Universitat Oberta de Catalunya

PFC-Xarxes de computadors

Enginyeria en Informàtica

Sistema de adquisición, almacenamiento y distribución de datos con sistema ARGOS

Estudiante: Fernando Pérez López
Consultor: Jordi Ceballos Villach
Estudios: Enginyeria en Informàtica
Fecha: 3 de junio de 2011

Sistema de adquisición, almacenamiento y distribución de datos con sistema ARGOS

Fernando Pérez López

3 de junio de 2011

Índice

1. Introducción **5**
1.1. Contexto del proyecto . . . 5
1.2. Descripción del proyecto . . . 6
1.3. Funcionalidades . . . 7
1.4. Objetivos . . . 7

2. Planificación **8**
2.1. Ciclo de vida . . . 8
2.2. Descripción de las Tareas . . . 8
2.2.1. Inicio y Planificación . . . 8
2.2.2. Análisis de requisitos . . . 9
2.2.3. Diseño . . . 9
2.2.4. Implementación y pruebas . . . 10
2.2.5. Cierre del Proyecto . . . 11
2.3. Temporización . . . 11
2.4. Análisis de riesgos . . . 13

3. Análisis de requisitos **14**
3.1. Requisitos no funcionales . . . 14
3.1.1. Escenario de partida . . . 14
3.1.2. Requisitos de interfaz . . . 19
3.1.3. Requisitos de seguridad . . . 19
3.1.4. Requisitos de hardware . . . 20
3.1.5. Requisitos de software . . . 20
3.2. Requisitos funcionales . . . 20
3.2.1. Funcionalidades de las aplicaciones en el Servidor ICM . . . 21
3.2.2. Funcionalidades de la aplicación móvil . . . 21

4. Diseño **22**
4.1. Diseño arquitectónico . . . 22
4.1.1. Vista física . . . 22
4.1.2. Vista de despliegue . . . 23
4.2. Diseño de clases . . . 26
4.3. Diseño de la persistencia . . . 26
4.3.1. Detalle de las tablas . . . 27

4.4. Tecnologías a usar . . . 29
4.5. Diseño aplicación móvil . . . 30
4.5.1. Sección Select Buoy . . . 30
4.5.2. Sección View Map . . . 31
4.5.3. Sección Measure Detail . . . 32
4.5.4. Sección About . . . 34

5. Implementación **35**
5.1. Persistencia . . . 35
5.2. Servidor de aplicaciones ICM Server . . . 35
5.2.1. THREDDS . . . 36
5.2.2. ncWMS . . . 38
5.3. Aplicación Buoys ICM Server . . . 39
5.3.1. Descarga de Datos . . . 42
5.3.2. Transformación de Datos . . . 43
5.3.3. Inserción en Base de Datos . . . 44
5.3.4. Acceso Base de Datos . . . 47
5.3.5. Exportación de datos . . . 49
5.3.6. Interfaz aplicación móvil . . . 56
5.4. Aplicación Buoys dispositivo móvil iOS . . . 56
5.4.1. Select Buoy . . . 58
5.4.2. About . . . 60
5.4.3. View Map . . . 61
5.4.4. Measure Detail . . . 65

6. Conclusiones **66**

7. Trabajos futuros **66**

8. Bibliografía **68**

Glosario **70**

Índice de tablas

1. Tareas fase Inicio y Planificación 9
2. Tareas fase Análisis de requisitos 9
3. Tareas fase Diseño 9
4. Tareas fase Implementación y pruebas 10
5. Tareas fase Cierre del Proyecto 11
6. Riesgos del Proyecto 13
7. Sistema Doppler vs. posicionamiento con GPS 16
8. Tabla codificación transmisiones sistema ARGOS 17
9. Tabla significado tramas sistema ARGOS 19

Índice de figuras

1. Arquitectura global del sistema . . . 6
2. Temporización y diagrama de Gantt . . . 12
3. Sistema de posicionamiento por efecto Doppler (*Fuente: CLS-ARGOS*). . . . 15
4. Distribución de las estaciones receptoras del sistema ARGOS. . . . 15
5. Página de entrada al website del sistema ARGOS. . . . 17
6. Vista física del sistema . . . 23
7. Diagrama despliegue del sistema . . . 24
8. Diagrama estático de diseño ICM Server . . . 26
9. Diseño de la base de datos central del sistema . . . 27
10. Diseño de esquema fichero Property List . . . 28
11. Sección Select Buoys . . . 30
12. Sección View Map . . . 32
13. Sección Measure Detail . . . 33
14. Sección About . . . 34
15. Consulta de datos en THREDDS . . . 38
16. Diagrama de clases 1 Buoys ICM Server . . . 40
17. Diagrama de clases 2 Buoys ICM Server . . . 40
18. Diagrama de secuencia Buoys ICM Server . . . 41
19. Fichero KML exportado en GoogleEarth . . . 51
20. Diagrama de clases 1 Buoys iOS . . . 57
21. Diagrama de clases 2 Buoys iOS . . . 57
22. Implementación Select Buoy: XCode Interface Builder . . . 58

1. Introducción

1.1. Contexto del proyecto

Una de las técnicas de observación y monitoreo del océano consiste en la utilización de boyas a la deriva (transportadas por la corriente) dotadas de sensores de medida de propiedades físico-químicas del agua (programa ARGO[1]). Estas boyas registran estas propiedades y se comunican vía satélite para la transmisión de los datos, incluida su posición, haciendo medidas o bien en superficie de forma continua o bien ejecutando perfiles verticales de datos hasta profundidades prefijadas. En algunos casos para reducir costes, tanto energéticos como de transmisión, la georreferenciación se fija por el corrimiento Doppler de la frecuencia de una señal que emite la boya al ser "escuchada" por una serie de satélites (sistema ARGOS[2] implementado en la serie de satélites NOAA[3]).

Los datos son almacenadas tras un proceso de control de calidad en un servicio de distribución (en Europa a través del CLS[4] en Toulouse, Francia). Dicho servicio está personalizado por usuario y ofrece acceso a los datos a través de una página web donde se pueden descargar una visualización de los mismos e incluso los datos. Este procedimiento es manual y se proporciona en un formato óptimo para la transmisión vía satélite pero poco práctico para la explotación científica de los mismos. Además la codificación de los datos excepto en el caso de la georreferenciación (posición y tiempo, usualmente a través de GPS) es particular de cada sensor y depende del diseño del sistema de adquisición implementado en la boya por el usuario (habitualmente cadenas en hexadecimal), lo cual implica que los datos deben ser postprocesados a posteriori por los usuarios para obtener los valores físicos de las variables de medida. En algunos casos la información es pausada en el tiempo y su disponibilidad y acceso puede no ser un factor relevante (por ejemplo seguimiento de aves, cetáceos, tortugas o en determinados experimentos a largo plazo). Sin embargo, en tareas operacionales como seguimiento de contaminantes en el mar, sistemas operacionales de predicción de corrientes, naufragios, etc., la accesibilidad en tiempo casi real es un factor clave para la toma de decisiones. Dichos servicios no están contemplados por los sistemas de adquisición y distribución y solamente son facilitados como un volcado RAW[5] de los datos, tal y como son transmitidos a través de un servidor FTP.

[1] Programa ARGO: http://www.argo.net
[2] ARGOS: http://www.argos-system.org
[3] National Oceanic and Atmospheric Administration: http://www.noaa.gov
[4] Collective Satellite Localization: http://www.cls.fr
[5] Datos sin procesar o con un procesamiento mínimo

