** se registra como *Observador* de la tabla que muestra información de todos los **AS** que se encuentran en la simulación.

Por tanto esta tabla, de la clase **ModeloTabla**, implementa la interfaz *Observable* de Java. El flujo del proceso es el siguiente:

1) Como veremos en el **Apartado 12.3.3**, dentro del lazo de control del núcleo, en cada ciclo de simulación el núcleo pedirá la información al **SE** sobre todos los **AS** que están siendo simulados.

2) Cada vez que recibe la información, actualiza la posición de cada **AS** en la tabla de tipo **ModeloTabla**.

3) Al actualizar los registros de la tabla, se invoca automáticamente el método **notify**, que ofrece la interfaz *Observable* de Java. Este método notifica a todos los *Observadores* registrados, en este caso **PanelMapa**.

4) Automáticamente, se ejecuta el método "**update"** de **PanelMapa**, al ser notificado. El panel entonces muestra la nueva posición y los posibles nuevos **AS** en el mapa.

Para refrescar la vista de *Google maps*, se siguen los siguientes pasos:

- A) Se crea un fichero XML local al prototipo **CIG**, toda la información a mostrar en el mapa *Google*. Se muestra un ejemplo en la Figura 201:

```
<?xml version="1.0" encoding="UTF-8"?>
<Google_Maps_characteristics>
  <options>
    <dim width="682.0" height="616.0" />
    <center_map lat="28.07" lng="-15.45" zoom="9" />
    <FirstCorner lat="72.0009" lng="359.9833" />
    <SecondCorner lat="-72.0009" lng="0.01666667" />
  </options>
  <markers>
    <marker id="SickAUV" tp="AUV" lat="28.025" lng="-15.79" deep="13.0">
      <html>&lt;h2&gt;SickAUV(AUV)&lt;/h2&gt;Latitud(°): 28.025&lt;br/&gt;Lo
    </marker>
    <marker id="AUV1" tp="AUV" lat="28.025" lng="-15.79" deep="13.0">
      <html>&lt;h2&gt;AUV1(AUV)&lt;/h2&gt;Latitud(°): 28.025&lt;br/&gt;Longi
    </marker>
    <marker id="AUV pruebas" tp="AUV" lat="28.025" lng="-16.79" deep="13.0">
      <html>&lt;h2&gt;AUV pruebas(AUV)&lt;/h2&gt;Latitud(°): 28.025&lt;br/&g
    </marker>
  </markers>
  <trajectories />
</Google_Maps_characteristics>
```

Figura 201: Fichero XML para imprimir en el mapa los distintos AUVs. .

Como observamos en la Figura 201, en el apartado de **markers**, hay un marcador <**marker**> por cada entrada de la tabla de flota. Para cada uno se puede ver en los atributos la posición y el nombre del **AS**.

- B) Un conjunto de ficheros Javascript contienen los métodos suficientes para analizar léxica y sintácticamente el fichero XML anterior y usar la API de *Google maps* para borrar el mapa anterior, y dibujar el nuevo mapa con un icono por cada AS en la posición según marca el fichero XML. Concretamente, se invoca el método **actualizar** de los ficheros Javascript, que desencadena todo el proceso de actualización.

- C) Tras ejecutarse el método **actualizar** escrito en Javascript, el usuario ya puede observar en el mapa las actualizaciones. Por cada **AS**, podrá ver una bandera que señaliza su posición.

El mismo flujo sigue el programa con la tabla de la plantilla "Parámetros Físico / Químicos": siendo una tabla de tipo **ModeloTabla**, y siendo un *Observable*, notifica a su *Observador* que es **PanelMapa,** cuando hay cualquier cambio en los datos de la tabla (se muestran los modelos de los parámetros del servidor, etc.), de forma que se genera un fichero XML (véase Figura 202) con los datos de la tabla, y el Javascript actualiza la vista gráfica del *Google maps*.

```
<?xml version="1.0" encoding="UTF-8"?>
<Google_Maps_characteristics>
  <options>
    <dim width="682.0" height="616.0" />
    <center_map lat="28.07" lng="-15.45" zoom="9" />
    <FirstCorner lat="72.0009" lng="359.9833" />
    <SecondCorner lat="-72.0009" lng="0.01666667" />
  </options>
  <markers>
    <marker id="SickAUV" tp="AUV" lat="28.025" lng="-15.79" deep="13.0">
      <html>&lt;h2&gt;SickAUV(AUV)&lt;/h2&gt;Latitud(°): 28.025&lt;br/&gt;Longitud(°): -15.
    </marker>
    <marker id="AUV pruebas" tp="AUV" lat="28.025" lng="-15.79" deep="13.0">
      <html>&lt;h2&gt;AUV pruebas(AUV)&lt;/h2&gt;Latitud(°): 28.025&lt;br/&gt;Longitud(°):
    </marker>
    <marker id="20.0°C" tp="FOCO" lat="28.027" lng="-16.788" deep="0.0" radio="100000.0">
      <html>&lt;h2&gt;20.0°C(Temperatura)&lt;/h2&gt;Latitud(°): 28.027&lt;br/&gt;Longitud(°
    </marker>
    <marker id="30.0°C" tp="FOCO" lat="29.027" lng="-15.788" deep="0.0" radio="50000.0">
      <html>&lt;h2&gt;30.0°C(Temperatura)&lt;/h2&gt;Latitud(°): 29.027&lt;br/&gt;Longitud(°
    </marker>
  </markers>
  <trajectories />
</Google_Maps_characteristics>
```

Figura 202: Fichero XML que imprime en el mapa marcadores para AUVs y focos de temperatura.

En el ejemplo de la Figura 202 se ven dos nuevos marcadores al final, cuyo atributo tp (**tipo**) es "FOCO", e indica que estos datos vienen de la tabla de parámetros fisicosquímicos, y que representa los distintos focos de temperatura del modelo de temperatura. Se marca la posición del centro del foco y el radio.

Al ser analizado el XML por el Javascript, el resultado en el navegador incrustado es el mostrado en la Figura 203 siguiente, donde podemos observar los dos focos de temperatura señalizados.

Figura 203: Focos de Temperatura en mapa.

12.3.3.- Núcleo del Prototipo CIG implementado.

El Núcleo tiene un buffer de entrada que posibilita el desacople con el **Gestor del Front-End** con el que se encuentra estrechamente relacionado: cuando éste último necesita ejecutar procesos pesados o que requieren peticiones al **SE** encola un mensajes internos en el buffer de entrada del Núcleo del prototipo, como se muestra en la Figura 204, el cual consumirá los mensajes en su debido momento.

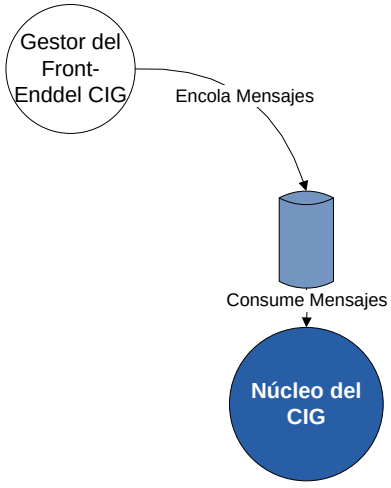


Figura 204: Relación entre Fron-End y Núcleo.

A continuación, se muestra el algoritmo básico de actuación en la Figura 205:

- En verde se muestran las peticiones hechas al **SE**.
- En azul las acciones que son ejecutadas contra el Gestor del Front-End del **CIG** para actualizar la información gráfica mostrada.

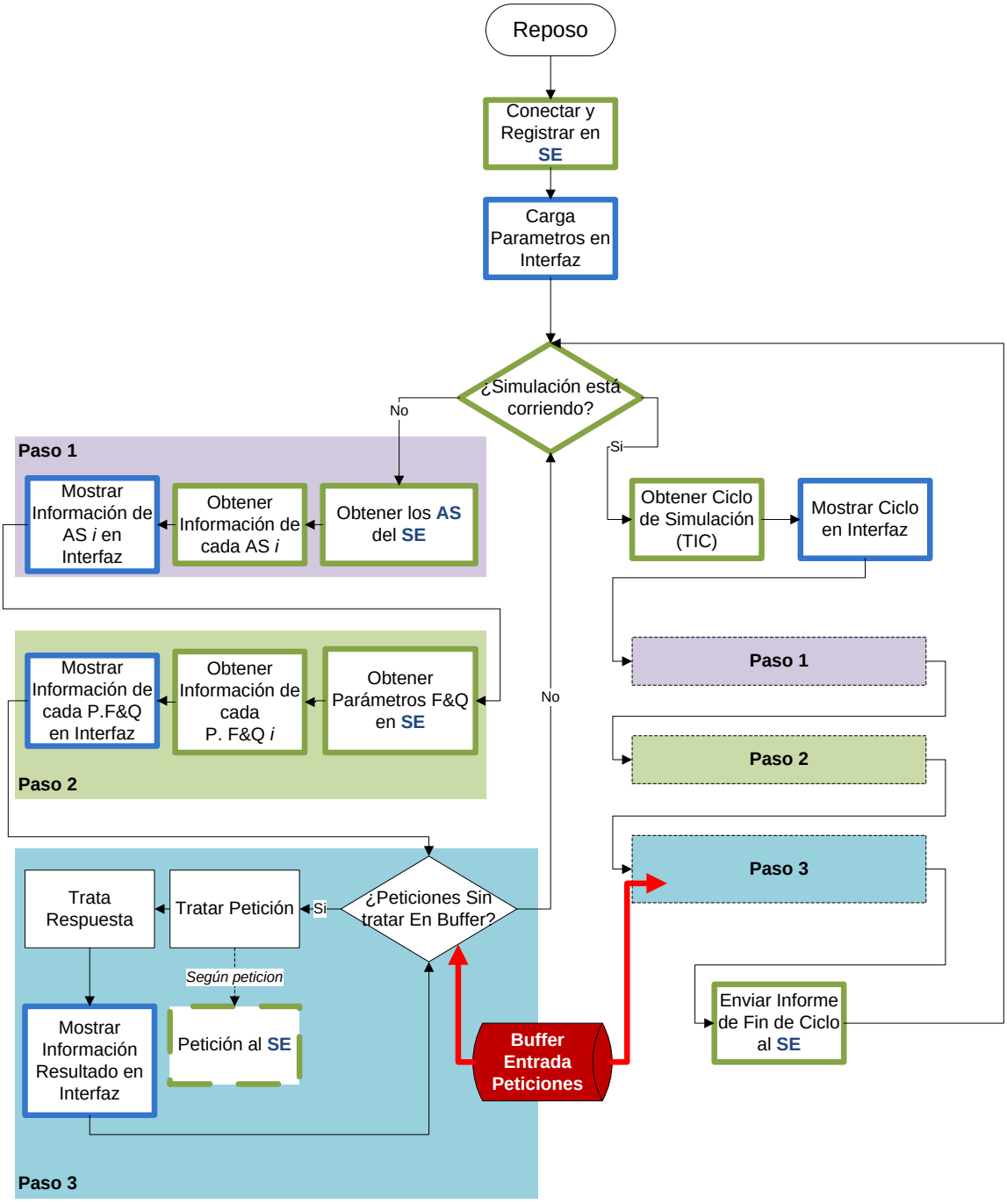


Figura 205: Diagrama de Flujo de actuación del Núcleo del CIG.

Primeramente registra al **CIG** como un cliente del **SE**. Establecida ya, envía comandos para obtener la configuración del servidor y mostrarlos gráficamente.

A partir de aquí, comienza el lazo de control principal, donde:

1) Mientras la simulación no esté corriendo en el servidor, se siguen los pasos básicos siguientes:

   - Obtener del **SE** todos los Agentes Simulados, tanto los creados por el propio **CIG** como los **AS** externos que se hayan conectado al **SE**, como un AUV hardware-in-the-loop. De los mismos obtener la información, actualizar los modelos de datos locales al **CIG** y actualizar la vista gráfica.

   - Obtener información de los distintos parámetros fisicoquímicos del Servidor del Entorno, para representarlos en la Interfaz. Este prototipo se ha especializado en la temperatura organizada como Focos de Temperatura, tal cual se describió en el **Apartado 10.2.7** la discretización de la Temperatura en el Servidor del Entorno.

   - Atender todas y cada unas de las peticiones encoladas en el Buffer de Entrada. Estas pueden ser de diversa índole:

     - Cambiar parámetros globales de configuración del Servidor del Entorno.
     - Arranque / parada de la simulación.
     - Creación / eliminación de **AS** locales.
     - Control de los **AS** (cambiar vector de velocidad que marca su navegación, plan de medidas, etc).
     - Registrar sensores de cada uno de los **AS** creados desde la interfaz.
     - Efectuar medidas desde cualquier sensor registrado para cada **AS**.
     - Otros procesos sobre la interfaz que puedan ser pesados.

2) En el momento que comience la simulación, y hasta que se detenga (en la mayoría de los casos se enviará un comando de finalización de la simulación desde el propio **CIG** prototipado):

   - Se recibirá el **TIC**, que marca el inicio de ciclo de simulación.

   - A partir de aquí, se siguen los mismos pasos básicos descritos en 1).

Finalmente, se envía un comando a **SE** de tipo “acknowledge” o confirmación, para que el servidor temporal dentro del servidor tenga en cuenta que el **CIG** cliente ya da por finalizado el ciclo de simulación, y por tanto hay conformidad con que avance el ciclo de simulación.

PARTE III

Resultados

CAPÍTULO 13.- *Realización del Proyecto*

En el presente Capítulo se describirá el grado de realización del proyecto, y se mostrarán los resultados obtenidos en las pruebas realizadas para validar la implementación actual del simulador.

13.1.- Grado de Realización del Proyecto

Ya fueron descritos en el **Capítulo 8** los distintos subsistemas que conforman la arquitectura ***SUBES***. También se muestran en la Figura 206.Los módulos que se han analizado, diseñado e implementado en el transcurso de este proyecto son los siguientes:

- El **Servidor del Entorno** (**SE**): ofrece el entorno virtual para la simulación de dispositivos de los Agentes Simulados.
- El **Agente Simulado** (**AS**): entidades con lazo de control para cumplir una misión usando los recursos ofrecidos por el **SE**.
- El **Cliente de Interfaz Gráfica** (**CIG**): cliente para monitorizar el estado de la simulación en todo momento.

Sin embargo, estos otros módulos han quedado sin implementar:

- El Servidor Temporal (**CLOCK**): pues en la implementación hecha está embebido en el propio **SE**.
- El Cliente Configurador del Servidor (**CCS**): el **CIG** implementado se ha hecho mixto, contemplando también comandos para configurar el servidor, por lo que no ha sido necesario hacer un cliente aparte del **CIG** para el propósito de configuración del servidor. En diseños más formales sí que habría que generar un **CCS** separado.
- El Cliente Configurador de Agente Simulado (**CCAS**): en la interfaz gráfica descrita en el **Capítulo 12**, ésta es capaz de comandar a los **AS** creados desde la propia interfaz de usuario, y, por tanto, la comunicación era interna. Queda pendiente desarrollar un **CCAS** capaz de comunicarse por XDR al **AS** y controlarlo mediante comandos.