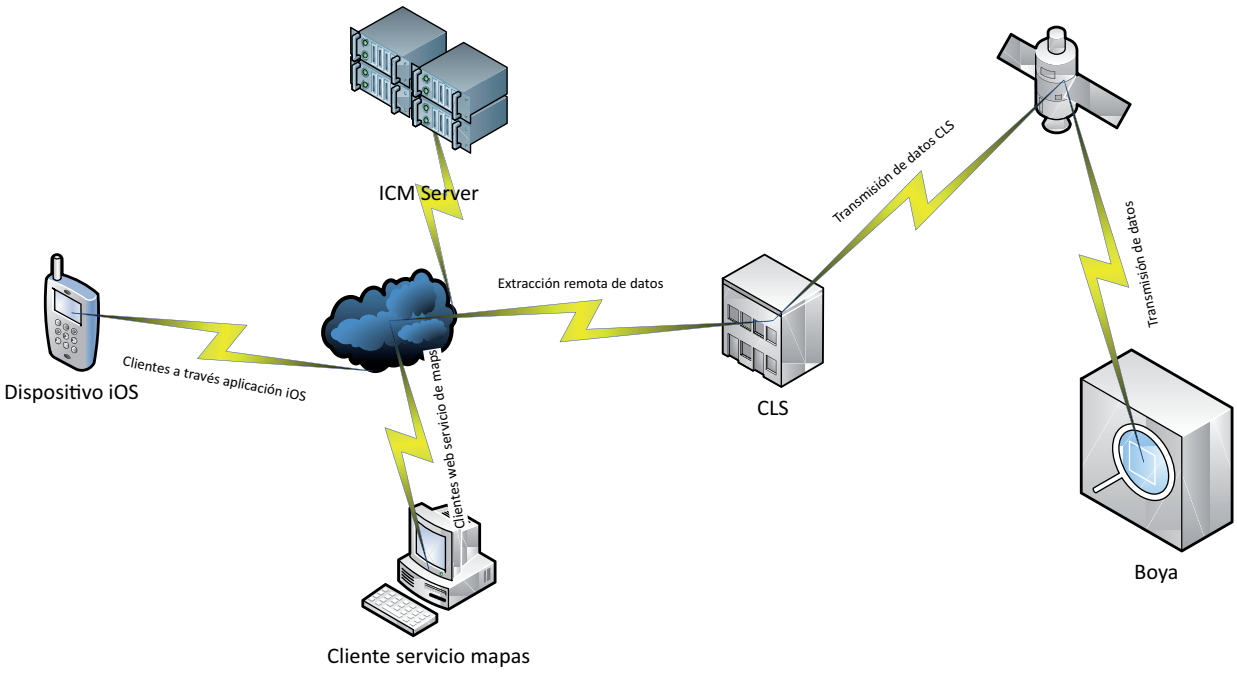

Figura 1: Arquitectura global del sistema

1.2. Descripción del proyecto

Se propone desarrollar un sistema automatizado para el seguimiento de estas boyas que permita la descarga automatizada de sus datos, su transformación en formatos estándar, inserción de estos datos en un SGBD y su visualización rápida vía web y, también, a través de una pequeña aplicación para dispositivos móviles.

Para ello se cuenta con la infraestructura informática de la que dispone el Institut de Ciències del Mar[6], que entre otros permite disponer de un SGBD PostgreSQL[7], diferentes estaciones de trabajo, tanto con sistemas operativos Linux y Mac OS X, y un servidor Blade IBM para virtualización que permite la creación de servidores virtuales.

Una visión general del sistema que se propone, dentro del contexto el proyecto, se muestra en la Figura 1

[6]Institut de Ciències del Mar: http://www.icm.csic.es
[7]PostgreSQL: http://www.postgresql.org

1.3. Funcionalidades

El sistema a desarrollar debe cumplir las siguientes funcionalidades:

1. Descarga automática y actualización de tiempo de observación, posiciones georrefenciadas y datos físicos de los sensores: la descarga de los datos se debe realizar de forma automatizada para cada boya.
2. Transformación de formatos y exportación de los datos a diversos formatos: por ejemplo NetCDF[8] y KML[9].
3. Diseños de visualización estática y dinámica de la información en formatos KML según especificaciones del Open Geospatial Consortium (OGC)[10].
4. Diseño de bases de datos para inserción de datos usando SGBD PostgreSQL: se debe crear una base de datos bajo el SGBD PostgreSQL así como las operaciones de consulta, extracción y actualización en la base de datos.
5. Implementación de un sistema de distribución basado en servicios ncWMS[11] y THREDDS [12] para la visualización de los datos extraídos de las boyas.
6. Instalación de todo el sistema en una máquina virtual bajo Xen[13]: esto incluye creación de máquina virtual, instalación y configuración de servidor en esta máquina virtual.
7. Creación de un pequeño aplicativo para iOS[14] que permita el acceso y visualización de datos extraídos de las boyas: se pretende experimentar con dispositivos móviles y ver sus posibilidades para la visualización de datos.

1.4. Objetivos

Los objetivos que se pretende conseguir al escoger este proyecto son:

- Poner en práctica los conocimientos adquiridos a lo largo de la titulación de Ingeniería en Informática de la UOC.

[8] NetCDF: http://www.unidata.ucar.edu/software/netcdf
[9] Keyhole Markup Language: http://www.unidata.ucar.edu/software/netcdf
[10] OGC: http://www.opengeospatial.org
[11] ncWMS: http://www.resc.rdg.ac.uk/trac/ncWMS
[12] THREDDS: http://www.unidata.ucar.edu/projects/THREDDS
[13] Xen: http://www.xen.org
[14] iOS: http://www.apple.com/ios

- Integrar diferentes tecnologías y dispositivos en un mismo sistema.
- Aprender a desarrollar aplicaciones para dispositivos móviles que usen el sistema operativo iOS.
- Trabajar con formatos estandard de visualización y almacenamiento de datos.
- Aprender a realizar textos de alta calidad con sistemas de composición de texto profesionales y de alta calidad. Por ello toda la documentación relativa a este proyecto será generada con LaTeX[15] .

2. Planificación

2.1. Ciclo de vida

El proyecto será desarrollado siguiendo el ciclo de vida clásico o también llamado en cascada, adaptado a las necesidades de este Proyecto Fin de Carrera:

A lo largo de todas las fases se irá generando documentación, que será entregada en las diferentes entregas parciales, PACs, y en una entrega final donde se entregará la Memoria del Proyecto Fin de Carrera.

2.2. Descripción de las Tareas

A continuación se procede a hacer una descripción detallada de cada una de las fases, las tareas a realizar, así como su temporización.

2.2.1. Inicio y Planificación

En este fase se elige el proyecto así como las líneas generales que lo conformarán: funcionalidades, propuesta y planificación.

La descripción de las tareas a realizar se muestran en la Tabla 1

[15] LaTeX: http://www.ctan.org

Tarea	Descripción
Selección proyecto	Selección del Proyecto entre diferentes alternativas
Definición alcance proyecto	Definición del alcance del proyecto
Elaboración propuesta proyecto	Redacción propuesta formal proyecto
Entrega propuesta proyecto	Envío propuesta proyecto al Consultor
Elaboración plan de trabajo	Elaboración de la temporización del proyecto y su planificación conforme a las planificación de la asignatura de PFC
Entrega PAC1	Envío del documento de la PAC1

Tabla 1: Tareas fase Inicio y Planificación

2.2.2. Análisis de requisitos

Durante esta fase se terminarán de cerrar los requisitos que ha de cumplir el sistema a nivel funcional y operacional.

La descripción de las tareas a realizar se muestran en la Tabla 2

Tarea	Descripción
Especificación requisitos funcionales	Creación y documentación de casos de uso
Especificación requisitos no funcionales	Identificar y documentar otros requisitos no funcionales

Tabla 2: Tareas fase Análisis de requisitos

2.2.3. Diseño

En esta fase se procederá a diseñar todo los elementos del sistema informático de forma que se cubran los requerimientos y objetivos del proyecto.

La descripción de las tareas a realizar se muestran en la Tabla 3

Tarea	Descripción
Diseño arquitectónico	Diseño de la arquitectura del sistema
Diseño de clases	Creación diagramas estáticos software del sistema
Diseño de la persistencia	Diseño de la base de datos del sistema
Estudio alternativas tecnológicas	Revisión alternativas tecnológicas de implementación software lado servidor
Redacción PAC2	Elaboración documentación PAC2
Entrega PAC2	Envío del documento de la PAC2

Tabla 3: Tareas fase Diseño

2.2.4. Implementación y pruebas

En esta fase se procederá a desarrollar, probar e implementar el sistema diseñado en la fase anterior de Diseño.

La descripción de las tareas a realizar se muestran en la Tabla 4

Tarea	Descripción
Desarrollo Aplicaciones ICM Server	Desarrollo e implementación de las aplicaciones lado servidor: ▪ Base de datos. ▪ Operaciones de inserción y extracción BD. ▪ Aplicaciones descarga automática de datos. ▪ Aplicaciones transformación de datos. ▪ Instalación de servidor virtual. ▪ Instalación y personalización servicio de mapas. ▪ Pruebas unitarias.
Desarrollo aplicación móvil iOS	Desarrollo de: ▪ Aplicación móvil visualización de datos. ▪ Operaciones de acceso a datos del sistema.
Pruebas de integración	Pruebas de integración de todos los componentes del sistema.
Preparación entrega PAC3	Elaboración documentación y entregables PAC3
Entrega PAC3	Envio del documento y entregables de la PAC3

Tabla 4: Tareas fase Implementación y pruebas

2.2.5. Cierre del Proyecto

En un proyecto TIC esta fase se aplica a, entre otros, la entrega final del producto, su instalación y documentación desarrollados durante el proyecto, así como la aceptación formal por parte del cliente. Por lo tanto en esta fase final se finalizará la elaboración de toda la documentación del proyecto.

La descripción de las tareas a realizar se muestran en la Tabla 5

Tarea	Descripción
Redacción memoria	Redacción Memoria Final del Proyecto
Preparación presentación virtual	Elaboración de presentación virtual
Preparación entregable producto	Paquetización productos desarrollados
Entrega final	Envío de la Memoria, presentación y entregables

Tabla 5: Tareas fase Cierre del Proyecto

2.3. Temporización

A continuación, en la Figura 2 , se muestra la temporización de todas las fases y tareas mostradas anteriormente, así como su diagrama de Gantt.

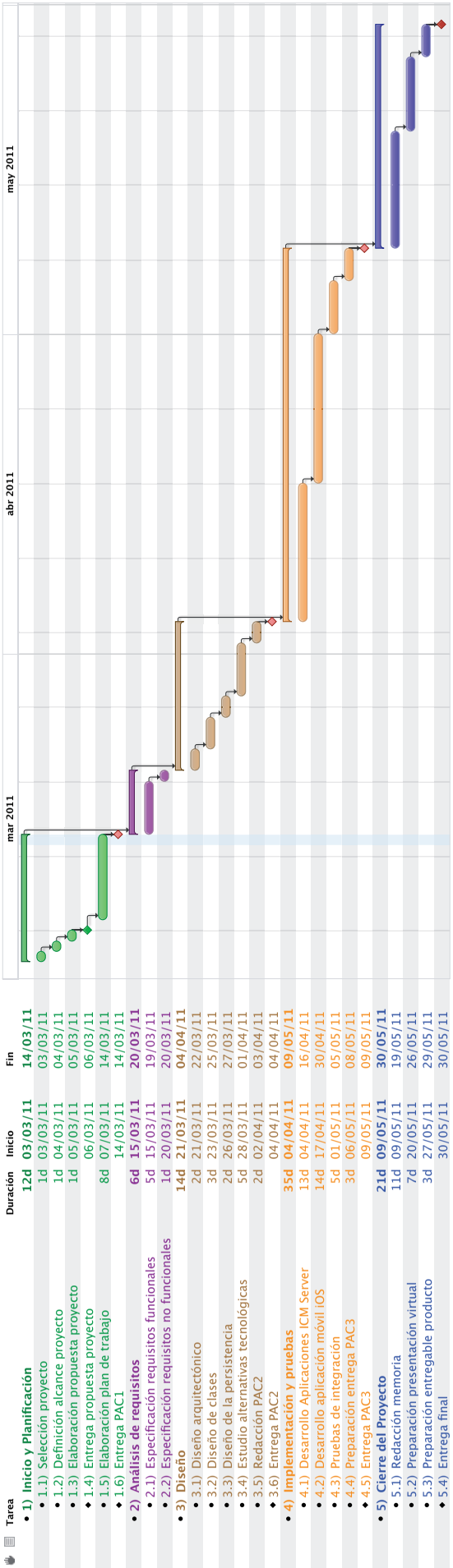

Figura 2: Temporización y diagrama de Gantt

2.4. Análisis de riesgos

En todo proyecto siempre existen riesgos que pueden hacer peligrar su éxito, así que es muy importante:

- Identificar los riesgos que pueden afectar al proyecto.
- Hacer un análisis cualitativo de los riesgos.
- Realizar un análisis cuantitativo de los riesgos.
- Planificar la respuesta a los riesgos.

En la Tabla 6 se muestran los principales riesgos identificados del proyecto y la tipificación del nivel de riesgo que tienen.

Riesgo	Causa	Consecuencias	Probabilidad	Impacto	Respuesta
Retrasos planificación	Viajes trabajo	Retrasos en la planificación	Media	Alto	Mitigar
Problemas investigación	Documentación APIs muy densa	Retrasos en la planificación	Alto	Alto	Mitigar
No se cumplen calendarios	Imprevistos	Reorganización planificación	Baja	Medio	Evitar
Problemas acceso documentación SDK iOS	IDE y SDK XCode iOS ahora es de pago por subscripción	Retraso planificación	Alta	Bajo	Mitigar
Oportunidad nuevas funcionalidades	Solicitud aplicación web	Descontrol proyecto	Alta	Alto	Compartir

Tabla 6: Riesgos del Proyecto

El significado de cada una de las respuestas a los posibles riesgos es:

- **Compartir**: asignar la oportunidad a un tercero.
- **Evitar**: cambiar el plan del proyecto de manera que se elimine la amenaza del riesgo.
- **Mitigar**: aplicar acciones preventivas o de contingencia para mitigar la probabilidad de que pase el impacto del riesgo. En el caso particular de este proyecto la medida de contingencia a aplicar con mayor probabilidad será el sobreesfuerzo del personal.

3. Análisis de requisitos

3.1. Requisitos no funcionales

3.1.1. Escenario de partida

El seguimiento a través del sistema ARGOS es uno de los más utilizados actualmente en proyectos y programas de investigación marina en todo el mundo (i.e. proyecto ARGO[16], el programa Global Drifter Program[17] dentro de la iniciativa internacional Global Ocean Observation System[18]).

El posicionamiento de una boya (o PTT[19] en la jerga ARGOS) seguida mediante el sistema ARGOS es administrado por la corporación internacional CLS Collective Satellite Localization: http://www.cls.fr con sede en Toulouse (Francia). Cada emisor tiene un identificador único que debe darse de alta en el organismo correspondiente (CLS en Europa). Los emisores (PTT) emiten constantemente una señal por radiofrecuencia que es recibida por los satélites cuando éstos pasan cerca de su posición. La señal se emite con una periodicidad de 90 segundos y una autonomía aproximada de un año y puede resultar amortiguada si encuentra obstáculos sólidos (barcos, fuerte oleaje), por lo que puede producirse fallos en la comunicación con el satélite. Una vez emitida la señal en radiofrecuencia es recibida por el sistema de adquisición de datos (DCS, *Data Collection System*) a bordo de los satélites NOAA. Para asegurar una razonable cobertura a escala global todos los satélites de la serie NOAA incorporan este sistema[20].

Los NOAA se mueven alrededor de la tierra en órbita heliosíncrona. Originalmente en el sistema ARGOS la boya puede ser posicionada por el satélite por efecto Doppler (fig. 3). Por tanto para calcular la posición de la boya, se necesita únicamente un satélite y cuatro señales: dos recibidas cuando el satélite se acerca a la boya y dos cuando se aleja; comparando el desplazamiento en el espacio de frecuencias de la señal debido al efecto Doppler y la posición del satélite se puede calcular la posición (fig. 3)

Según el modelo, la boya puede disponer adicionalmente de sensores de medida (temperatura, salinidad, etc.). En tal caso, los datos se codifican y se envían empaquetados en un telegrama hacia el satélite, y por tanto cuando el satélite detecta la boya calcula su posición por efecto Doppler y al mismo tiempo recibe la información relativa a las posiciones de la boya durante instantes anteriores a un ritmo prefijado, habitualmente

[16] ARGO: http://www.aoml.noaa.gov/phod/argo/index.php
[17] http://www.aoml.noaa.gov/phod/dac/gdp.html
[18] GOOS: http://www.aoml.noaa.gov/phod/goos.php
[19] ***P****latform* ***T****erminal* ***T****ransmitters*
[20] http://www.oso.noaa.gov/poesstatus/

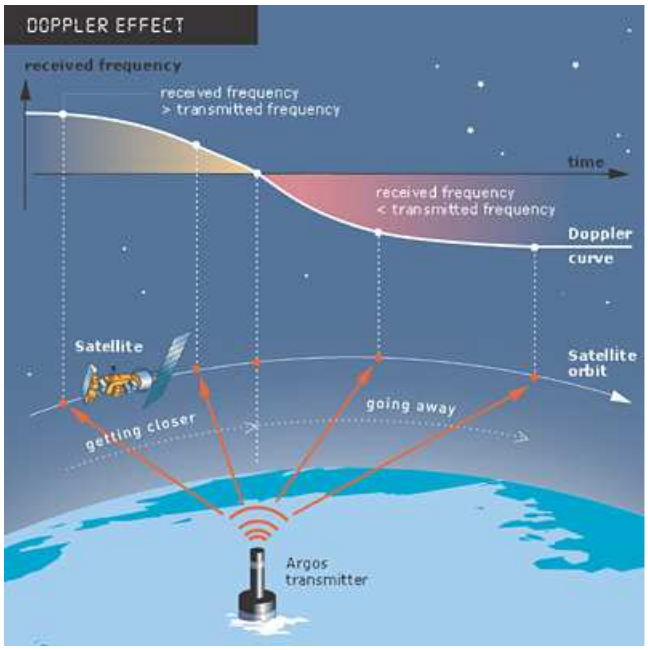

Figura 3: Sistema de posicionamiento por efecto Doppler (*Fuente: CLS-ARGOS*).

una hora.