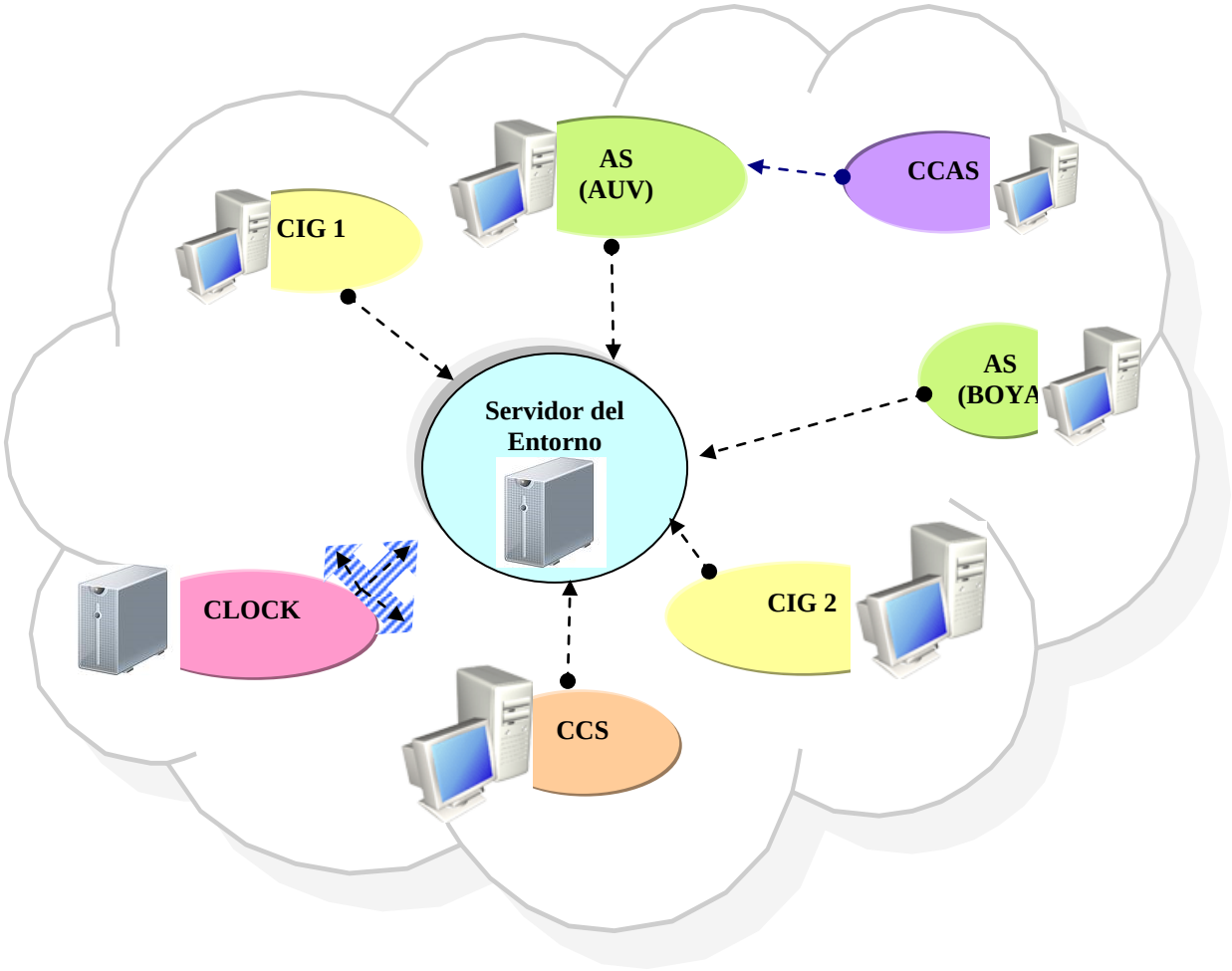


Figura 206: Subsistemas del simulador SUBES

13.1.1.- Comunicación Externa

La comunicación entre los Subsistemas de la arquitectura ***SUBES*** mediante paquetes de datos XDR fue explicada en el **Apartado 8.4,** donde se citaban 3 tipos de Mensajes:

- **Telecomandos**: para la ejecución de comandos y sincronización. Eran tres tipos según los *elementos de comunicación para integración entre subsistemas del simulador* del **Apartado 8.3.1**:
  - **TIC** : sincronización temporal.
  - **PET** : petición o solicitud de comando
  - **RES** : respuesta o "informe".

Todos fueron implementados en el presente proyecto como unidades básicas de comunicación de cualquier Cliente Externo con el Servidor del Entorno.

- **Transmisión de Datos Escalares** o paquetes de tipo **DATA,**  según la iconografía utilizada, para el envío de datos en *modo raw*, no ha sido implementado en el presente proyecto, pues tiene más utilidad en la comunicación entre el **AS** y el **CCAS**, no tratado en el proyecto.
- **Transmisión de Ficheros**: para el envío de ficheros en formato binario. No ha sido implementado en el presente proyecto.

13.1.2.- Servidor del Entorno

Este subsistema de explicó ampliamente en el **Capítulo 10**. En el mismo se propuso un catálogo de servicios, entendido cada uno como grupo de comandos que pueden ser solicitados por clientes. Se organiza el servidor en nueve componentes Gestores o módulos que llevan a cabo parte del catálogo de servicios, y que han sido implementados con las siguientes excepciones:

- **Gestor de Logs**: para registro de todas las acciones llevadas a cabo en el servidor. Su utilidad es puramente de auditoría o depuración, con lo que en esa fase se consideró prescindible.
- **Gestor de Dispositivos de Comunicación**: para la comunicación de los Agentes Simulados a través del propio servidor. Si bien es un módulo muy interesante, el presente proyecto se ha centrado en misiones individuales del AUV, por lo que se han delegado las misiones cooperativas (donde existe comunicación) para un segundo futuro paso más avanzado.

Por tanto, el grado de realización de este Subsistema es elevado, como también puede apreciarse en la Figura 207.

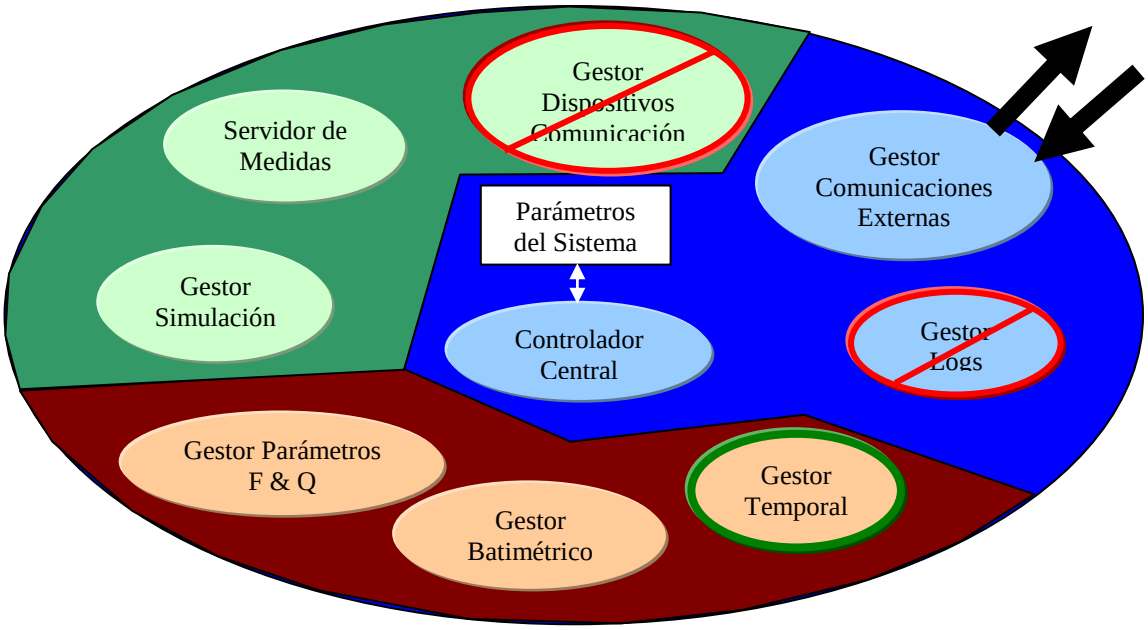


Figura 207: Componente Gestores del Servidor del Entorno implementados.

En cuanto a la gestión temporal de la simulación, en el **Apartado 9.1**, se proponían dos esquemas:

- Un Servidor Temporal externo: ideal para sincronizar varios Servidores del Entorno, que, como ya hemos comentado, no ha sido implementado.
- Un Servidor Temporal embebido en el Servidor del Entorno: ideal cuando hay un único Servidor del Entorno. Este fue el esquema implementado (marcado en verde en la figura).

13.1.3.- Interfaz Gráfica de Usuario

Se ha implementado el prototipo descrito en el **Apartado 12.3**. Este prototipo es plenamente funcional, si bien no alcanza todos los objetivos descritos en el diseño funcional de la interfaz en el **Apartado 12.1**. Se puede apreciar en la Figura 208.

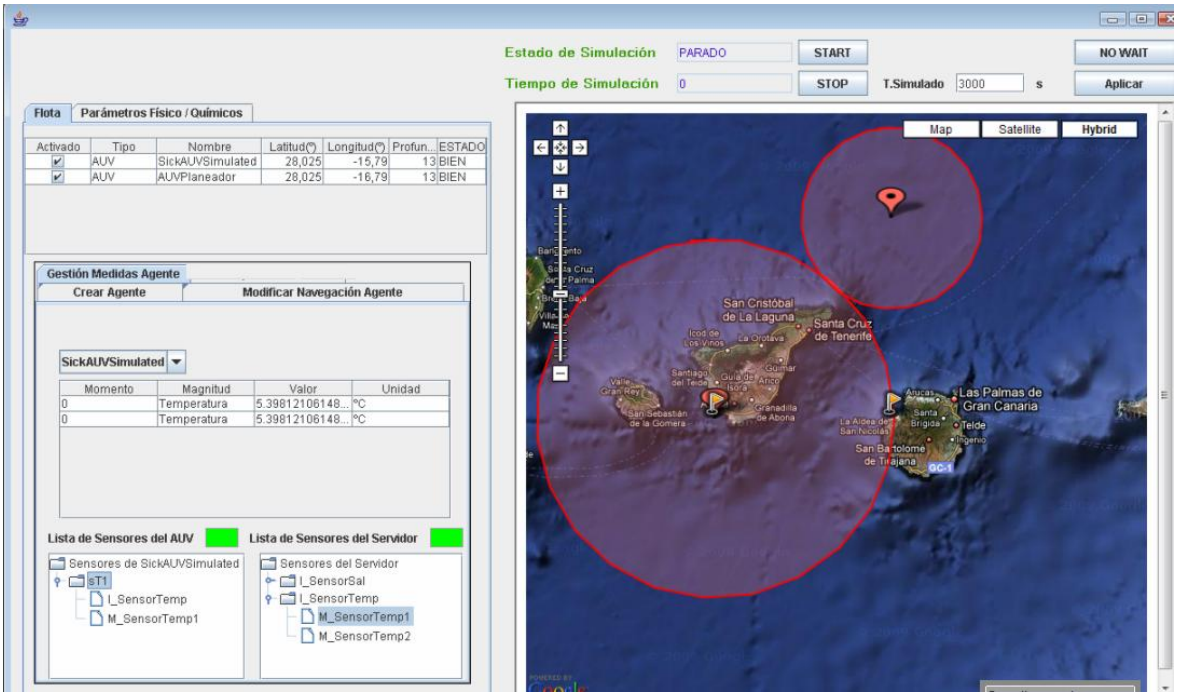


Figura 208: Interfaz Gráfica de Usuario implementada.

El objetivo fue diseñar una aplicación muy sencilla que permitiese monitorizar toda la actividad del Servidor del Entorno (que hace las veces de módulo **CIG,** el principal), a la vez que configurar y gestionar la simulación (que son comandos pertenecientes al **CCS** y que se han incorporado al **CIG** para simplificar el diseño del prototipo). Sin embargo, gracias a las características de escalabilidad del Servidor del Entorno, esta aplicación puede coexistir con cualquier otra aplicación de Interfaz Gráfica de Usuario más avanzada que sea diseñada con posterioridad al presente proyecto. Se han omitido en el prototipo módulos pesados de representación de mapas batimétricos o distribuciones de parámetros fisicoquímicos, usando para ello *GoogleMaps*.

Local a la interfaz, se pueden crear y comandar **AS** que se conecten al Servidor del Entorno para realizar sus planes de misión. Nótese que en esta implementación, la comunicación entre los AS y la Interfaz es mediante mensajería interna en Java, no mediante XDR, por lo que a este nivel la interfaz gráfica no puede ser entendida como un **CCAS**.

13.1.4.- Agentes Simulados

Se han implementado los **AS** que fueron descritos en el **Apartado 11.4**. Tal y como se explicó en el **Capítulo 11**, consiste en un mero prototipo para probar los servicios y

comandos que el Servidor del Entorno ofrece a los agentes móviles. Este prototipo no lee los planes de misión descritos en el capítulo, y organiza su operación en base a los siguientes dos planes elementales:

- *Navegación*: sigue un vector de velocidad, emergiendo y sumergiéndose continuamente durante su navegación, tal y como se muestra en la Figura 209.

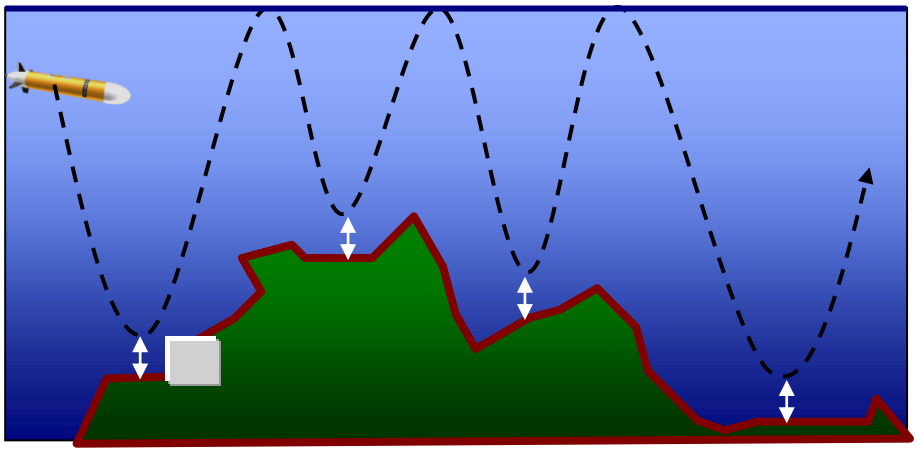


Figura 209: Plan de navegación de Zig Zag Vertical.

- *Medición*: realiza una medición de todos sus sensores periódicamente en cada ciclo de simulación.

Si bien puede ejecutarse individualmente, fue diseñado para ser creado y gestionado desde el prototipo de Interfaz Gráfica de Usuario implementado. De esta manera, el AS acepta comandos (a modo de ROV) para cambiar su plan de navegación, medición, etc. En el futuro se pueden desarrollar otros **AS** que mejoren los planes básicos aquí propuestos.

13.2.- Pruebas de Validación

Para todas estas pruebas se han implementado programas en Java, que se adjuntan en el CD anexo incluido en la documentación del proyecto. Se citarán los programas creados para cada prueba para que el usuario pueda reproducirlas.

13.2.1.- Pruebas de Conexión de Clientes Externos al Servidor del Entorno.

En esta primera prueba, el objetivo es comprobar que el protocolo de registro en el Servidor del Entorno funciona según lo previsto. Incluye los siguientes test:

- Que pueden conectarse clientes remotos, desde máquinas distintas.
- Que es posible realizar la comprobación de clave.
- Se verifica que el cliente no esté ya registrado en el servidor.