Una vez que el satélite detecta la boya, guarda la información transmitida por ésta y la transmite a cualquiera de las estaciones receptoras terrestres que el sistema ARGOS tiene repartidas por todo el globo (fig. 4). Si no hay ninguna estación terrestre cercana

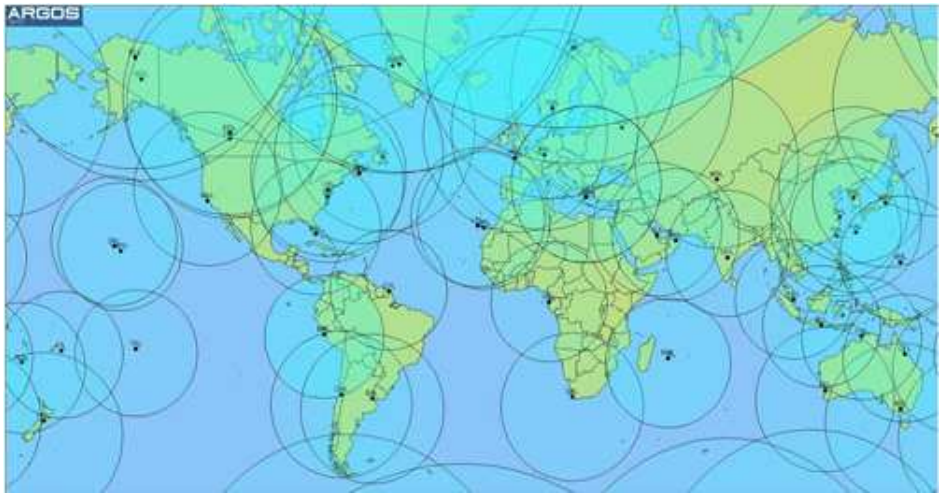

Figura 4: Distribución de las estaciones receptoras del sistema ARGOS.

al satélite en el momento de recibir los datos, éstos son descargados y transmitidos en la primera estación que encentra a lo largo de su trayectoria. De ahí que las posiciones de la boya pueden aparecer un cierto retraso (4 horas como promedio) en ser procesadas y puestas a disposición del usuario.

Posteriormente una vez descargados los datos en alguna estación de recepción, éstos se envían automáticamente a los centros de procesado (Francia y Estados Unidos) donde son

utilizados para recalcular las posiciones de los PTT detectadas por uno o varios satélites. Finalmente, se añaden a la base de datos y se publican en internet[21]. Actualmente y dada la proliferación de receptores GPS [22], la PTT puede disponer además, según el modelo, de un receptor de posicionamiento GPS que va registrando la posición de la boya a una frecuencia predeterminada. Esta información es almacenada y posteriormente se envia como parte del mismo paquete de información que recibe el satélite con el identificador y datos complementarios.

Como el sistema tiene una cierta componente aleatoria puede producirse un cierto desorden temporal en la compilación de los datos en función de la posición de la PTT, el satélite que la ha fijado y la estación receptora que ha recibido los datos. Por ello los centros de procesado envian los datos rutinariamente en el orden en que los reciben y a las 24 h de cada día, los datos se reordenan temporalmente. El sistema de posicionamiento mediante el sistema GPS es más preciso y puede obtenerse con una frecuencia muy alta comparado con el sistema de posicionamiento Doppler, debido esencialmente al número de plataformas utilizadas, y por tanto la cobertura, para fijar el receptor. En la tabla 7 se adjunta una breve comparativa de los dos sistema de posicionamiento de la PTT.

	Doppler[23]	GPS
Margen error	150 m	100 m
Intervalo entre registros	Irregular	Uno cada hora
Mensajes necesarios	Mínimo 4	Mínimo 1

Tabla 7: Sistema Doppler vs. posicionamiento con GPS

Cabe señalar que en el caso de disponer de un emisor con GPS se puede almacenar datos sobre la trayectoria de las boya de forma regular en el tiempo y por lo general con más precisión. Sin embargo el ritmo de actualización y disponibilidad de los datos es igual que en el caso del posicionamiento por efecto Doppler puesto que el refresco de datos depende de la conexión entre la trayectoria del satélite y de la boya. [24]

Las boyas con el sistema ARGOS transmiten una trama de 31 bytes codificados en hexadecimal para optimizar las transmisiones. Para codificar la medida de cada uno de los sensores son usados 3 bytes y 1 byte como checksum. Actualmente las boyas sólo recogen datos de conductividad y temperatura, aunque en el futuro se podrán conectar otros tipos de sensores, con lo cual el sistema ha de ser escalable en este sentido. En base a lo anterior sólo se transmiten 7 bytes útiles en cada trama, y su formato y significado se describe a continuación en la Tabla 8 en base a un ejemplo:

```
16        4A        20        35        53        48        50
```

[21] http://www.argos-system.org/html/system/faq_en.html
[22] ***G**lobal **P**ositioning **S**ystem: http://en.wikipedia.org/wiki/Global_Positioning_System*
[24] ver http://www.argos-system.org/manual/

Hexadecimal	Decimal	Descripción
16	22	Temperatura en grados centígrados: decenas y unidades
4A	74	Temperatura en grados centígrados: décimas y centésimas
20	32	Temp. en grados centígrados: milésimas y diezmilésimas
35	53	Conductividad en milisiemens por centímetro: decenas y unidades
53	83	Conductividad en milisiemens por centímetro: décimas y centésimas
48	72	Conductividad en milisiemens por centímetro: milésimas y diezmilésimas
50	80	Checksum: primer byte de la suma hexadecimal de temperatura y conductividad

Tabla 8: Tabla codificación transmisiones sistema ARGOS

La operativa que se sigue actualmente para recopilar los datos recogidos por las diferentes boyas es la siguiente:

1. Las boyas recogen y transmiten los datos al servicio de satélites contratado, actualmente CLS.
2. Los datos son almacenados en el servicio de tierra de las plataformas satelitales.
3. Los datos son descargados de forma manual por el investigador interesado en ellos de forma manual desde el portal web del servicio ARGOS, mostrado en la Figura 5, en el formato escogido por el usuario.

Figura 5: Página de entrada al website del sistema ARGOS.

El sistema ARGOS también ofrece otras vías para descargar los datos, en concreto el servicio CLS permite descargar los datos en formato RAW usando el protocolo Telnet,

que es el medio que se quiere utilizar para el sistema a construir. La descarga se realiza autenticándose mediante un usuario/contraseña en un servidor de datos del servicio CLS, y, una vez conectados hay diferentes comandos que retornan los datos formateados y filtrados de diferentes formas, según cada comando. Los datos descargados están en formato ASCII y contienen los datos transmitidos por las boyas y otros proporcionados por el servicio CLS, como pueden ser geoposicionamiento y satélites que captaron la transmisión de los datos.

El comando que se utilizará, "PRV/A,TX", mostrará:

1. Datos de localizaciones y sensores.
2. Datos de diagnóstico de transmisión.
3. Mensajes más signficativos por pasada de satélite.

Una muestra del mensaje ASCII descargado desde el servidor del servicio CLS sería:

```
073399  24.969N  43.601W   2            080/0122Z-080/0119
  ( 3)         16           4A           25           35
               53           3A           47

      005 msgs 003>-120dB  Best:  -117  Freq:  652682.7  IQ : 68
      Lat1: 24.969N  Lon1:  43.601W  Lat2: 17.263N  Lon2:  78.429W

073399  24.967N  43.620W   1            080/0319Z-080/0318
  ( 4)         16           4A           2A           35
               53           34           46

      005 msgs 000>-120dB  Best:  -126  Freq:  652690.8  IQ : 58
      Lat1: 24.967N  Lon1:  43.620W  Lat2: 15.674N  Lon2:   1.759W
```

El significado de cada uno de los campos de este mensaje, por ejemplo el último, se muestra en la Tabla 9:

Para más detalles se pueden consultar el manual del sistema ARGOS [25].

[25] ver: http://www.argos-system.org/manual/index.html#6-data/635_prv_command.htm

Cadena	Descripción
073399	Id boya
24.967N 43.620W	Posición estimada por servicio CLS
1	Clase de localización
080/0319Z	Día calendario juliano y hora:minuto UTC recolección datos
-080/0318	Día calendario juliano y hora:minuto UTC localización boya
(4)	Número de mensajes idénticos recibido durante pasada satélite
16 4A 2A	Temperatura en formato hexadecimal
35 53 34	Conductividad en formato hexadecimal
46	Checksum: Primer byte suma hexadecimal temperatura y conductividad
005 msgs	Número de mensajes recibidos por el satélite
000>-120dB	Número de mensajes recibidos con potencia de señal de más de 120db
Best: -126	Mejor potencia de señal en dB
Freq: 652690.8	Frecuencia de transmisión
IQ : 58	Indicador de calidad de transmisión
Lat1: 24.967N Lon1: 43.620W	Posición 1, sin documentar
Lat2: 15.674N Lon2: 1.759W	Posición 2, sin documentar

Tabla 9: Tabla significado tramas sistema ARGOS

3.1.2. Requisitos de interfaz

El sistema deberá contar con las siguientes interfaces:

- Interfaz móvil que permita visualizar los datos del sistema geoposicionados sobre un mapa. En concreto se debe ofrecer la funcionalidad de que el usuario escoja rangos de fechas, boyas o variables a visualizar.
- Interfaz que permite al usuario conectar directamente con el SGBD del sistema y consultar los datos del sistema desde la red interna del Institut Ciències del Mar, realizando consultas SQL.
- Interfaz web que permita visualizar los datos del sistema en un servicio de distribución de datos con interfaz web.