Diseño de "situación de conectividad"

1) Arrancar el Servidor del Entorno.
2) Conectar una Interfaz Gráfica de Usuario local a la máquina del servidor con clave incorrecta.
    a. Comprobar que no puede conectarse.
3) Conectar una Interfaz Gráfica de Usuario local a la máquina del servidor (**CIG1**) con clave correcta.
    a. Comprobar que puede conectarse.
4) Conectar Interfaz Gráfica de Usuario remota, con igual nombre que la anterior.
    a. Comprobar que no puede conectarse por coincidir el nombre del cliente.
5) Conectar Interfaz Gráfica de Usuario remota (**CIG2**) con nombre diferente.
    a. Comprobar que puede conectarse correctamente.
6) **CIG1:** Crear 3 AUVs.
7) **CIG2**: Crear 2 AUVs
8) Comprobar, tanto en CIG1 como CIG2, que los cinco AUVs son monitorizados.

El resultado es la Figura 210.

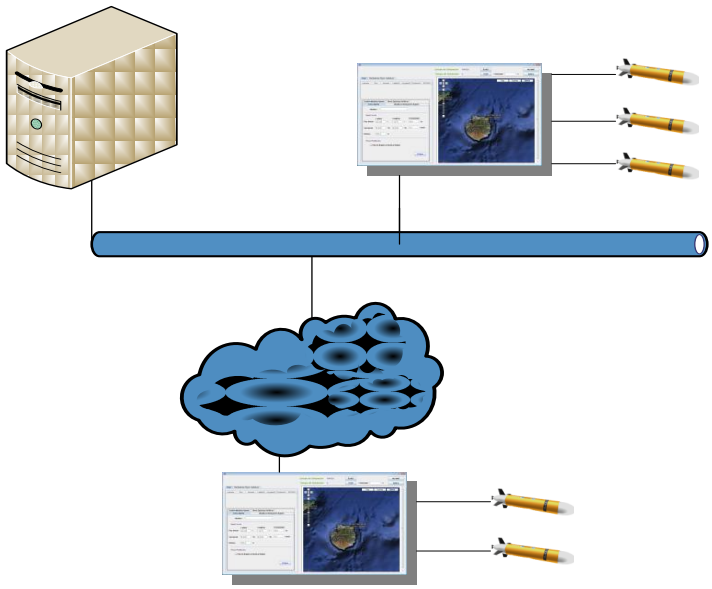


Figura 210: "Situación de Conectividad"

9) Para finalizar la prueba, ejecutar código para finalizar el Servidor del Entorno, y cerrar CIG1 y CIG2.

Resultado de la "situación de conectividad"

Tras arrancar el Servidor del Entorno, y ejecutar dos Interfaces Gráficas de Usuario (la local al servidor con 3 AUV Simulados, y la remota con 2), ambas monitorizan la misma situación del Servidor del Entorno:

- En ambas tablas aparecen los cinco AUV (los propios, y los creados por la otra interfaz gráfica).
- Sin embargo, en cada interfaz el usuario puede configurar la monitorización de los mismos datos del servidor, sin que uno influya en el otro:
    - El usuario del CIG1 puede querer monitorizar solo parte de los AUVs (tres por ejemplo). Se muestra en la Figura 211.

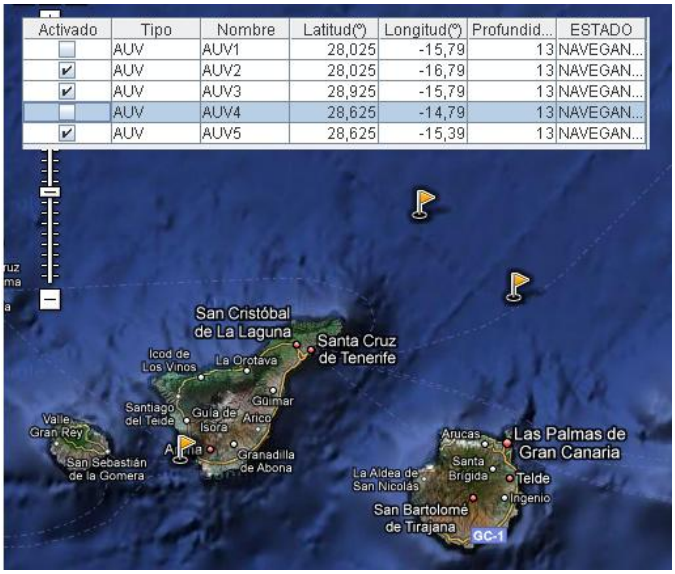


Figura 211: Resultado de "Situación de Conectividad" en CIG1.

- El usuario del CIG2 puede querer monitorizar todos los AUVs, y además los parámetros fisicoquímicos, en este caso los focos de Temperatura. Se muestra en la Figura 212.

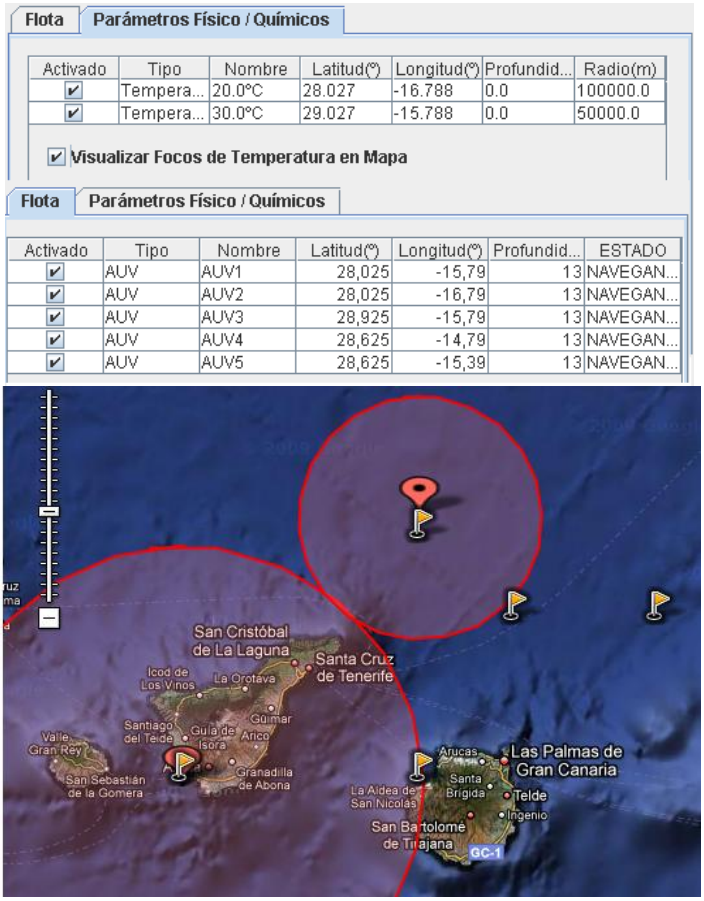


Figura 212: Resultado de "Situación de Conectividad" en CIG2.

- Aunque desde cualquier CIG se pueden ver todos los AUVs simulados en el sistema, cada uno sólo puede gestionar los AUVs que creó.
    - Esto puede verse en la pestaña **Flota**, tanto en el panel de **Gestión Medidas Agente** como en **Modificar Navegación Agente**, en ambos sólo es posible comandar a los agentes creados desde el propio CIG, como muestra la Figura 213.

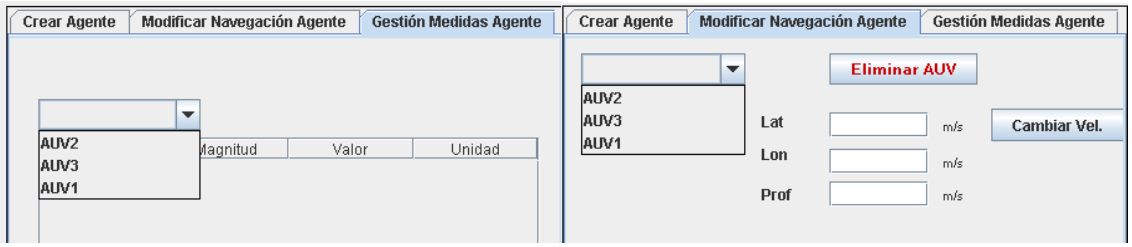


Figura 213: AUVs que pueden ser comandados desde el CIG.

La Figura 213 es un ejemplo del CIG1, donde sólo podemos seleccionar AUV1, AUV2 y AUV3 para poder comandarlos, por ejemplo, para cambiar el plan de medidas o el de navegación.

Si tratamos de ejecutar una tercera Interfaz Gráfica de Usuario, cambiando la clave de acceso al servidor (en el fichero **GUI.xml** de la interfaz, como se muestra en la Figura 214).

```
<?xml version="1.0" encoding="UTF-8"?>
<Parameters>
  <Server name="localhost" port="1983" password="cambiada"/>
  <Gui name="GUIsynch2.0" localhost="GUIsynch2.0@synch.iusiani.ulpgc.es" />
</Parameters>
```

Figura 214: Cambio de la clave de conexión del CIG.

Como resultado, el CIG no se puede registrar en el servidor, por lo que no puede monitorizar la simulación ni enviar ningún comando al mismo: no funciona el CIG. El resultado es similar cuando el CIG tiene el mismo nombre que alguno de los anteriores registrados (en el fichero XML): al no poderse registrar en el Servidor del Entorno no funciona el CIG.

En resumen, el resultado es satisfactorio, comprobándose que el Servidor del Entorno es escalable y admite múltiples clientes de diversos tipos al mismo tiempo corriendo sobre el mismo. Esto da gran versatilidad, pudiendo, por ejemplo, coexistir

varios usuarios con distintas Interfaces Gráficas de Usuario y monitorizando la misma simulación, pero de forma personalizada. Por otra parte, el protocolo de conexión al mismo funciona tal cual fue diseñado, teniendo que usar una clave y un nombre distinto para poder registrarse cada cliente.

13.2.3.- Pruebas de Sincronización.

El objetivo es comprobar que el Servidor del Entorno es capaz de llevar a cabo su protocolo de sincronización con todos los clientes conectados al mismo, a la vez que comprobar que el servidor lleva a cabo correctamente su política ante contingencias para evitar el bloqueo.

Diseño Paso 1

En este paso vamos a configurar un caso de sincronización normal.

1) Volver a la "*"situación de conectividad"*" explicada en el **Apartado 13.2.1**.
2) En cualquiera de las dos Interfaces Gráficas de Usuario, pulsar el botón **START** para comenzar la simulación.

Resultados Paso 1

El funcionamiento es normal, y todos los AUVs actualizan su posición en cada ciclo, mientras que también en cada ciclo la Interfaz Gráfica de Usuario monitoriza el estado de toda la flota. Veamos dos instantes en la Figura 215 y Figura 216:

Figura 215: Instante 300.

Figura 216: Instante 250800.

Como se observa, cada AUV ha seguido su plan de navegación:

- Los que tenían como plan de navegación el seguir su vector de velocidad, en su mayoría ya no están operativos (estado **NO RESPONDE**). Esto es así porque el usuario configuró un vector de velocidad con velocidad de descenso constante, con lo que finalmente, si no cambia dicho vector, terminará encontrándose con el fondo - según el mapa batimétrico del Servidor – y quedando no operativo.
- Otros siguen su curso de navegación, principalmente los que tenían un plan de navegación más elaborado, como era el "Plan de Zig Zag Vertical", de forma que se sumergen hasta una distancia de seguridad con el fondo, para volver a emerger a la superficie.

Como podemos comparar en la siguiente Figura 217, la posición del instante 25800 es muy diferente a la posición en el instante 300.

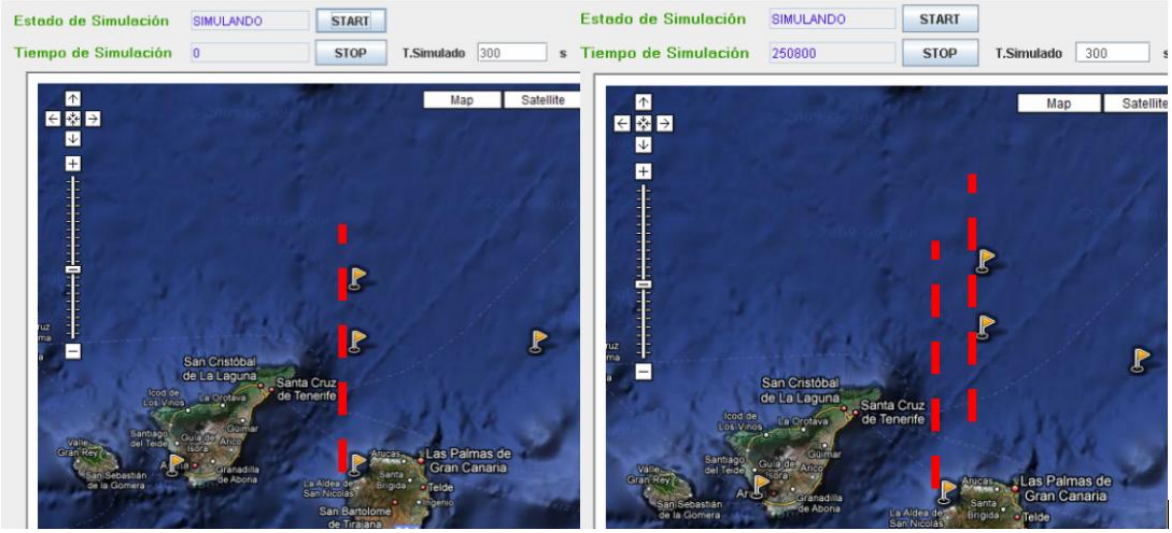


Figura 217: Instante 300 vs Instante 250800.

Se observa Figura 217 que mientras algunos AUVs avanzan, otros ya no responden.

Diseño de Paso 2

En este paso, se van a comprobar la resolución de contingencias, para lo que partimos de la situación que quedó en el Paso 1: la simulación corriendo con cinco AUVs en la misma.

1) Durante la simulación del **Paso 1**, ejecutar un nuevo cliente de tipo AUV externo (***MainCLIENTE_no_SINCRONIZADOs.java,*** que arranca agentes del código ***AGENTE_no_SINCRONIZADO.java***).
    a. Diseñado especialmente para al recibir el inicio de ciclo de simulación del Servidor del Entorno, no devuelva una respuesta al mismo como que ya terminó de realizar todas las acciones y está listo.

Resultado de Paso 2

En el segundo paso, y todavía durante la simulación, ejecutamos el cliente **MainCliente_no_SINCRONIZADOs**: el Servidor no sabe si ha recibido el tiempo de ciclo o no el cliente, pues no devuelve respuesta. Por ello no puede avanzar la simulación. El resultado es que la simulación se queda temporalmente bloqueada manteniendo el mismo tiempo de ciclo: no avanza y no se actualiza la posición de los AUVs simulados.

En cuando al cliente conectado, permanece bloqueado esperando recibir un nuevo ciclo de reloj, que no recibe porque el servidor está bloqueado esperando su respuesta.Tras

varios ciclos (exactamente 6 ciclos, que es el parámetro global del fichero de configuración del servidor del entorno **TopeCiclos**), finalmente el AUV es expulsado del servidor, o sea, se desregistra y no es tomado en cuenta en la sincronización, con lo que se puede avanzar el ciclo. A partir de aquí, la simulación seguirá su curso, y cada ciclo se actualiza la posición de los AUVs.

13.2.4.- Pruebas de Gestión de la Simulación

Estas pruebas están destinadas a probar las capacidades de la Interfaz Gráfica de Usuario implementada para gestionar la simulación: arranque, parada, elementos a visualizar, configuración de los parámetros globales de la simulación, etc.

Diseño de "situación de gestión de simulación"

1) Arrancar el Servidor del Entorno.
2) Arrancar la Interfaz Gráfica de Usuario.
3) Creación de 2 AUVs en la pestaña **Flota / Crear Agente**, en posiciones (latitud y longitud) y con vectores de velocidad distintos (así se apreciará la diferencia en el movimiento de ambos). Activar un plan de navegación para que oscile entre la superficie y el fondo cada uno de ellos.
4) En la pestaña **Flota / Otras Opciones Gráficas**, pulsar en centrar el mapa, para que se ajuste el zoom a los AUVs que el usuario ha elegido que desea que sean visualizados.
5) Cambiar el ciclo de simulación a 300 s (cada ciclo de simulación equivale al paso de 300 segundos).

Resultado de la "situación de gestión de simulación".

El resultado se muestra en la Figura 218 es el siguiente:

Figura 218: Situación de Gestión de simulación.

Diseño del Paso 2 y Resultados

En este paso, el objetivo es analizar y gestionar la monitorización de la simulación:

1) Pulsar START.
    a. El usuario comprobará que el **Estado de Simulación** pasa a SIMULANDO, y que el **Tiempo de Simulación** va en aumento de 300 en 300, a medida que en la tabla de la flota se observan cambios en longitud, latitud y profundidad.
2) En la tabla de la flota, desactivar uno de los dos AUVs
    a. En el *Google Map* se debe observar que desaparece el AUV desactivado.
    b. Podemos activarlo o desactivarlo: al margen de que se muestre o no en el *Google Map*, el AUV seguirá avanzando (como se observará en la tabla).

Vemos en la Figura 219 siguiente por ejemplo el resultado de desactivar el AUV1.

Figura 219: Resultado de desactivar el AUV1 de la visualización.

3) En la pestaña de Parámetros Físico / Químicos, activar "Visualizar Focos de Temperatura en el Mapa".

Se puede observar los atributos del modelo de temperatura empleado, tanto en la Figura 220 como representado en el *Google Map* en la Figura 221.

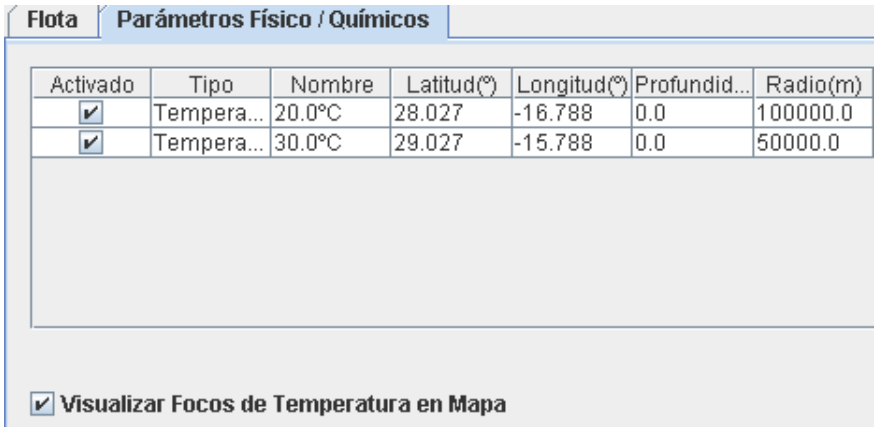


Figura 220: datos de Focos de Temperatura.

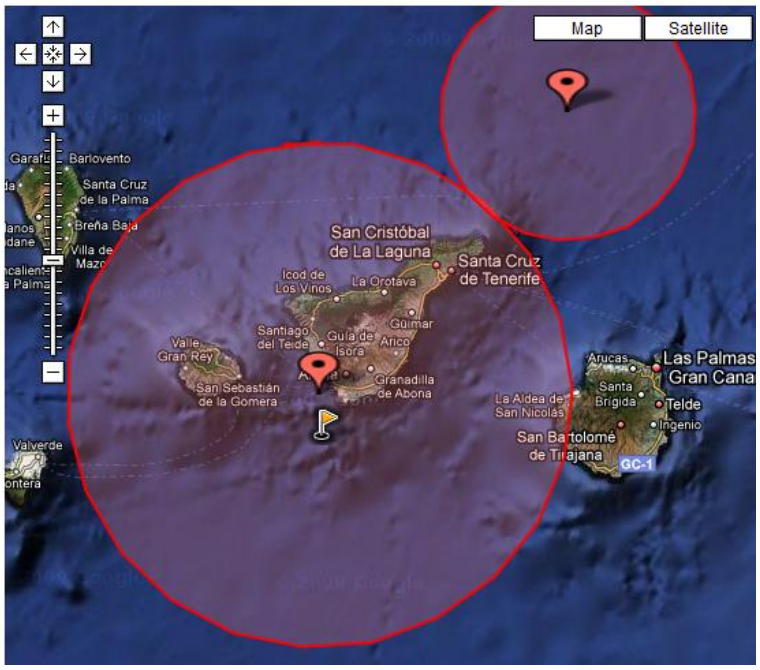


Figura 221: representación de focos en Google Map.

4) Crear un nuevo AUV en **Flota/ Crear Agente**.

   a. Se observa que se integra a la simulación, sin para ello hacer falta parar la simulación, como se muestra en la siguiente Figura 222:

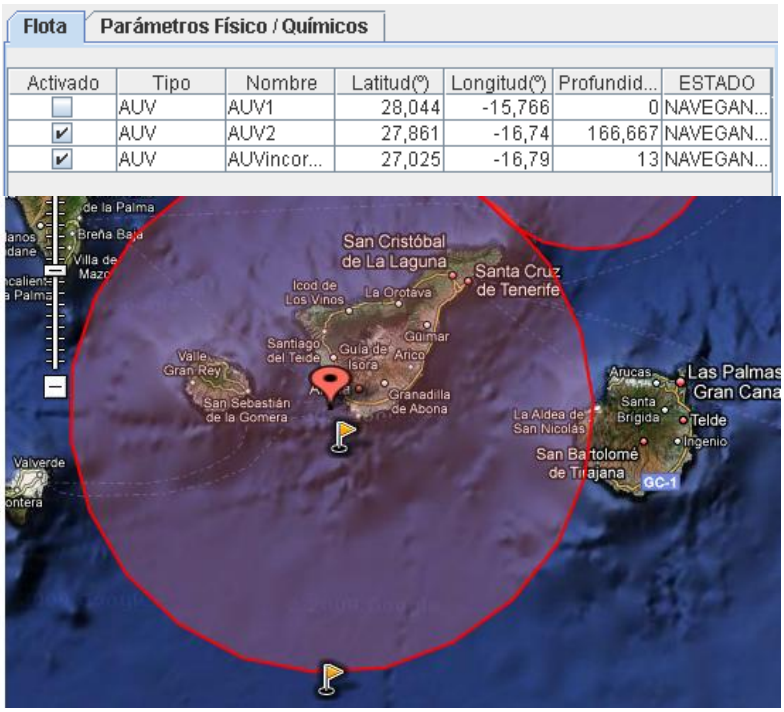


Figura 222: Incorporación de un nuevo AUV a la simulación.

b. De la misma manera, al eliminar cualquier agente en **Flota/ Modificar Navegación de Agente** se eliminará de la simulación sin que haya que detenerla ni afecte a los demás clientes del Servidor del Entorno.

Diseño y resultados del Paso 3

El objetivo es manipular la velocidad de simulación, para lo que se proponen dos técnicas:

1) Técnica 1: Aumentar el **Tiempo Simulado** a 3000s:

   Es la técnica más efectiva, pues dado que aumenta el tiempo simulado al que equivale cada ciclo, se discretiza más la simulación, si bien esta discretización puede implicar ciertas situaciones no deseadas:

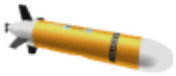
   - Si el paso de tiempo simulado es demasiado grande un AUV podría "atravesar una isla", ya que en la posición inicial está en el agua, y en la final también está en el agua.
   - Sucede algo similar con el plan de navegación emergiendo y sumergiéndose: se ha marcado una distancia de seguridad para sumergirse sin tocar fondo, pero si el paso de tiempo es demasiado grande se puede rebasar el umbral de seguridad para no tocar el fondo.

2) Técnica 2: Configurar el Servidor del Entorno para que no espere a la finalización del tiempo de ciclo (**NO WAIT**).

   Se obtiene una simulación más eficiente, puesto que cada ciclo dura justo el tiempo que tiene que durar: tan pronto responden todos los clientes, avanza el ciclo. Para pausar la simulación se puede configurar la simulación pulsando el botón **WAIT**, haciendo que cada ciclo de simulación tenga la misma duración real.

En definitiva, la Interfaz Gráfica de Usuario cumple plenamente su función de monitorización, permitiendo adaptar la monitorización a las necesidades de visualización del usuario en cada momento, y dando las herramientas suficientes para gestionar la simulación.

13.2.5.- Pruebas de Gestión de los AUVs simulados



El objetivo de estas pruebas es verificar que los Agentes Simulados implementados son capaces de usar los recursos del Servidor del Entorno para llevar a cabo su plan de Navegación y de Medidas.

Todos los Agentes Simulados que se incorporen serán gestionados a través de la Interfaz Gráfica de Usuario. Por tanto estas pruebas también validan las capacidades de la interfaz gráfica a la hora de comandar a estos Agentes Simulados.

Diseño de "situación de gestión de AUVs"

Primeramente se va a configurar la situación, creando varios AUVs:

1) Arrancar el Servidor del Entorno y la Interfaz Gráfica de Usuario.
2) Crear un AUV1 (**Flota / Crear Agentes**):
    a. Cerca de tierra (por ejemplo en Costas de Tenerife).
    b. Cerca del foco de Temperatura.
    c. El vector de velocidad, hacia tierra y alejándose del foco de Temperatura.
    d. La velocidad en profundidad 0.
3) Crear un segundo AUV2 (**Flota / Crear Agentes**) que tenga activado el plan de navegación para emerger y sumergirse al fondo constantemente.
4) Un tercer AUV3, sin el plan de navegación para emerger y sumergirse al fondo activado, y con velocidad en el eje de profundidad.

Resultado de "situación de gestión de AUVs".

El entorno de pruebas queda tal como se muestra en la Figura 223:

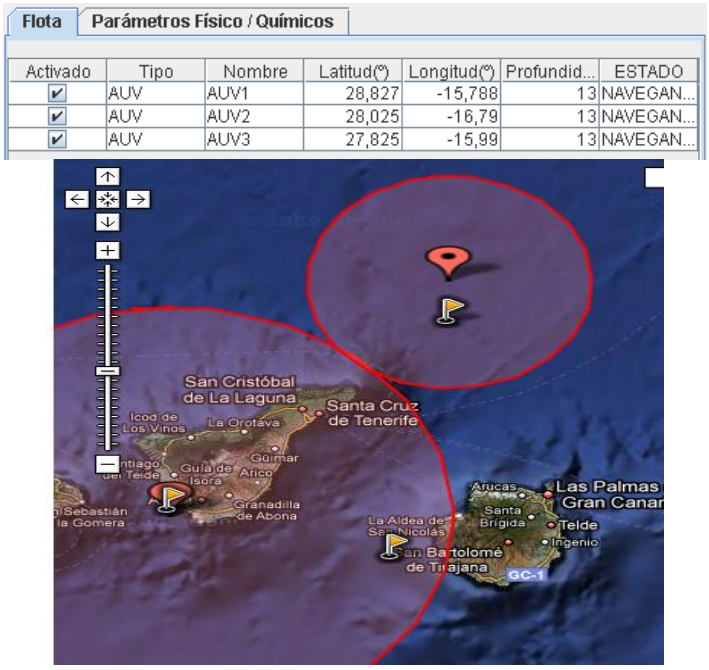


Figura 223: Situación de Gestión de AUVs.

Diseño de pruebas del Plan de Medidas y Resultados

Al AUV1, registrarle 2 sensores de temperatura, llamados **stemp1** y **stemp2**, con los dos modelos que ofrece el sistema: **M_SensorTemp1, M_SensorTemp2**, mostrado en la Figura 224.

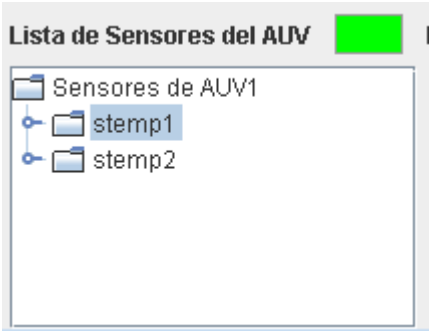


Figura 224: Sensores configurados en el AUV1.

Al comenzar la simulación, vemos cómo evoluciona la temperatura en función de la posición del AUV1 (véase la Tabla 46).

Momento	Magnitud	Valor	Unidad
0	Temperatura	21.288330673...	ºC
0	Temperatura	21.398330673...	ºC
9000	Temperatura	21.454638923...	ºC
12000	Temperatura	21.646067179...	ºC
21000	Temperatura	21.773046081...	ºC
24000	Temperatura	21.959487531...	ºC
60000	Temperatura	22.651583417...	ºC
105000	Temperatura	23.223836885...	ºC
150000	Temperatura	23.402706495...	ºC
186000	Temperatura	23.147864946...	ºC
286000	Temperatura	21.091267562...	ºC
377000	Temperatura	18.104668212...	ºC
442000	Temperatura	15.694290037...	ºC
507000	Temperatura	13.163600601...	ºC
598000	Temperatura	5.11	ºC
637000	Temperatura	5.11	ºC

Tabla 46: Tabla de medidas tomadas desde los sensores de AUV1.

En la Figura 225, vemos cómo ha evolucionado la navegación del AUV1:

Figura 225: evolución del AUV1 durante la navegación.

En el primer tramo (flecha discontinua roja) el AUV1 se acerca al foco de temperatura (el máximo central está a 30ºC). Así, hasta el instante 150000. A partir de este momento, comienza el segundo tramo (flecha discontinua naranja) el AUV1 se aleja del foco, hasta llegar al mínimo de 5ºC ya fuera del foco. Se observa que en el instante

637000, en realidad la temperatura es de 5,11ºC, debido al error que introduce el sensor de + 0.11.

Para terminar, véase que en las dos primeras medidas (las dos en el instante 0) se observan dos temperaturas diferentes: esto es porque fueron tomadas desde **stemp1**, y la segunda desde **stemp2**, por lo que cada una introduce su propio error en función del modelo.

Diseño de pruebas de Plan de Navegación y Resultados

Para estas pruebas nos centramos en el AUV2 y AUV3. Al comenzar la simulación, podemos cambiar la velocidad del AUV2. Veremos que con el plan de navegación configurado (*Zig Zag Vertical*) no choca con el fondo, ya que se sumerge hasta una distancia de seguridad, y acto seguido cambia el sentido de su velocidad en la componente de profundidad, para emerger. Por tanto, tiene una navegación sostenible, que se repite a lo largo de los ciclos, como se observa en la Figura 226 y Figura 227:

Figura 226: Instante 8200s, Emerge.

Figura 227: Instante 9100s, se sumerge hasta el límite sin tocar fondo.