3.1.3. Requisitos de seguridad

El sistema central del Institut Ciències del Mar deberá identificar a los usuarios para comprobar, previamente, si:

- Están autorizados a acceder al sistema.
- Están autorizados a realizar las acciones previamente solicitadas.

La autenticación del usuario se realizará mediante la inserción de un usuario y contraseña y la posterior comprobación de la validez de las mismas por parte del sistema.

En cuanto a la aplicación móvil para visualización de datos esta no requerirá ninguna autenticación por parte del usuario y cualquier usuario que ejecute la aplicación tendrá acceso a los datos que solicite.

3.1.4. Requisitos de hardware

El sistema se debe ejecutar en las siguientes plataformas de hardware:

- Servidor del sistema: Servidor IBM BladeCenter HS22 - Type 7870
- Aplicación móvil: cualquier dispositivo móvil Apple que funcione con el sistema operativo iOS.

3.1.5. Requisitos de software

El sistema se debe ejecutar sobre los siguientes sistemas operativos:

- Aplicaciones en el servidor del sistema: Red Hat Enterprise Linux 5 64 bits o cualquier de sus clones, como pueden ser CentOS 5 64 bits o Scientific Linux 5 64 bits.
- Aplicación móvil: sistema operativo móvil iOS 4 o superior.

3.2. Requisitos funcionales

El sistema consta básicamente de dos partes claramente muy diferenciados, tal como se puede ver en la Figura 1, que son:

- Funcionalidades de las aplicaciones instaladas en el ICM Server.
- Funcionalidades de la aplicación móvil.

Paso a detallar a continuación los requisitos de forma diferenciadas para cada una ellas.

3.2.1. Funcionalidades de las aplicaciones en el Servidor ICM

Las funcionalidades que deben ofrecer las aplicaciones instaladas en el ICM Server serán muchas de las mencionadas en el apartado 1.3, en concreto:

- Descarga automática de los datos de las boyas desde el servicio CLS por medio del protocolo Telnet.
- Almacenamiento en sistema de ficheros de los datos de las boyas descargados en formato RAW.
- Transformación de los datos descargados a formatos NetCDF y KML.
- Servidor de datos vía web para visualización de los datos del sistema.
- Inserción automática de los datos descargados en el SGBD del sistema.

3.2.2. Funcionalidades de la aplicación móvil

Las funcionalidades de visualización que debe ofrecer la aplicación móvil serán:

- Visualización de los datos del sistema geolocalizados en un mapa mediante Puntos de interés (POI [26]) para cada boya activa, que se encuentre recogiendo datos oceanográficos. En cada POI se mostrarán los datos de geoposionamiento de ese punto y las medidas de salinidad y/o temperatura registradas por la boya en ese punto.

En cuanto a los criterios de búsqueda por parte del usuario, éste ha de poder:

- Escoger los datos a visualizar por identificador de boya.
- Poder visualizar toda la trayectoria de una boya en base a sus diferentes posiciones.
- Escoger los datos a visualizar por posición de la trayectoria de la boya.
- Poder visualizar los datos de cualquier punto de la trayectoria de cada boya.

[26] Point of interest: http://en.wikipedia.org/wiki/Point_of_interest

4. Diseño

4.1. Diseño arquitectónico

La arquitectura global del sistema será distribuida, de tres capas y de tres niveles, en concreto se tendrá:

1. **Nivel de presentación**: el encargado de mostrar los datos e interactuar con el usuario. Este nivel residirá en la aplicación móvil y el servidor web de datos situado en el ICM Server.
2. **Nivel de negocio**: se encargará de toda la lógica que modela los procesos de negocio del sistema y será el que hará todo el procesamiento necesario para atender las peticiones del usuario. Este nivel residirá en el ICM Server
3. **Nivel de administración de datos**: será el responsable de toda la persistencia del sistema. En este nivel estarán almacenados todos los datos del sistema y será el encargado de todas las operaciones de consulta, actualización y borrado de datos. Este nivel residirá en el SGBD del sistema.

4.1.1. Vista física

Aunque en la Figura 1 se muestra la vista física de la arquitectura global del sistema, a continuación en la Figura 6 se detalla más esta arquitectura.

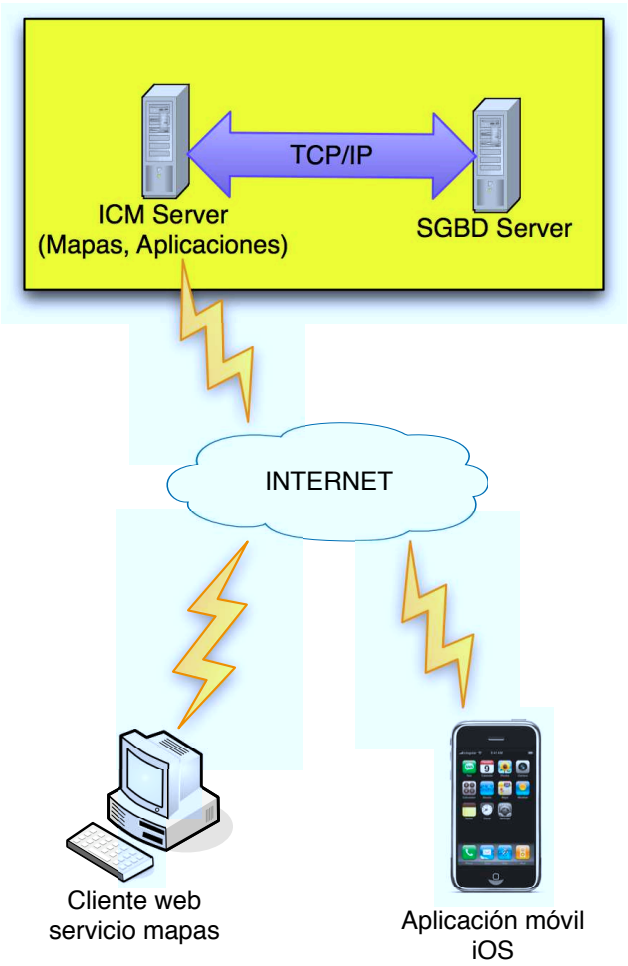

Figura 6: Vista física del sistema

Se puede ver que el sistema es distribuido y es fácilmente escalable. Tal como se ha dicho en el anterior apartado, cada nivel de la arquitectura reside en diferentes nodos, con excepción del nivel de presentación que reside en dos nodos diferentes, aplicación móvil y servidor de datos vía web en el ICM Server.

4.1.2. Vista de despliegue

A continuación, en la Figura 7 se presenta el diagrama de despliegue del sistema a desarrollar:

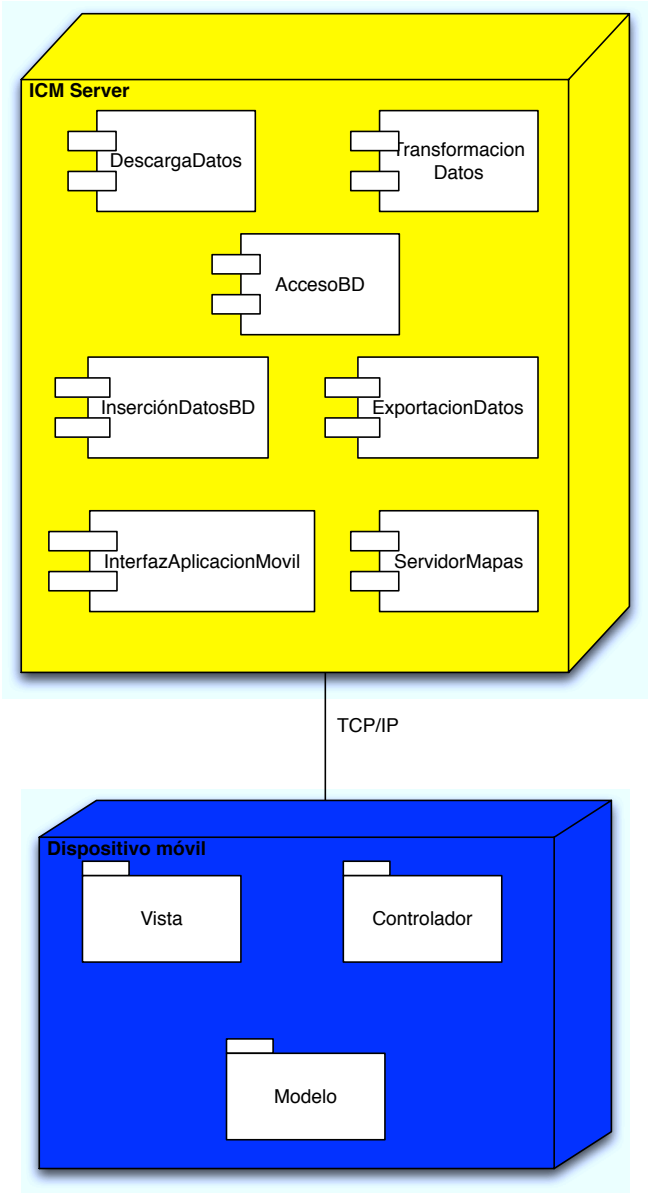

Figura 7: Diagrama despliegue del sistema

En este diagrama se pueden distinguir los diferentes componentes y nodos del sistema. A continuación se procede a la descripción de cada uno de ellos

Nodo ICM Server:

- DescargaDatos: componente que se encargará de la descarga de los datos RAW desde el servidor de CLS.
- InsercionDatosBD: componente que insertará los datos en el SGBD del sistema.
- TransformacionDatos: componente que transformará los datos a los distintos formatos requeridos antes de su inserción en el SGBD o de su exportación a NetCDF y KML.