En cuanto al AUV3, ya en el instante 4200 pasa al estado "**NO RESPONDE**", como se observa en la Figura 228: la razón es que tiene en su vector de velocidad una

componente positiva para sumergirse a velocidad constante, de forma que cuando llega al fondo (usando la batimetría del Servidor del Entorno), choca y se queda en estado "**NO RESPONDE**".

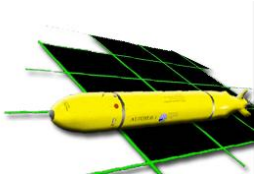
Este modelo de plan de navegación no es real, ya que no tiene sentido no controlar la cercanía con el fondo. Tiene sólo utilidad en el presente proyecto para mostrar al usuario que el AUV interacciona con el medio.

Figura 228: AUV3, NO RESPONDE.

Resultado General

En resumen, los AUVs simulados pueden seguir su Plan de Medición y Plan de Navegación usando los recursos del Servidor del Entorno, y ser monitorizados y comandados con versatilidad desde la Interfaz Gráfica de Usuario, permitiendo generar escenarios adaptados a las necesidades de la misión.

13.2.6.- Pruebas de Configuración del Servidor del Entorno.



En todas las pruebas anteriores se han contemplado todos los aspectos del Servidor del Entorno, excepto los cambios de configuración mediante el fichero xml. Con esta sencilla prueba se pretende ilustrar sobre el uso de este fichero.

Diseño

1) Acceder al fichero **configuración.xml** del Servidor del Entorno.

2) Ir al Apartado GParamFisicoQuimicos, y cambiar los dos focos de ejemplo mostrados (quitar uno, o modificar el centro, o el radio, etc).
3) Arrancar el Servidor del Entorno.
4) Arrancar la Interfaz Gráfica de Usuario.
5) En la pestaña **Parámetros Físico / Químicos**, activar **"Visualizar Focos de Temperatura"**.
    a. Se podrá observar que aparecen en *Google Maps* y en la tabla de datos los focos que se incluyeron en el fichero de configuración.

Resultado

Si bien el Servidor del Entorno ya ha sido probado en extensión en cada una de las pruebas anteriores, queda añadir un ejemplo de la fácil modificación del fichero de configuración del servidor (**configuracion.xml**). En este caso, en todas las pruebas hemos partido de la siguiente configuración mostrada en la Figura 229 de los focos de temperatura dentro del componente Gestor de Parámetros Fisicoquímicos:

```
<InterfazModelos interfaz="I_FocusData">
  <Modelo fichero="M_TempFocus">
    <Parametro id="1" valor="28.027" />
    <!--Latitud-->
    <Parametro id="2" valor="-16.788" />
    <!--Longitud-->
    <Parametro id="3" valor="0" />
    <!--Profundidad-->
    <Parametro id="4" valor="20" />
    <!--Temperatura en centro-->
    <Parametro id="5" valor="100000" />
    <!--Radio-->
    <Parametro id="6" valor="5" />
    <!--Temperatura final atenuada-->
  </Modelo>
  <Modelo fichero="M_TempFocus">
    <Parametro id="1" valor="29.027" />
    <Parametro id="2" valor="-15.788" />
    <Parametro id="3" valor="0" />
    <Parametro id="4" valor="30" />
    <Parametro id="5" valor="50000" />
    <Parametro id="6" valor="10" />
  </Modelo>
</InterfazModelos>
```

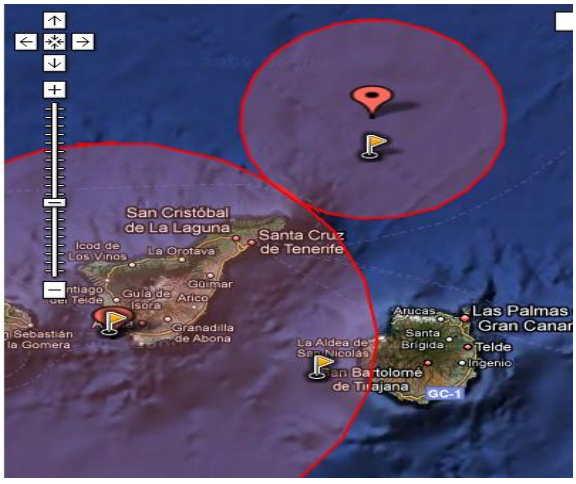


Figura 229: Configuración de dos focos en el fichero XML de configuración.

Si modificamos el fichero XML, y añadimos un tercer modelo como en la Figura 230, podemos obtener otra situación bien distinta:

```
<InterfazModelos interfaz="I_FocusData">
  <Modelo fichero="M_TempFocus">
    <Parametro id="1" valor="28.027" />
    <Parametro id="2" valor="-16.788" />
    <Parametro id="3" valor="0" />
    <Parametro id="4" valor="20" />
    <Parametro id="5" valor="40000" />
    <Parametro id="6" valor="5" />
  </Modelo>
  <Modelo fichero="M_TempFocus">
    <Parametro id="1" valor="29.027" />
    <Parametro id="2" valor="-15.788" />
    <Parametro id="3" valor="0" />
    <Parametro id="4" valor="30" />
    <Parametro id="5" valor="50000" />
    <Parametro id="6" valor="10" />
  </Modelo>
  <Modelo fichero="M_TempFocus">
    <Parametro id="1" valor="29.027" />
    <Parametro id="2" valor="-15.79" />
    <Parametro id="3" valor="0" />
    <Parametro id="4" valor="50" />
    <Parametro id="5" valor="30000" />
    <Parametro id="6" valor="10" />
  </Modelo>
</InterfazModelos>
```

Figura 230: Configuración de tres focos en el fichero XML de configuración.

Una simple modificación de los modelos del componente Gestor de Parámetros Fisicoquímicos nos da un nuevo entorno virtual: el primer foco ha sido diminuido en radio, y se ha añadido un nuevo foco en Gran Canaria.

En definitiva, es muy sencillo configurar el servidor para adaptarse a cualquier entorno que el usuario quiera recrear y simular.

CAPÍTULO 14.- *Conclusiones y Trabajo Futuro*

Este capítulo cierra el presente Proyecto Final de Carrera, y en él se presentarán las conclusiones finales, así como las tareas pendientes por hacer o propuestas para dar continuidad al presente proyecto.

14.1.- Conclusiones

Al comienzo del presente proyecto, se han sentado las bases a partir de las cuales se desarrolla el resto del trabajo realizado. Tras una breve introducción, donde se han definido los distintos conceptos relacionados con la robótica al servicio de la oceanografía, se ha completado la información con el estado del arte, para poder hacer contraste y concretar las aportaciones de este proyecto.

Análisis Previos

Se corresponde con la **PARTE I** del proyecto. Dado el carácter pionero y exploratorio del proyecto dentro del marco de trabajo de la ULPGC, se hizo necesario un análisis en profundidad en ciertos aspectos.

Un primer estudio comparativo de distintos entornos de programación en función de las necesidades previstas, sitúa Java como el lenguaje más adecuado para realizar toda la implementación del proyecto. XDR y XML se concluyen como adecuados para la serialización en la comunicación y configuración de las entidades del proyecto.

En segundo lugar, se han analizado distintas Arquitecturas de Simuladores de AUVs ya existentes, los distintos tipos de simuladores de múltiples agentes y protocolos de sincronización empleados. Se perfila en este capítulo la arquitectura CADCON / CODA **[ALB03]** como la más adecuada para el presente proyecto, por ser escalable y estar planteada como una arquitectura cliente / servidor.

A continuación, el proyecto aporta un análisis en profundidad de la batimetría y los distintos formatos y bancos de información existentes. Es sin duda netCDF el formato más adecuado para el presente proyecto, por ser el más ampliamente utilizado para representar mapas batimétricos.

Finalmente se hace un estudio de distintos modelos, tanto del entorno submarino (corrientes, temperaturas, etc.), como de la hidrodinámica, sensores y dispositivos de Comunicación de los AUVs, no sin previamente situar al lector en las variables oceanográficas que son captadas normalmente a bordo de los AUVs.

Arquitectura del Simulador *SUBES*.

Se corresponde con la **PARTE II**, donde se diseña una solución para los objetivos marcados en el proyecto. El diseño es escalonado, de forma que inicialmente se identifican los distintos elementos (llamados **Subsistemas**), y las bases de comunicación entre los mismos, en términos de paquetes de datos serializados en XDR.

En este punto, el proyecto aporta una librería Java llamada XDRVSUBES, que adapta los tipos de Java al estándar XDR. Al seguir el estándar, no sólo tiene utilidad en el presente proyecto, sino en cualquier proyecto de cualquier disciplina que necesite el uso de XDR en la serialización de datos.

Los siguientes capítulos describen los cuatro Subsistemas principales considerados en el proyecto:

- *Gestión Temporal y Sincronización del Simulador*: Se concluye el diseño del protocolo de sincronización entre Subsistemas.
- *Servidor del Entorno*: Es la contribución más importante del presente proyecto. Se diseña el motor de la simulación, que creará los entornos virtuales y atenderá a todo cliente que necesite monitorizar o participar en la simulación.

  El diseño, que se hace atendiendo a criterios de optimización de recursos, calidad, robustez, escalabilidad, configurabilidad, es plenamente funcional y cumple con los objetivos principales del proyecto.

  Finaliza el capítulo con un caso de estudio donde se discute cómo distribuir el Servidor del Entorno sobre un grid computacional.
- *Cliente Simulador de Agentes Simulados*: el comienzo es puramente analítico, haciendo un estudio de las misiones que puede realizar el mismo, y diseñando los planes de misión posibles. Acto seguido, se especifican los distintos subsistemas que integran el AUV.

  Se hace una importante aportación en el capítulo, como es la especificación de las misiones individuales en XML (trabajadas junto a colaboradores citados en el Apéndice).

  A continuación, el capítulo plantea dos soluciones:

  - La adaptación de un AUV real, el *SickAUV*, que se puede consultar en **[ENR08]** para poder simular ciertos sistemas en el Servidor del Entorno.
  - El diseño de un prototipo virtual (simulado), realizado para probar las capacidades del sistema.

- *Interfaz Gráfica de Usuario*: se realiza un estudio de los requisitos de software para que la interfaz pueda monitorizar y gestionar adecuadamente la simulación, para seguidamente diseñar un prototipo que, si bien no alcanza todos los requisitos software planteados, sí permite probar y monitorizar la actividad del Servidor del Entorno.

  En este capítulo se hace una importante aportación, como es el uso de Google Maps para representación del entorno simulado. Esto da valor añadido al proyecto, simplificando su implementación y utilizando una potente herramienta para monitorización de escenarios geo-espaciales.

Resultados

Se corresponde con la **PARTE III** de este Proyecto Fin de Carrera, donde se describe el grado de realización alcanzado, y se diseña y ejecuta una serie de pruebas de validación.

Se obtiene como resultado un Servidor del Entorno con un grado de realización bastante alto, entendido este grado como la obtención de un Subsistema plenamente funcional; si bien los modelos sensoriales, hidrodinámicos, distribuciones de parámetros fisicoquímicos y procesos se han simplificado.

Las características que más destacan del Servidor del Entorno implementado son las siguientes:

- **Modularidad**: el servidor es completamente modular, lo cual facilita su fácil mantenimiento.
- La característica anterior aporta **Comprensividad** por parte de cualquier desarrollador familiarizado con el entorno de programación en el que se ha programado (Java). Se complementa el proyecto con un **Manual de Implementación del Servidor** para la comprensión y fácil desarrollo de cualquier programador avezado.
- **Eficiencia y optimización de los tiempos de respuesta**: como veremos, en su diseño se contemplan módulos funcionales o componente**s Gestores**

especializados en gestionar ciertos servicios del Servidor, y a su vez cada centro de proceso es **multihilo**, pudiendo especializar cada hilo en un servicio concreto del componente, permitiendo así evitar cuellos de botella y agilizar la entrega de las respuestas a los clientes del servidor.

Se utilizan criterios para equilibrar la carga de trabajo del Servidor soportado en varios componentes Gestores, y también se **balancea la carga de trabajo** de cada componente Gestor en sus múltiples hilos.

- **Versatilidad**: esta de una de las características fundamentales y que aporta un gran valor añadido al presente proyecto. Mediante un fichero de configuración, el usuario puede parametrizar el servidor cada vez que lo arranca, y decidir:
    - Cuántos hilos son usados en cada servicio ofertado por el Servidor, pudiendo asignar más hilos a los servicios más pesados.
    - Qué componentes Gestores y, por tanto, qué servicios, se quieren habilitar o deshabilitar.
    - Qué clases de Java se van a usar para ofrecer cada servicio, y qué modelos de datos se van a usar en cada servicio para soportar el mismo.
    - Si es necesario **cargar nuevas clases** desarrolladas por el propio usuario. Todo esto se consigue mediante el uso de la **reflexión** y el trabajo del servidor con **interfaces** que prototipan cualquier clase que se quiera desarrollar. Esto posibilita la carga, en tiempo de ejecución, de clases previamente compiladas.

  En definitiva, el Servidor tiene un alto grado de configuración que permite adaptarlo al entorno concreto que se quiera simular, con el grado de realismo y complejidad que el mismo usuario elija. No se propone un servidor para cualquier situación, sino un servidor flexible y adaptable.
- **Escalabilidad**: De la característica anterior se deriva la escalabilidad, o sea, posibilidad de crear desarrollos propios e incorporarlos de forma sencilla al servidor.
- **Robustez:** El servidor habilita mecanismos para sincronizarse y comunicarse con clientes, sin que la actividad de los mismos en el servidor bloquee la marcha de la simulación. Así, se han diseñado protocolos de contingencia ante pérdidas de conectividad con clientes, o clientes que tarden demasiado y que retrasen el buen desarrollo del Servidor.

En cuanto a las comunicaciones, se ha diseñado un protocolo de comunicación **completo y robusto**, que permite responder ante cualquier situación que se plantee en la simulación de clientes en el Servidor del Entorno.

- **Multiplataforma**: Esta característica deriva del uso de Java como entorno de programación. Por tanto, es independiente del sistema operativo y del entorno: el Servidor del Entorno puede trabajar en múltiples plataformas, incluso en plataforma Web.

  En cuanto a la comunicación, XDR permite que la comunicación siga un estándar al que se puedan adherir los clientes, sea cual sea la arquitectura y lenguaje de programación del cliente. De hecho, para el presente proyecto se han hecho pruebas con clientes programados en C++.
- **Calidad**: la solución propuesta utiliza patrones de diseño que aportan calidad y eficiencia a la solución.

En cuanto a la arquitectura en general, se aprecia una alta **Escalabilidad**, pudiendo incorporar a la simulación clientes de muy distinto origen y con distinta finalidad: Interfaces Gráficas de Usuario, Agentes Simulados, AUVs reales con simulación de ciertos componentes, etc.

14.2.- Trabajo Futuro

14.2.1.- Mejoras Aplicables

Existen muchas mejoras aplicables al presente proyecto, si bien las más relevantes y que dan mayor valor añadido al proyecto pueden ser las siguientes:

Serialización XDR

- Implementar la serialización XDR de *Mensajes para Transmisión de Datos Escalares* en el **Apartado 8.4.4** y *Mensajes para Transmisión de Ficheros* en el **Apartado 8.4.5,** en la librería XDR ya implementada parcialmente. Esto permitiría desarrollar ciertos comandos de servicios del Servidor del Entorno, como es la obtención de un rango de batimetría de un fichero netCDF del servicio Batimétrico.

Servidor del Entorno:

- El modelo de simulación hidrodinámica de los Agentes Simulados en el componente Gestor de Simulación es demasiado sencillo, y el desarrollo de modelos hidrodinámicos más realistas permitiría realizar una simulación mucho más elaborada de la dinámica de los vehículos. Habría dos maneras de mejorarlo:
    - Alimentar al servidor del Entorno con datos de corrientes marinas obtenidos a partir de modelos ROM (Regional Ocean Model) como el proyecto de Puertos del Estado ESEOO **[ESE07]** (*Establecimiento de un Sistema Español de Oceanografía Operacional*). Estos modelos puede proporcionar campos de corrientes en 3D y en intervalos temporales de una hora.
    - Implementar un modelo de corrientes marinas en el componente Gestor de Parámetros Fisicoquímicos que afecte a la velocidad de navegación del Agente Simulado. Esta posibilidad puede ser especialmente interesante en el caso de gliders simulados.
    - En consecuencia, actualizar el componente Gestor de Simulación para que la actualización de la posición dependa de las corrientes marinas.
- El Modelo batimétrico es sencillo, tomando datos de un solo fichero batimétrico. No sería complicado y enriquecería mucho la simulación que el componente Gestor Batimétrico fuera capaz de manejar ficheros batimétricos de distinta resolución e integrar ambos para dar solución a una petición hecha.
- El consumo de energía durante la simulación, lo que equivale a simular la autonomía de los vehículos, no se ha implementado. Incorporarlo implicaría:
    - Que el componente Gestor de la Simulación gestione los cambios de la misma.
    - Que el componente Gestor de Simulación modifique el nivel de carga de la batería automáticamente con cada cambio de posición del AUV.
    - Que el componente Servidor de Medidas modelo el consumo energético cuando se realizan mediciones con los instrumentos a bordo del vehículo.
    - Que en cada ciclo, en Gestor de Simulación compruebe la batería, de forma que ante un nivel demasiado bajo de la batería el AUV pase a diferentes estados (a la deriva, etc.).

- Incorporar otros modelos de Parámetros Fisicoquímicos distintos de la Temperatura, que enriquezcan la simulación.
- En el diseño del Servidor de Medidas se habilitó la posibilidad de registrar sensores, e incluso que éstos tuvieran un vector de configuración, que afecta a cada medida tomada (resolución, etc.). Implementarlo daría mayor entidad a los modelos de sensores registrados.
- Implementación de un sistema de registro de eventos o logger para depuración de la actividad del Servidor del Entorno (**SE**). Sin él, ahora mismo la actividad del **SE** sólo se puede observar a través de las Interfaces Gráficas de Usuarios. Un logger bien estructurado permitiría analizar los procesos en el mismo con mucho más detalle.
- Implementación del componente Servidor de Dispositivos de Comunicación, para completar la implementación del Servidor del Entorno.

Agente Simulado

El prototipo de Agente Simulado implementado se puede mejorar en múltiples aspectos, entre los más importantes cabe destacar los siguientes:

- Actualmente la comunicación es mediante mensajes internos a la Interfaz Gráfica de Usuario. Esto significa que no podría funcionar desde el exterior a la misma. Hay que mejorar este aspecto, e implementar un módulo que permita que esa comunicación sea mediante XDR, como ocurre entre todos los Subsistemas del Simulador. Esto proporcionaría mayor versatilidad y escalabilidad, haciendo al Agente Simulado independiente de la Interfaz, aunque puede seguir gestionándose desde la misma mediante comunicación externa en XDR.
- Los planes de misión se especifican mediante código. Habría que mejorar este aspecto, y que el Agente Simulado sea capaz de deducir sus planes desde la Especificación de la Misión en ficheros XML como los descritos en el **Apartado 12.2**. Esto permitiría lograr un hito importante en el desarrollo del modelo formal del simulador.

Interfaz Gráfica de Usuario

- Tiene dos grandes deficiencias en el prototipo implementado:

  - Al comenzar, automáticamente lee el fichero XML y se conecta al Servidor. Si no puede conectarse al mismo, queda fuera del simulador. Lo ideal sería que con anterioridad a usar la información del XML para conectarse al servidor, el usuario tuviera acceso a una pantalla previa donde cambiar esta configuración, y pudiera seleccionar a qué servidor conectarse, a qué puerto, con qué clave, etc.
  - La configuración del Servidor del Entorno debe realizarse directamente sobre el fichero XML de configuración del mismo. Sería de gran utilidad que la Interfaz Gráfica de Usuario incorporase un cuadro de mandos para cambiar esta configuración de forma sencilla.
- El resto de deficiencias podrían solucionarse implementando una nueva Interfaz Gráfica de Usuario siguiendo las directrices de los **Apartados 12.1 y 12.2**.

14.2.2.- Propuesta de Proyectos Relacionados

Se enumeran a continuación los proyectos que considero deben seguir al presente, para completar y aprovechar todo el trabajo realizado.

Relacionados con los AUV

Proyecto1.- ***"****Implementación de Vehículo Submarino Autónomo completamente simulado en el simulador* ***SUBES"***

El proyecto pretende mejorar el prototipo rudimentario propuesto en el presente proyecto. Así, el objetivo será disponer de un AUV completamente funcional, capaz de interpretar los planes de misiones descritos en el presente proyecto, y comunicarse con el Servidor del Entorno para simular cualquier componente. Para este proyecto, son necesarios:

  - Estudio de la Especificación de la Misión que adjunta en este proyecto y que fue diseñado dentro del marco de proyectos de la ULPGC sobre vehículos submarinos autónomos.
  - Estudio de arquitectura de AUVs reales.
  - Análisis, diseño e implementación del mismo, con la capacidad de interpretar todos los aspectos de los planes de misión.

- Análisis diseño e implementación de un Cliente Configurador del Agente Simulado, para comandar el AUV: con capacidad de cambiar planes de misiones incluso cargar un plan o misión completos.
- Adaptación de la Interfaz Gráfica de Usuario propuesta en el presente proyecto para incorporar el cliente anterior, y ofrecer al usuario un cuadro de mandos para comandar y cargar o modificar misiones en el AUV.

Proyecto 2.- "*Simulación de la hidrodinámica de Vehículos Oceanográficos a través del simulador* ***SUBES***"

Se pretende dotar al simulador de un modelo fidedigno de la hidrodinámica de los vehículos que participen en la simulación. Sin duda, este proyecto es de gran valor científico, pues aportaría realismo a las pruebas hechas por el desarrollador.

Nótese que se habla de "vehículos oceanográficos", que engloba al conjunto de sistemas robóticos al servicio de la oceanografía. Tal como se citó en la introducción, son principalmente tres: AUV (*Autonomous Underwater Vehicle)*, ROV (*Remote Operated Vehicles)* y boyas oceanográficas. El proyecto deberá resolver:

- Estudio, diseño e incorporación al Servidor del Entorno de un modelo oceanográfico complejo, que integre **corrientes marinas, temperatura y salinidad** entre otros. Deberá incorporarse al componente Gestor de Parámetros Fisicoquímicos.
- Estudio del modelado de la hidrodinámica de un vehículo oceanográfico real, en base a la interacción del vehículos con el modelo oceanográfico complejo antes descrito. Mientras que en el presente proyecto el movimiento de los vehículos se hace en base a un vector de velocidad del vehículo en su conjunto, en este proyecto propuesto haría falta un nivel más profundo de simulación, para poder identificar las órdenes a ejecutar en los actuadores para que el vehículo adopte cierta pose y se mueva a un punto concreto.
- Por su parte, en el Servidor del Entorno, en el Gestor de la Simulación, habrá que incorporar un servicio para el registro y manejo de actuadores según modelos reales, y que será usado por el modelo de hidrodinámica implementado en el mismo componente.

- Incluir servicios en el componente Gestor de la Simulación para la predicción de posición del vehículo oceanográfico.
- Mejora del prototipo de Interfaz Gráfica de Usuario, añadiendo:
  - Representación gráfica y tridimensional a mucho mayor nivel de detalle de la posición, orientación, etc. de los distintos tipos de vehículos oceanográficos.
  - Capacidad de incorporar u ocultar en el mapa predicciones de movimiento, información que obtendrá del Servidor del Entorno.

Proyecto 3: "*Integración de SickAUV para la simulación de componentes en el simulador **SUBES***"

Como ya se ha mencionado, este proyecto surge en un marco de proyectos de la ULPGC relacionados con los vehículos submarinos autónomos. Uno de estos proyectos fue el citado en la referencia **[ENR08]**, donde se diseño e implementó SickAUV en CoolBot. Este proyecto trata de simular ciertos dispositivos (sensores, hidrodinámica, etc.) conectando a SickAUV como **Agente Simulado** al Servidor del Entorno. El proyecto requerirá:

- Análisis de modelos de sensores reales y de interés para SickAUV, así como diseño e incorporación al Servidor del Entorno (componente Servidor de Medidas).
- Análisis de modelos de hidrodinámica real y de interés para SickAUV, así como diseño e incorporación al Servidor del Entorno (componente Gestor de la Simulación).
- Análisis de modelos de consumo de baterías real y de interés para SickAUV, así como diseño e incorporación al Servidor del Entorno (componente Gestor de la Simulación).
- Análisis de SickAUV
  - Adaptar los componentes CoolBot, para comunicación con el Servidor del Entorno de ***SUBES***.
  - Estudio e implementación de adaptaciones de SickAUV para la sincronización con el Servidor del Entorno.

Proyecto 4: "*Adaptación del simulador **SUBES** para misiones colaborativas entre múltiples Vehículos Submarinos Autónomos.*"

Se trata – sin duda - de un proyecto interesante, que aporta un segundo nivel de complejidad de mucho interés para oceanógrafos e ingenieros. Se persigue diseñar y simular misiones en las que los AUVs se comunican entre sí, y adoptan formaciones y estrategias para cumplir un objetivo de forma autónoma en función de las condiciones del medio. Algunos de los hitos que habrá que acometer en el proyecto son:

- Estudio de otros simuladores, como el CODA/CADCON **[ALB03]**, que permiten misiones colaborativas. Análisis de los algoritmos utilizados.
- Análisis y diseño de distintos protocolos organizativos y algoritmos enfocados a resolver ciertas misiones complejas.
- Análisis, diseño y adaptación de la especificación de los planes de misión propuestos en el presente proyecto para incluir misiones incluyendo los protocolos citados en el punto anterior.
- Adaptación del Servidor del Entorno, para implementar el componente Servidor de Dispositivos de Comunicación.
- Diseño e Implementación de Agentes Simulados capaces de seguir los nuevos planes de misiones cooperativos.

Relacionados con el Servidor del Entorno

Proyecto 5: "*Optimización del Servidor del Entorno del Simulador **SUBES** mediante un entorno de cómputo distribuido (grid)*"

Es importante optimizar el Servidor del Entorno, pues es el motor de la simulación, sin el cual no pueden escalarse el sistema con modelos más complejos. Por tanto, se propone analizar, diseñar e implementar el Servidor del Entorno con cada uno de sus componentes en máquinas remotas entre sí, tal y como se propuso en el caso de estudio del presente proyecto en el **Apartado 10.6**.

Relacionados con la Interfaz Gráfica de Usuario

Proyecto 6: *"Implementación de la Interfaz Gráfica de Usuario para el simulador* ***SUBES****"*

Como se ha expuesto en el presente proyecto, se ha implementado un prototipo perfectamente útil para pruebas, si bien no es nada cómodo para el usuario final (científicos y oceanógrafos). Por tanto, se proponer replantearlo, acorde a la propuesta hecha en el **Apartado 12.1** y **12.2**.

BIBLIOGRAFÍA

[AKANTP] "*Introduction to NTP*".
http://www.akadia.com/services/ntp_synchronize.html

[AGU2005] *"Aguja Giroscópica: Rigidez y Precesión giroscópicas".* Abril 2005
http://www.estudiosnauticosta.com/articulo.php?num=84

[AKE1999] Per Akesson, *"Side Scan Sonar"* 1999
http://www.abc.se/%7Epa/mar/sidescan.htm

[ALB03] Erik Albert, Jonathan Bilodeau, and Roy M. Turner. "*Interfacing the Coda and Cadcon Simulators: A Multi-Fidelity Simulation Testbed For Autonomous Oceanographic Sampling Networks*". Proceedings of the 13th International Symposium on Unmanned Untethered Submersible Technology (UUST), Durham, NH, August, 2003.

[ALEX] Alex Khaneles. *"Integer Lattice Gases at Equilibrium"*.

[ARL] "*Underwater Acoustic Communications*". Nacional University of Singapore.
http://www.arl.nus.edu.sg/web/research/acomms

[BODC] "Gridded Bathymetry data (GEBCO)"
http://www.bodc.ac.uk/data/online_delivery/gebco/

[BRAU04] Braubach, L., Pokahr, A., Lamersdorf, W., Krempels, K.H., Woelk, P.O. "*A Generic Simulation Service for Distributed Multi-Agent Systems*. Universidad de Hamburgo. (2004)

[BU2005] Bureau Hidrográfico Internacional. "*Manual de Hidrografía*", 1ª Edición. Mayo de 2005

[CAB05] J.M. Cabanellas. "*Tecnología de Simuladores"*. Grupo de Ingeniería Gráfica. Escuela Superior de Ingenieros Industriales. Universidad Politécnica de Madrid. (2005)

[CAD] *Cadcon: cooperative AUV Development Concept*
http://www.ausi.org/cadcon/simulation.html

[CAV2004] E. Cavallo , R.C. Michelini , *"A robotic equipment for the guidance of a vectored thrustor AUV"*, 35th International Symposium on Robotics ISR 2004, Paris, 23-26 March 2004.
http://www.dimec.unige.it/PMAR/

[CCMA] National Oceanic and Atmospheric Administration (NOAA). *"The Center for Coastal Monitoring and Assessment"*
http://ccma.nos.noaa.gov/

[CH1996] S.K. Choi & J. Yuh, "*Experimental Stud of a Learning Control System with Bound Estimation for Underwater Robots, J. of Autonomous Robots*". March 1996.

[CH1997] S. K. Choi and O. T. Easterday . *"An UnderwaterVehicle Monitoring System and Its Sensors"*. Departament of Mechanical Engineering. University of Hawaii. 1997.

[COI2005]E. Coiras, Y. Petillot. , Lane, D. M. "*An Expectation-Maximization Framework for the Estimation of Bathymetry from Side-scan Sonar Images"*. Junio 2005

[CRIS89] Cristian, F. "*Probabilistic clock synchronization*", Distributed Computing (Springer) **3** (3): 146-158. (1989)

[DAV2003] Dr. David P. Stern. Traducción J. Méndez y F. Pz. Guinea. Septiembre 2003
http://www.iki.rssi.ru/mirrors/stern/stargaze/Mlatlong.htm

[DOSITS] University of Rhode Island. *"Acoustic Modem"*.
http://www.dosits.org/gallery/tech/c/am1.htm

[ENR08] Enrique Fernández Perdomo. Proyecto Final de Carrera "Sistema Integrado de Control para un Vehículo Submarino Autónomo".

[ESE07]: "*Proyecto ESEOO: Establecimiento de un Sistema Español de Oceanografía Operacional"*.
www.eseoo.org

[F2002]Fossen, T. I, "Marine Control Systems: Guidance, Navitaion and Control of Fhips, Rigs and Underwater Vehicles", Marine Cybernetics AS, Trondheim, December 2002.