- AccesoBD: componente responsable del acceso al SGBD del sistema.
- InterfazAplicacionMovil: componente que permitirá acceder a los datos del sistema a los clientes que quieran interactuar con el sistema a través de la aplicación móvil.
- ServidorMapas: componente que gestionará el servidor de datos vía web.

Dispositivo móvil: En el dispositivo móvil se usará el patrón de arquitectura de software MVC [27], tal como recomienda Apple en su documentación sobre desarrollo con iOS.

[27]Model View Controller: http://en.wikipedia.org/wiki/Model–view–controller

4.2. Diseño de clases

A continuación, en la Figura 8, se muestra el diseño estático de clases de las aplicaciones a desarrollar en el ICM Server, en el que se recogen las entidades utilizadas por el sistema:

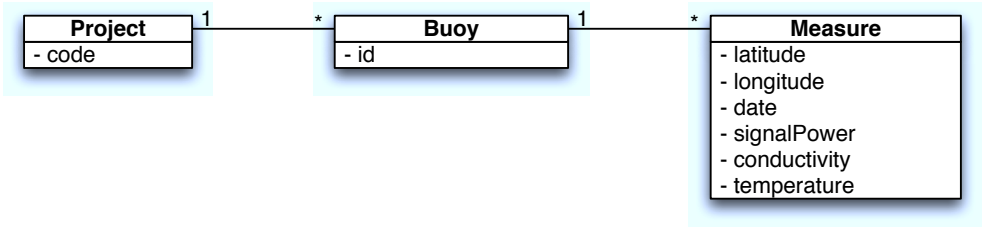

Figura 8: Diagrama estático de diseño ICM Server

Se puede ver que queda un diagrama relativamente sencillo, pero acorde con los datos recogidos por las boyas.

Añadir que se podría crear una subclase para los tipos de medidas, lo cual haría el sistema, en teoría, más fácilmente escalable, pero se ha optado por no hacerlo debido a que no está planteado añadir más tipos de sensores a corto o medio plazo, y los cambios de añadir más tipos de sensores obligarían a cambiar las operaciones de transformación de datos, lo cual es más complejo que cambiar el diseño estático.

4.3. Diseño de la persistencia

En el lado servidor será necesario guardar todos los datos relativos a las boyas y sus medidas. El esquema del diseño de la base de datos quedaría tal como se muestra en el siguiente diagrama:

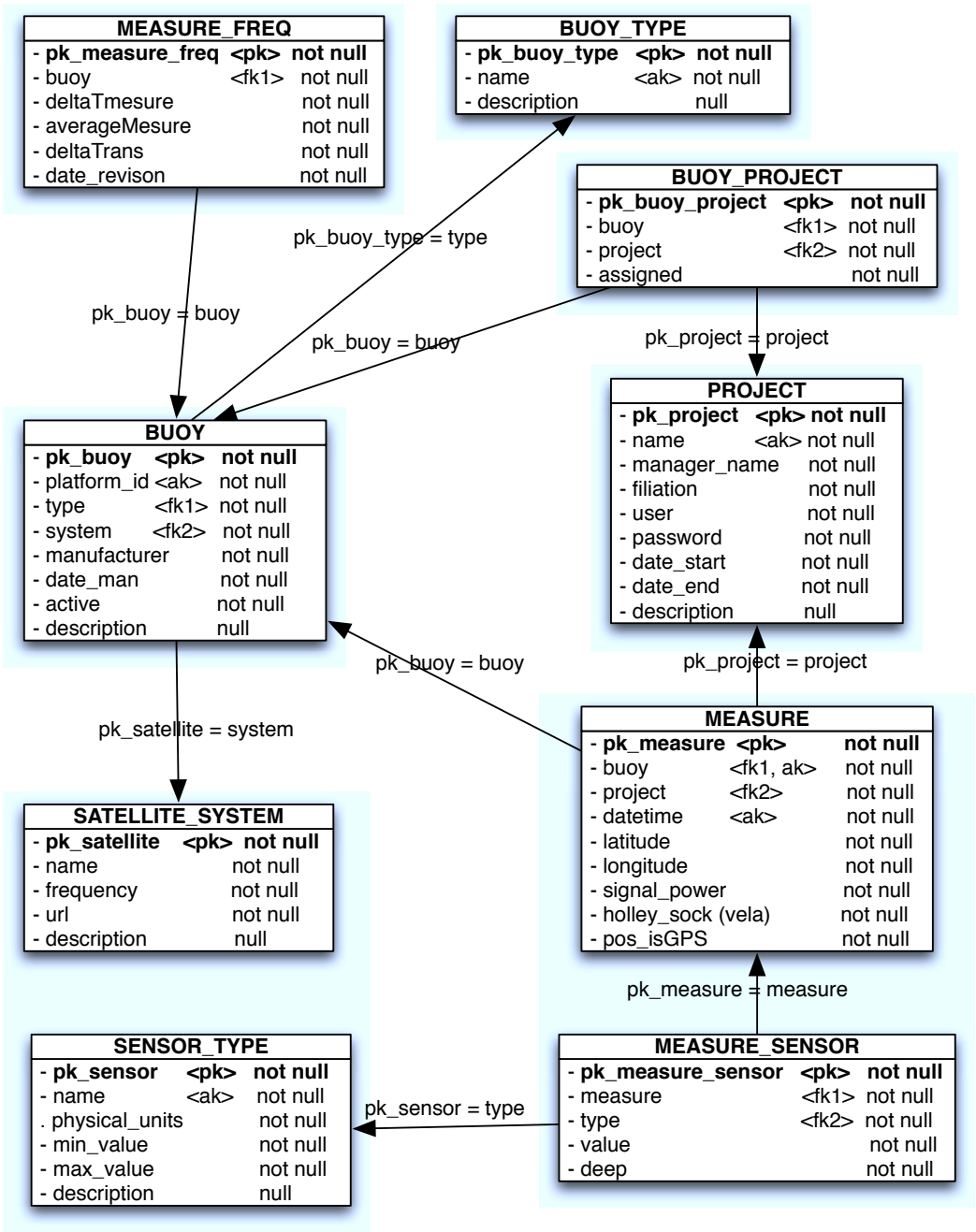

Figura 9: Diseño de la base de datos central del sistema

4.3.1. Detalle de las tablas

- **BUOY**: Datos de identificación de cada boya.
- **BUOY_TYPE**: Tipos de boya.
- **BUOY_PROJECT**: Tipos de boya.
- **PROJECT**: Datos de cada proyecto de ARGOS al que pertenezca cada boya. Comentar al respecto que de momento sólo hay un proyecto que se usa para diferentes proyectos de investigación o campañas.

- **SATELLITE_SYSTEM**: Tipo de sistema de satélites de recepción de datos.
- **MEASURE**: Datos comunes a todos las medidas relativos al sistema de posicionamiento de cada boya.
- **MEASURE_SENSOR**: Datos de los diferentes sensores para cada medida.
- **MEASURE_FREQ**: Información sobre la frecuencia de muestreo de cada boya.
- **SENSOR_TYPE**: Tipos de sensor.

También será necesario almacenar datos en el dispositivo móvil para minimizar el consumo de ancho de banda. Básicamente lo que se hará es almacenar en el dispositivo los datos relativos a las medidas tomadas por las boyas, su posición y la fecha en la que fueron tomadas, que es lo que se podrá visualizar con la aplicación móvil. Dado que el volumen de datos no será muy grande se opta por almacenar los datos de las boyas un archivo Property List [28], que tendrá la estructura que se puede ver en la Figura 10:

Key	Type	Value
▼Item 0	Diction...	(2 items)
code	String	73339
▼Measures	Array	(2 items)
▼Item 0	Diction...	(6 items)
Conductivity	String	258
Date	String	21-01-2010 00:06
Latitude	Number	25
Longitude	Number	25
SignalPower	String	125
Temperature	String	22.875
▼Item 1	Diction...	(6 items)
Conductivity	String	245
Date	String	21-01-2010 00:06
Latitude	Number	24
Longitude	Number	21
SignalPower	String	121
Temperature	String	21.254
▼Item 1	Diction...	(2 items)
code	String	73358
▼Measures	Array	(2 items)
▼Item 0	Diction...	(6 items)
Conductivity	String	258
Date	String	21-01-2010 00:06
Latitude	Number	25
Longitude	Number	25
SignalPower	String	125
Temperature	String	22.875
▼Item 1	Diction...	(6 items)
Conductivity	String	245
Date	String	21-01-2010 00:06
Latitude	Number	24
Longitude	Number	21
SignalPower	String	121
Temperature	String	21.254

Figura 10: Diseño de esquema fichero Property List

Como se puede ver la estructura consta de un array que contiene un elemento *dictionary* para cada boya, y dentro de éste hay un atributo con el identificador de la boya y un array, Measures, que contiene las diferentes medidas para cada una de ellas en un *dictionary*.