[FAK2004] A. El-Fakdi, M. Carreras, N. Palomeras y P. Ridao. "Autonomous Underwater Vehicle Control using Reinforcement Learning Policy Search Methods". Institute of Informatics and Applications University of Girona. Edifici Politecnica 4, Campus Montilivi (EPS). Girona, 17071, España. 2004.

[FUJI99] R. M. Fujimoto. "*Parallel and distributed simulation*". En la conferencia " Proceedings of the Winter Simulation"paginas 122–131, (1999).

[GAR1970]. M. Gardner. *"Scientific American, Mathematical Games column"*. Octubre 1970

[GEODES] Geospatial Designs *"Surfer 7 Grid File Format"*
http://www.geospatialdesigns.com/surfer7_format.htm

[GFDL] National Oceanic and Atmospheric Administration (NOAA). "*Netcdf Common Data Format*"
http://www.gfdl.noaa.gov/products/vis/data/netcdf/index.html

[GIO2006] José Luis Giordano. *"La Brújula"*. 18 de Febrero, 2006
http://www.profisica.cl/comofuncionan/como.php?id=22

[GMT] "*Output Data Formats*"
http://gmt.soest.hawaii.edu/gmt/doc/html/GMT_Docs/node57.html

[GUS89] Gusella and Zatti. "*The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD*", Software Engineering, IEEE Transactions on (IEEE) 15 (7): 847-853 (1989)

[HARD96] Harding A. (ed): "Microsimulation and Public Policy". Elsevier (1996)

[ICOOL]: International Coalition of Ocean Observing Laboratories.
www.i-cool.org/?cat=38

[JAVA] *Java Sun Microsystems, Inc*
http://Java.sun.com

[JAVBAS] "*Tutor Java Nivel Básico"*
http://www.programacion.com/Java/tutorial/Java_basico/

[JAVTUT] "*The Java Tutorials*"
http://Java.sun.com/docs/books/tutorial/

[LABXML] José Emilio Labra Gayo. "*Aplicaciones avanzadas de XML: Web Semántica.*" Universidad de Oviedo.

[LI2003] Liliana Di Prieto. "*Modelización de la dinámica de fluidos con autómatas celulares*". Noviembre 2003.

[MABS00]. Paul Davidsson. "*Multi Agent Based Simulation: Beyond Social Simulation*". Department of Software Engineering and Computer Science. University of Karlskrona/Ronneby. (2000)

[MAT2004] The MathWorks, Inc. *"Ayuda de Matlab, Versión 7.0.0.19920 (R14)".* Copyright 1984- 2004. Mayo 06, 2004.

[MIL06] Mills, David L. *Computer Network Time Synchronization: The Network Time Protocol*, Taylor & Francis / CRC Press. (Mayo 2006)

[MVS98]: MVS: Multi- Vehicle – Simulator.
http://underwater.iis.u-tokyo.ac.jp/mvs/mvs-top-e.html

[NETCDF] Unidata. *"NetCDF"*
http://www.unidata.ucar.edu/software/netcdf/

[NGDC] National Oceanic and Atmospheric Administration (NOAA). "*Nacional Geophysical Data Center*"
http://www.ngdc.noaa.gov/

[NGDC2] National Oceanic and Atmospheric Administration (NOAA). "*The Marine Geophysical Data Exchange Format.- MGD77*"
http://www.ngdc.noaa.gov/mgg/gdas/data/docs/mgd77.txt

[NI1991] Nick P. Fofonoff y Robert C. Millard, Jr. "*Calculation of Physical Properties of Seawater*". Woods Hole Oceanographic Institution. Julio de 1991.

[NOR2003] Nordlys Northern Lights, "*Realtime Measurements. Magnetometer*". 2003. www.northern-lights.no/english.

[OCEX] National Oceanic and Atmospheric Administration (NOAA). *"What are Auvs? Why are they used?"*
http://oceanexplorer.noaa.gov/explorations/08auvfest/background/auvs/auvs.html

[OCEAN] "*Ocean Floor Databases*"
http://ocean-ridge.ldeo.columbia.edu/

[OT2001]Fernando Ot y Silvia Lladó. "*Proyecto Inclinómetro*". 2001.

[PAGE91] B. Page. « *Diskrete Simulation: eine Einführung mit Modula-2"*. Springer Verlag Berlin, (1991).

[PARU98] Parunak H.V.D., Savit R., Riolo R.L.: "*Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and Users' Guide. In Sichman, J.S., Conte, R., Gilbert, N. (eds.),Multi-Agent Systems and Agent-Based Simulation"* Springer Verlag (1998)

[PHO01] S. Phoha, E. M. Peluso, R. L. Culver,"*A High-Fidelity Ocean Sampling Mobile Network (SAMON) Simulator Testbed for Evaluating Intelligent Control of Unmanned*

Underwater Vehicles", IEEE Journal of Oceanic Engineering, Vol. 26, No. 4, pp. 646-653, (2001)

[PET2005] Peter Schwartz, Michael Barad, Phillip Coella, Terry Ligocki. *"A Cartesian gris embedded boundary method for the heat equation and Poisson's equation in three dimensions*". University of California. 2 de Junio de 2005.

[PREE] Preecha P. Yupapin and Phongphan Rattanathanawan. "*Numerical Simulation of the Three-Dimensional Heat Equation and Applications"*. King Mongkut's Institute of Technology Lat Krabang. Bangkok. Thailand.

[RFC1832] R. Srinivasan working gropu. "XDR: External Data Representation Standard". Agosto 1995.

[RID04] P.Ridao, D.Ribas, E.Batle, E.Hernández. "*Simulation of physical agents. An application to underwater robots.*" (Marzo 2004)

[RID042]P. Ridao, E. Batle, D. Ribas, M. Carreras. "*Neptune: a HIL simulator for Multiple UUVs"*. Universidad de Girona. 2004

[RO2005] Robert H. Stewart. "*Temperature, Salinity, and Density"*. Department of Oceanography, Texas A&M University. 2005
http://oceanworld.tamu.edu/resources/ocng_textbook/chapter06/chapter06_01.htm

[ROY00] Roy M. Turner. "*An Intelligent, Context-Sensitive AUV Mision Controller for Standalone and AOSN Missions*". (2000)

[SAND1996] David T. Sandwell, Walter H. F. Smith "Global Bathymetric Prediction for Ocean Modelling and Marine Geophysics". 1996
http://topex.ucsd.edu/marine_topo/text/topo.html

[SHOABAT] *"Bases de datos Batimétricos"*
http://www.shoa.cl/cendoc-jsp/hidro2/index.htm

[SON03] F. Song, P.E. An, A. Folleco, "*Modeling and Simulation of Autonomous Underwater Vehicles: Design and Implementation*", IEEE Journal of Oceanic Engineering, Vol. 28, No. 2, April 2003.

[TH2004] Thomas Bräun, Adrian Boeing, Louis Gonzales, Andreas Koestler, Minh Nguyen, Joshua Petitt. "*The Autonomous Underwater Vehicle Initiative– Project Mako*". 2004 IEEE Conference on Robotics, Automation, and Mechatronics (IEEE-RAM), Diciembre. 2004.

[UNE1981] Millero, F. J. , Poisson A. *"Summary of data treatement for the Unesco one atmosphere equation of state of seawater".* 1981

[UNNE] "*Curso de Electricidad y Magnetismo. Guía de Laboratorio: magnetómetro-brújula de tangentes*". http://exa.unne.edu.ar/depar/areas/fisica/electymagne/LABORATORIOS/L10_BRUJULA/L10_BRUJULA.doc

[VTERRAIN] *"Undersea Terrain Elevation (Bathymetry)"* http://www.vterrain.org/Elevation/Bathy/index.html

[XML11] *Recomendación XML (Extensible Markup Language (XML) 1.1) W3C (World Wide Web Consortium),* http://www.w3.org, http://www.w3.org/TR/2004/REC-xml11-20040204 (Manual XML)

[XMLSCH] *Recomendación Esquemas XML (XML Schema Part 0, 1 y 2) W3C (World Wide Web Consortium),* http://www.w3.org , http://www.w3.org/TR/2004/REC-xmlschema-0-20041028 , http://www.w3.org/TR/2004/REC-xmlschema-1-20041028 , http://www.w3.org/TR/2004/REC-xmlschema-2-20041028

PARTE IV

Apéndices

ANEXO I.- *Colaboraciones y Otras Contribuciones*

1.- Colaboraciones

De forma colateral al proyecto, se han hecho colaboraciones que son de interés mencionar.

Primeramente y como se ha nombrado a lo largo del proyecto, éste se enmarca dentro de un grupo de proyectos final de carrera propuestos por el SIANI dentro de la ULPGC para el uso de la Ingeniaría Informática aplicado al estudio oceanográfico a través de Vehículos Submarinos Autónomos (AUVs).

Se ha colaborado estrechamente con los siguientes proyectos finales de carrera:

- *Sistema Integrado de Control para un Vehículo Submarino Autónomo*, desarrollado y ya presentado por **Enrique Fernández Perdomo**. El proyecto se centra en el análisis, diseño e implementación del sistema de control de un AUV en CoolBot, dando como resultado el *SickAUV*. La relación con el simulador es estrecha, ya que en esta primera fase, se hace ideal que el *SickAUV* simule sensores no disponibles y entornos de simulación a través del presente proyecto.

- *Sistema Integrado de Planificación y Control de Misiones con Vehículos Submarinos Autónomos*, desarrollado por **Eliezer Ramírez Cabrera**. El proyecto se centra en el análisis, diseño e implementación de una aplicación que permita al usuario generar planes de misiones de forma sencilla, compilarlos y cargarlos en el mismo. Esta aplicación también debe tener capacidades para comandar al AUV y monitorizar su actividad. La relación con el simulador es indirecta, ya que comanda AUVs que pueden ser simulados con los resultados del presente proyecto. Afectará más a la parte del presente proyecto que trata el diseño de AUVs simulados, ya que debe ser capaz de integrar y seguir los planes diseñados desde el proyecto nombrado.

En la estrecha colaboración entre los 3 proyectos (sobre todo en las fases iniciales) se han obtenido estándares de formatos y representación, como son:

- Formato de representación de mapas Batimétricos en netCDF.
- Formato de especificación de misiones y AUVs en XML.
- Formato de intercambio de paquetes en XDR.
- Protocolo de comunicación para conexión al Simulador.

2.- Otras Contribuciones

En el desarrollo del presente proyecto, se han realizado otras contribuciones, como son:

- *Estudio del Sónar de Barrido Lateral*: se ha hecho un estudio en profundidad de este tipo de sónar, focalizado el estudio en su simulación. Se adjunta ese estudio como uno de los anexos de este proyecto.

- *Uso de Moddle* para *gestión de contenidos y creación de un entorno de colaboración*: en las primeras fases del presente proyecto, donde la colaboración con otros proyectos ha sido más estrecha, se ha utilizado esta herramienta educativa para:

    - Entrega y compartición de documentos de análisis con el resto de colaboradores y tutores del proyecto.
    - Planificación de reuniones y gestión de informes de cada reunión.
    - Foros para la discusión de temas comunes.

ANEXO II.- *Patrones de Diseño utilizados*

Por motivos de calidad, en el diseño del proyecto se han utilizado los patrones de diseño que se detallan en ese anexo.

***Patrón de Diseño Singleton*:** sirve para restringir el número de instancias que puede tener una clase a uno, de forma que cualquier otra clase del sistema que tenga acceso a la clase, podrá acceder al único objeto instanciado de la misma. Se ha utilizado por ejemplo en la Pizarra común a todos los componentes Gestores del Servidor del Entorno (ver **Capítulo 10**).

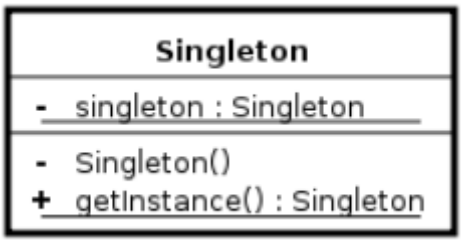


Figura 231: Diagrama UML de clase Singleton.

Patrón de Diseño Objeto Nulo: en proyectos cuyo diseño e implementación es orientado a objetos, puede suceder que un objeto sea nulo. Sin el patrón, en todo punto en el que se utiliza el objeto habría que introducir una comprobación para comprobar si el objeto es nulo (y por tanto no se pueden usar sus métodos) o no.

El patrón simplifica el código y mantenimiento: en vez de hacer comprobación en todo punto, el objeto nunca adopta el valor nulo, de manera que, cuando no tiene objeto instanciado, hereda de una clase que no implementa métodos. Así sus métodos se pueden invocarse, pero no realizan ninguna acción.

Se ha utilizado por ejemplo en el desarrollo de la clase **Proceso Ejecutor**. Se puede revisar en el **Manual de Implementación del Servidor del Entorno** en los documentos anexos al proyecto.

***Patrón de diseño Prototipo*:** su finalidad es crear nuevos objetos clonándolos de una instancia creada previamente. Cada objeto es en sí mismo una fábrica especializada en construir objetos iguales a sí mismos.

Facilita el mantenimiento e implementación, ya que sin este prototipo, cada vez que se instancia un objeto, éste está completamente vacío, por lo que

tienen que "rellenarse" todos sus atributos desde cero. De esta manera, si todos los objetos instanciados tienen características y atributos comunes según el contexto, se pueden clonar.

Se ha utilizado en la creación de nuevos mensajes de tipo petición, respuesta, informe y sincronización por parte del Front-End, basado en que todos los mensajes de tipo petición comparten características, como los respuesta, etc. Se puede consultar el **Manual de Implementación del Servidor del Entorno** para más información.

Patrón de diseño Observador Observable: permite resolver el problema de que una clase necesita conocer un cambio en un modelo que observa (Subject en la Figura 232) para poder actualizarse.

Por ejemplo, en el presente proyecto, el servidor del Entorno tiene una serie de parámetros globales, que los distintos componentes Gestores consumen y copian localmente. El problema es que si un cliente cambia el valor de cualquier parámetro global, los componentes Gestores que lo han copiado localmente tienen que conocer el cambio para actualizarlo localmente. Se puede consultar en el **Manual de Implementación del Servidor del Entorno** anexo al proyecto.

Este patrón soluciona el problema, de forma que todo observador (clase que necesita conocer cambios) se registra en un observable (clase que puede modificar sus valores). Cuando se produce un cambio en el observable, este notifica automáticamente a todos sus subscriptores, que toman el nuevo valor.

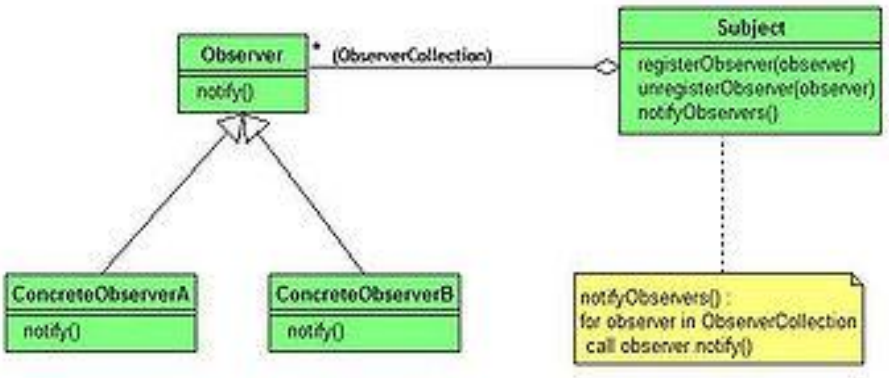


Figura 232: Patrón de diseño Observador Observable.