[28] Property Lists

Se opta por usar esta estructura, fichero Property List, y no una base de datos de las que se pueden incorporar en iOS, por su sencillez, por la facilidad que ofrece Objective-C para su tratamiento y porque el volumen de datos a tratar es muy pequeño.

4.4. Tecnologías a usar

A continuación se muestran las tecnologías que se usarán para el sistema:

- **SGBD**: Se usará PostgreSQL, ya que es el SGBD del que se dispone en el Institut de Ciències del Mar.
- **Lenguaje de desarrollo aplicaciones ICM Server**: Se usará Java por ser orientado a objetos y multiplaforma.
- **Acceso a SGBD**: Se usarán las interfícies proporcionadas por el lenguaje Java y el JDBC Driver proporcionado por PostgreSQL.
- **Comunicaciones Aplicación móvil - ICM Server**: se usará el acceso por medio de servicios web a un fichero XML en formato PLIST [29] con los datos a descargar por la aplicación móvil. Esta decisión se espera que disminuya el tiempo de desarrollo de la aplicación móvil, ya que las aplicaciones móviles para iOS reconocen nativamente el formato PLIST, aunque, por contra, será necesario transformar los datos a este formato para que estén accesibles para la aplicación móvil en una URI [30] en el ICM Server.
- **Aplicación móvil**: Se desarrollará para los sistemas operativos iOS usando el lenguaje y las APIs de desarrollo recomendadas por su fabricante, Apple Inc.
- **Servidor de mapas**: Se utilizará el servicio de distribución datos THREDDS [31] instalado bajo un servidor de aplicaciones Tomcat en el ICM Server. También se podrán visualizar los datos en formato KML usando Google Maps.

[29] Property Lists
[30] Uniform Resource Identifier http://en.wikipedia.org/wiki/Uniform_Resource_Identifier
[31] THREDDS: http://www.unidata.ucar.edu/projects/THREDDS

4.5. Diseño aplicación móvil

La aplicación móvil se compondrá básicamente de cuatro secciones diferenciadas de las que se muestra a continuación un prototipo de pantalla y una pequeña descripción. En las siguientes secciones se muestra un prototipo de como serán las pantallas del aplicación.

4.5.1. Sección Select Buoy

En este pantalla el usuario podrá:

- Seleccionar la boya de la que quiere visualizar los datos.
- Ver información relativa a la aplicación pulsando sobre el botón superior derecho de información.

El prototipo de pantalla se muestra a continuación en la Figura 11

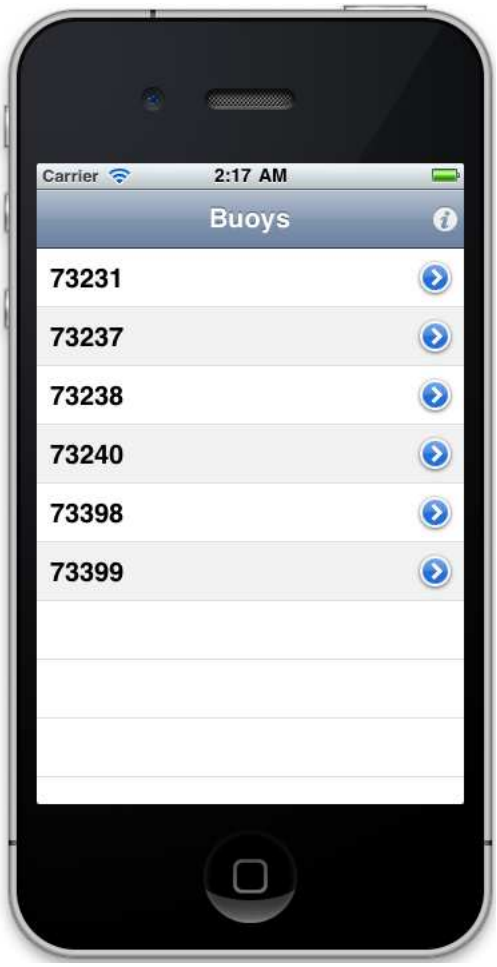

Figura 11: Sección Select Buoys

4.5.2. Sección View Map

En esta sección se verán las posiciones de las boyas geoposicionadas sobre un mapa, con todo su histórico de posiciones y datos. El usuario podrá:

- Pulsar sobre un punto y ver los datos recogidos en él en la siguiente sección de la aplicación.
- Escoger el tipo de mapa, satélite o normal.
- Ver su localización en el mapa mientras ve la trayectoria de una boya.

A continuación, en la Figura 12 se muestra el prototipo de las diferentes pantallas de esta sección de la aplicación.

(a) Pantalla Mapa Satélite **(b)** Pantalla Mapa normal

Figura 12: Sección View Map

4.5.3. Sección Measure Detail

En esta sección se muestran la pantalla que permitirá ver todos los datos del punto seleccionado por el usuario en el mapa. La pantalla de esta sección se puede ver en la Figura 13:

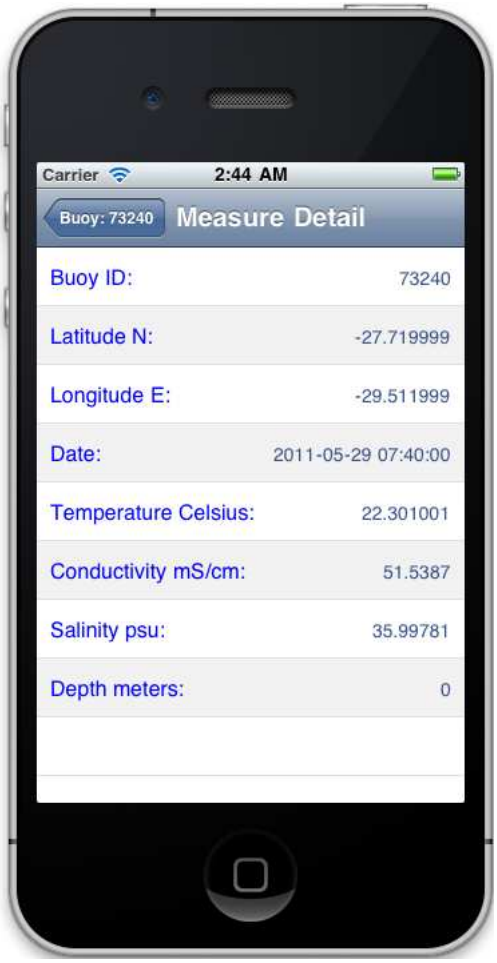

Figura 13: Sección Measure Detail

4.5.4. Sección About

En esta sección se mostrará información sobre el desarrollador y el Institut de Ciències del Mar.

Se podrá acceder a esta información desde la página principal de la aplicación, a continuación se muestra como será esta pantalla en la Figura 14:

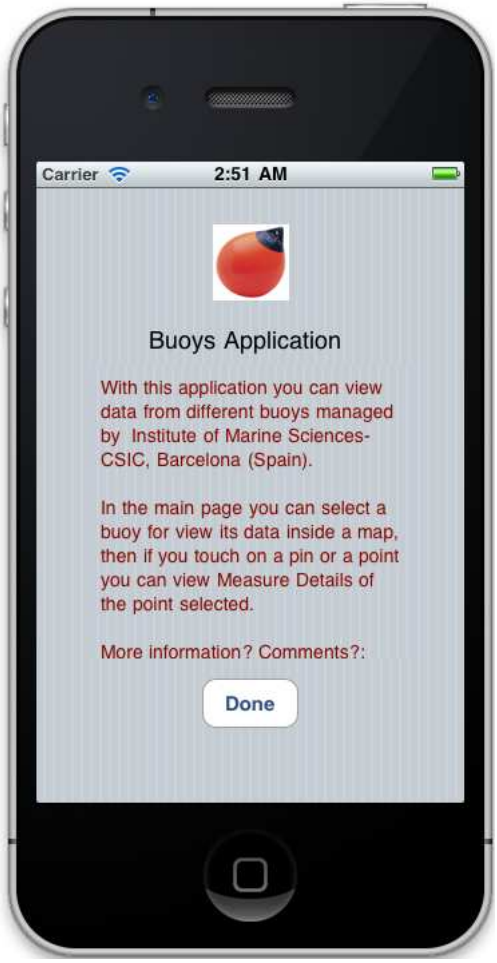

Figura 14: Sección About

5. Implementación

En los siguientes apartados se explican todos los detalles de la implementación para cada uno de los componentes del sistema.