Patrón de Diseño Controlador Vista Modelo (MVC): basado en el patrón de diseño observador Observable, pero más focalizado en aplicaciones gráficas, resuelve un problema muy común en las interfaces gráficas de Usuario: un mismo modelo de datos puede tener varias vistas gráficas distintas, donde en cada una el usuario parametriza cómo desea que sea la visualización. Un cambio en los modelos debería actualizar todas las vistas, sin que se pierda la parametrización fijada por el usuario para cada vista.

La solución es dividir la aplicación en tres componentes: datos, gráficos y controlador.

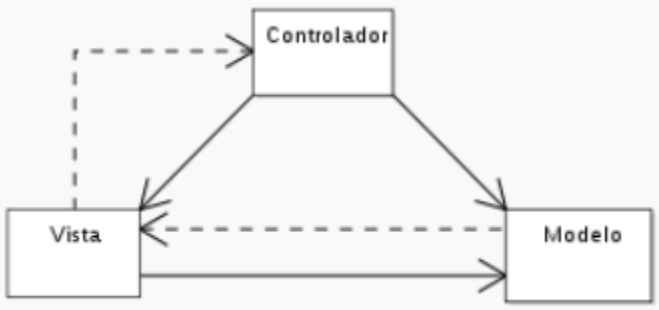


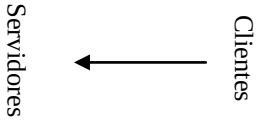
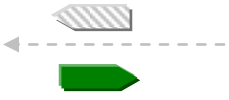
Figura 233: Patrón de diseño MVC.

Se ha utilizado en el desarrollo del prototipo de Interfaz Gráfica de Usuarios, descrito en el **Capítulo 12**: así, cuando la tabla de flota o de parámetros fisicoquímicos es modificada, automáticamente es mostrado en el Google Maps (la vista), tras el proceso del núcleo de la aplicación (controlador).

ANEXO III.- Diagramas de Representación utilizados.

Diagramas de Flujo de Información entre Subsistemas

El objetivo es representar la relación que existe entre subsistemas de la arquitectura ***SUBES***.

Elemento	Imagen	Significado
Subsistema Izquierda		Subsistema objeto de la representación
Subsistema Derecha		Subsistemas que se comunican con el Subsistema Izquierda.
Canal de Comunicación		En la punta de la flecha se sitúan los subsistemas que suministran servicios (servidor), mientras a la derecha los que consumen esos servicios (clientes). La flecha no siempre indica la dirección de los elementos de comunicación, como es en los **TIC**, **RES** y **DATA** que justo siguen el camino inverso (del servidor al cliente)
Acción		Elemento de comunicación que se envía en la parte superior del canal de comunicación
Reacción		Elemento de comunicación que se desencadena por la Acción que se sitúa justo encima de la flecha y de la

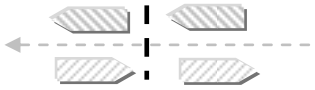
		propia Reacción.
Cardinalidad		A ambos extremos, indican: - Si sólo un número: mínimo y máximo número de elementos. - Si dos números: el primero es el mínimo, y el segundo el máximo (* es múltiples).
Columnas Verticales Discontinuas		Separa verticalmente distintos tipos de comunicación entre origen y destino.

Tabla 47: Símbolos del Diagrama de Flujo de Información entre Subsistemas.

Diagramas UML de Casos de Uso

Tienen como finalidad mostrar las necesidades que pueden tener distintos roles, y cómo la aplicación responde ante dichas necesidades reales. Es de utilidad en fases de análisis.

En el cuadro explicativo siguiente describimos cada uno de los elementos que conforman un diagrama UML de Casos de Uso, para que los menos habituados puedan interpretar los diagramas de caso de uso utilizados:

Elemento	Descripción	Representación Gráfica
Roles	Entidades que hacen uso del sistema de algún caso de uso.	
Casos de Uso	Cada proceso que un rol pide o ejecuta en el sistema. Se puede entender como servicios. Normalmente interrelacionados entre sí y con los roles.	

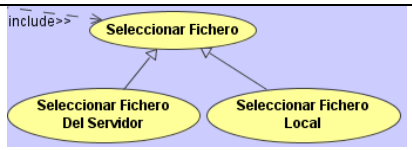
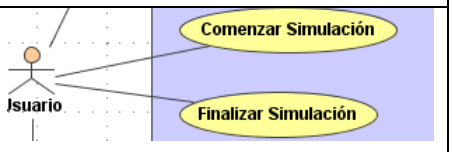
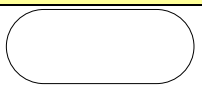
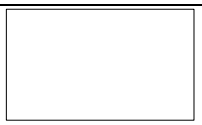
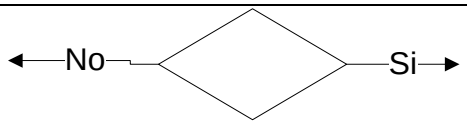
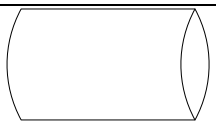
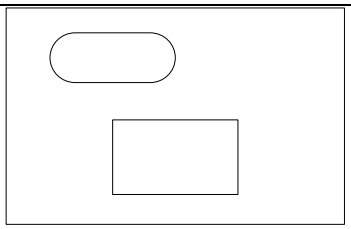
Include	Con una flecha, significa que el caso de uso al que llega la flecha es necesario ser ejecutado durante la ejecución del que parte la flecha.	
Uses	Con una flecha, significa que el caso de uso al que llega puede ser usado opcionalmente durante la ejecución del caso de uso del final de la flecha.	
Extend	Con una flecha, significa que el caso de uso del que parte la flecha extiende la funcionalidad el caso de uso al que llega. Es decir, hace las mismas funciones pero ampliadas en algún aspecto para adaptarse a una situación.	
Herencia	En la parte del triángulo es el padre, y el otro caso de uso el Hijo. El hijo hereda todas las funcionalidades del padre, de manera que tiene las mismas funciones ampliadas o modificadas en algún aspecto para adaptarse a una situación. La diferencia con Extend, es que el hijo puede sustituir al padre sin influir en el flujo del programa; los Extend no.	
Relación-Ejecución	Permite señalar que casos de uso son los que ejecutan cada rol (y relaciona casos de uso con rol por tanto).	

Tabla 48: Diagrama UML de casos de usos.

Diagramas UML de Flujo de Datos

Permite expresar el lazo de control de la aplicación en ciertos puntos. Se usa por tanto en fases de diseño.

En el cuadro explicativo siguiente describimos cada uno de los elementos que conforman un diagrama UML de Flujo de Datos, para que los menos habituados puedan interpretar los diagramas de caso de uso utilizados:

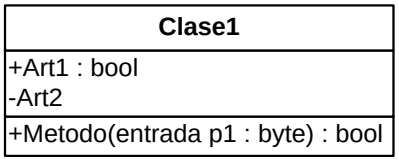
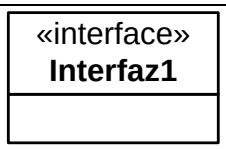
Elemento	Descripción	Representación Gráfica
Estado Inicial / Final	Estado inicial del flujo o final del mismo.	
Proceso de datos.	Estado en el que se realiza alguna acción en el sistema.	
Cuadro de Decisión	Estado en el que, en función de ciertos atributos, se decide la dirección del flujo de información.	
Datos	Modelos de datos y Bases de Datos donde se almacena y lee la información.	
Grupo	Indica un grupo de acciones que pueden ser tratadas como un todo. Clarifica el diagrama y permite simplificaciones.	
Fleja de Flujo	Conecta los distintos bloques antes presentados. El origen indica el proceso del que se parte, y en la flecha el nuevo bloque al que deriva el flujo de la información. En ciertos diagramas, se ha	

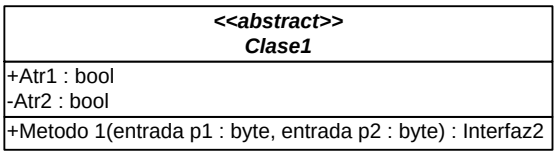
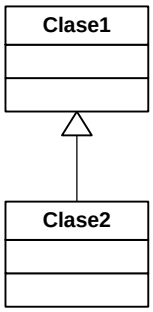
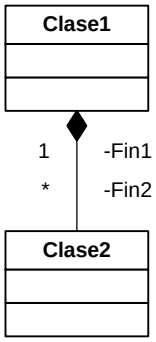
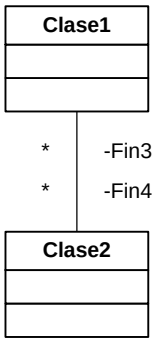
	utilizado esta flecha de forma no estándar, con colores, distintos grosores y líneas suspensivas, para indicar caminos opcionales, u otros aspectos.	

Tabla 49: Diagrama UML de flujo de datos.

Diagrama UML de Clases

Se ha utilizado en el proyecto para representar el diseño orientado a objetos de la aplicación en ciertos puntos de la misma. Si bien el estándar contempla muchas más información a representar, en el presente proyecto se ha utilizado de forma básica:

Elemento	Descripción	Representación Gráfica
Clase Básica	Clase, donde se presentan: - Atributos: el nombre del atributo seguido del tipo. Con + los públicos y con – los privados. - Métodos: con los parámetros (y tipo de cada uno) y finalmente el tipo devuelto. Antecedido por un + indica que es público, y privado en el caso de -.	
Clase Interfaz	El nombre de la clase se antecede de la etiqueta <<interface>>. Solo presenta métodos (los prototipos de la	

	interfaz).	
Clase Abstracta	Se representa con el nombre de la clase en cursiva. De forma opcional (y no siguiendo el estándar) en ciertas partes de este proyecto se ha añadido la etiqueta <<abstract>> por mayor claridad.	
Herencia	Indica que la clase 2 hereda de la clase 1 sus atributos y métodos.	
Composición	Indica que algún atributo de la Clase 1 se forma de objetos de la Clase 2.	
Asociación	Indica que algún método o atributo de la Clase 1 está asociado con la Clase 2 (lo utiliza en un determinado punto, etc.).	

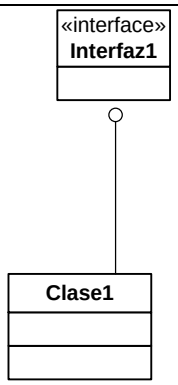
Interfaz	Sirve para indicar que la Clase 1 implementa la Interfaz1.	

Tabla 50: Diagrama UML de Clases.

ANEXO IV.- Otros documentos anexos

Se adjuntan al proyecto documentos completos que han sido desarrollados en el transcurso del proyecto, y pueden ser de utilidad para ampliar la visión dada en el proyecto. Son:

Manuales de Usuario: Documento con dos manuales para instalar y probar las dos aplicaciones resultados del presente proyecto: el Servidor del Entorno y la Interfaz Gráfica de Usuario.

Manual de Implementación del Servidor del Entorno: extenso documento con gran detalle en las explicaciones en aspectos de implementación y diseño. De gran utilidad para proyectos final de carrera que se basen en los resultados del presente.

***Manual de Análisis de Requisitos de Software*:** Este documento es el resultado del análisis desarrollado iterativamente durante el proyecto. Son diagramas de Casos de Uso y prototipos que describen lo que es espera del software.

Especificación de Misiones de AUVs_XML_Y_XSD: documento redactado y aprobado por Enrique Fernández Perdomo, Eliezer Ramírez Cabrera, por el autor de este proyecto y por los tutores del presente proyecto. Dan una descripción detallada de la especificación en ficheros XML y XSD de misiones individuales de AUVs.

Estudio de Sónar de Barrido Lateral Completo: se puede encontrar en la carpeta **Sónar de Barrido Lateral**, y es el proyecto completo entregado para la asignatura de Sistemas Robóticos Móviles y propuesto a partir del presente proyecto al profesor de la asignatura, con detalles de implementación incluidos. También se pueden encontrar los ficheros fuentes en Matlab para poder analizar los resultados.

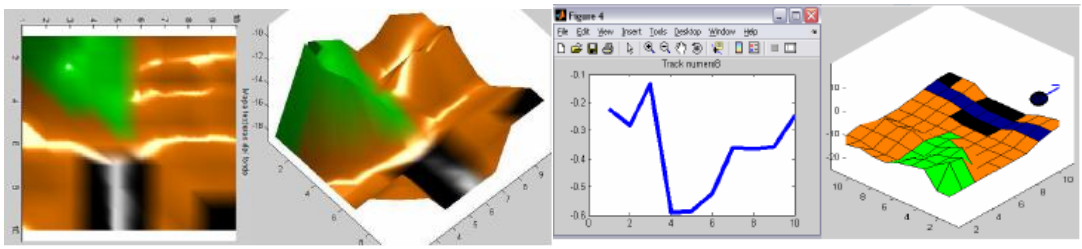


Figura 234: Resultados del Estudio de Sónar de Barrido Lateral anexo.

Descripción Técnica de Formatos de Datos Utilizados: se describen los distintos formatos utilizados, como es netCDF, XDR, XML y XSD, etc. Sin duda de interés para el usuario que quiera ahondar en los formatos de ficheros empleados en el presente proyecto.

ANEXO VIII.- Jerarquía de Ficheros Entregados

Se entrega en el CD adjunto al presente proyecto la siguiente información:

Carpeta de **Documentación**

- PFC_ Rodrigo Heredero Robayna_ Entorno Simulación para el Desarrollo de Misiones con Vehículos Submarinos Autónomos.pdf: *Presente documento.*
- Resumen_PFC.pdf: *Resumen en castellano del presente PFC*
- Summary_PFC.pdf: *Resumen en inglés del presente PFC*
- Carpeta **Documentos Anexos**
    - Manuales de Usuario: *manuales de usuario de Servidor del Entorno y de Interfaz Gráfica de Usuario*
    - Manual de Implementación del Servidor del Entorno.pdf: *manual completo de implementación del Servidor del Entorno.*
    - Manual de Análisis de Requisitos de Software.pdf: *Análisis de requisitos en fases tempranas del proyecto.*
    - Carpeta **Misiones**
        - Especificación de Misiones de AUVs_XML_Y_XSD.pdf: *Manual de planes de misión diseñados.*
        - Carpeta **Misiones:** *conjunto de ficheros XML y XSD de misiones ejemplos, siguiendo las directrices de la especificación anterior.*
    - Carpeta de **Sónar de Barrido Lateral**.
        - Estudio de Sónar de Barrido Lateral.pdf: *trabajo entregado para Sistemas Robóticos móviles relacionado con el PFC.*
        - Carpeta de **Información**: *bibliografía.*

        - Carpeta de **Código**: *fuentes en Matlab*.
    - Carpeta de **Bibliografía:** *fuentes bibliográficas usadas en el proyecto final de carrera.*

Carpeta de **Instalación**

- SUBES Servidor.msi: *Instalar el Servidor del Entorno.*
- SUBES Interfaz Gráfica de Usuario.msi: *Instalar la Interfaz Gráfica de Usuario.*
- Otras **aplicaciones requeridas:** *otros instaladores que pueden hacer falta que funcionen los instalables.*

Carpeta de **Fuentes**

- Carpeta **Compilados:** *ficheros Java ya compilados (*.class).*
- Carpeta **Fuentes:** *ficheros Java sin compilar (*.java)*
- Carpeta **Librerías:** *conjunto de librerías necesarias para compilar y ejecutar las aplicaciones resultado.*