5.1. Persistencia

Para este punto simplemente se crea la base de datos en el servidor mencionado en la descripción del proyecto 1.2 siguiendo el esquema 9 mencionado en el apartado de Diseño.

Comentar a este respecto que se ha procedido a realizar cambios menores en el esquema de la base de datos debido a nuevas exigencias de nuestro cliente, el ICM, cambios que han consistido en añadir una nueva tabla para guardar el histórico de asignaciones de boyas a proyectos, y que han sido añadidos al esquema anteriormente mencionado.

5.2. Servidor de aplicaciones ICM Server

Previamente se ha procedido a crear una nueva máquina virtual en el servidor disponible, mencionado previamente en los requisitos 3.1.4, que cuenta con las siguientes características a nivel de software:

1. Sistema operativo Scientific Linux [32] versión 5.4 de 64 bits.
2. Kernel con hypervisor Xen [33] versión 3, tal como se ha mencionado también en la sección de Descripción del presente proyecto 1.2.
3. Virtualización completa, ya que a nivel de hardware cuenta con dos procesadores Intel que la soportan mediante la tecnología *IVT*[34].

La imagen de máquina virtual instalada y configurada cuenta, entre otras, con las siguientes características a nivel de configuración:

1. Sistema operativo CentOS [35] 5.5
2. Maquina virtual Java [36] Versión 6 Update 23

[32] Scientific Linux http://www.scientificlinux.org
[33] Xen: http://www.xen.org
[34] Intel Virtualization Technology
[35] CentOS: http://www.centos.org
[36] Java: http://www.java.com

3. Servidor web Apache [37] Versión 2.0.3

4. Servidor de aplicaciones Tomcat [38] Versión 6.0.32

5. Servicio de distribución datos THREDDS [39]

5.2.1. THREDDS

La instalacción de este componente requiere seguir las instrucciones que se pueden encontrar en la página del proyecto, que son, básicamente, descargarse el Web Application Archive (WAR) y desplegarlo bajo Tomcat

Seguidamente es necesario configurarlo creando un nuevo catálogo en la configuración del servicio. Para poder ver los datos relativos a las boyas creados por nuestro sistema creamos un catálogo para estos datos:

[37] Apache: http://httpd.apache.org/
[38] Tomcat: http://tomcat.apache.org/
[39] THREDDS: http://www.unidata.ucar.edu/projects/THREDDS

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="http://www.unidata.ucar.edu/namespaces/thredds/InvCatalog/v1.0" ↩
    xmlns:xlink="http://www.w3.org/1999/xlink"
   name="Unidata THREDDS-IDD NetCDF-OpenDAP Server" version="1.0.1">

  <service name="latest" serviceType="Resolver" base="" />
  <service name="allServices" base="" serviceType="Compound">
    <service name="ncdods" serviceType="OpenDAP" base="/thredds/dodsC/" target="_blank"↩
        />
    <service name="httpserver" serviceType="HTTPServer" base="/thredds/fileServer/" />
    <!--service name="wcs" serviceType="WCS" base="/thredds/wcs/" /-->
    <!--service name="wms" serviceType="WMS" base="/thredds/wms/" /-->
    <!-- <service name="ncss" serviceType="NetcdfSubset" base="/thredds/ncss/grid/" /--↩
       >
  </service>
  <service name="subServices" base="" serviceType="Compound">
    <service name="ncdods" serviceType="OpenDAP" base="/thredds/dodsC/" target="_blank"↩
        />
    <service name="ncss" serviceType="NetcdfSubset" base="/thredds/ncss/grid/" />
  </service>


  <dataset name="Buoys ARGO data PFC Fernando Perez Lopez" ID="buoys">
    <metadata inherited="true">
      <serviceName>allServices</serviceName>
      <dataFormat>NetCDF</dataFormat>
        </metadata>


    <datasetScan name="Individual Files" ID="buoys"
                 path="buoys" location="/usr/local/apache-tomcat-6.0.32/content/thredds↩
                     /public/buoys/">

      <filter>
        <include wildcard="*.nc" />
      </filter>
<netcdf xmlns="http://www.unidata.ucar.edu/namespaces/netcdf/ncml-2.2" enhance="true" /↩
    >
      <addID/>
      <sort>
        <lexigraphicByName increasing="false"/>
      </sort>
      <addLatest/>
      <addDatasetSize/>
    </datasetScan>


</dataset>
</catalog>
```

Ahora simplemente al introducir los ficheros netCDF en el directorio que se incluye en la configuración, el sistema los leerá automáticamente y podrán ser consultados por los usuarios:

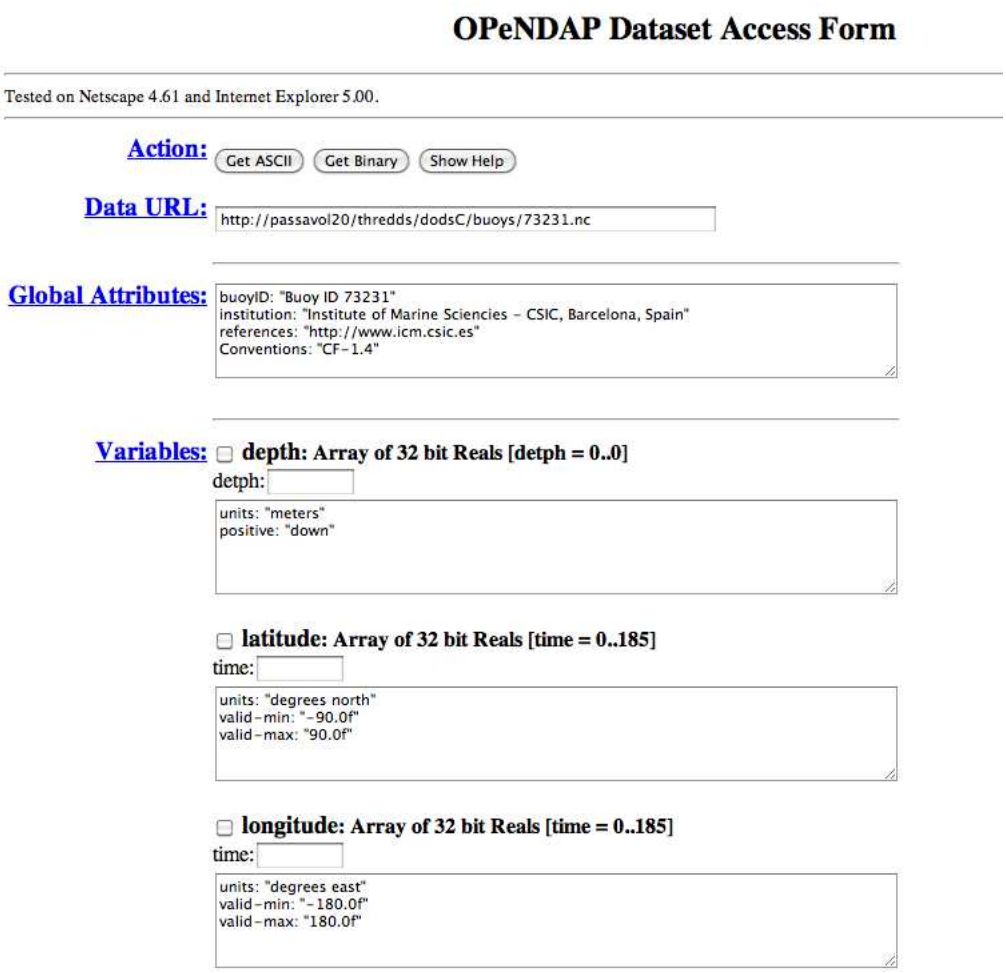

Figura 15: Consulta de datos en THREDDS

5.2.2. ncWMS

ncWMS permite visualizar datos en formato netCDF que tengan como dimensiones sus coordenadas de posición, esto es, latitud y longitud.

Tal como se explica posteriormente en 5.3.5 no se instalará este componente, ya que los datos procedentes de las boyas no tienen como dimensiones las coordenadas de latitud y longitud.

Esto no impedirá representar los datos de forma geoposicionada sobre un mapa, ya que se exportarán sus datos y sus trayectorias al formato KML, como veremos en apartados posteriores 5.3.5, lo cual permitirá representar los datos de forma geoposicionada.

5.3. Aplicación Buoys ICM Server

La aplicación, y todos sus componentes, se ha desarrollado con lenguaje Java, tal como se ha mencionado en secciones anteriores, usando el IDE Eclipse. En concreto se ha utilizado el *Java Development Kit 6* junto con las siguientes librerías/APIs:

1. **Apache Commons Net**[40]: proporciona un fácil acceso a la descarga de datos a través de diferentes protocolos, entre ellos Telnet que es el que tendrá que usar la aplicación del lado servidor para la descarga de datos.

2. **NetCDF Java**[41]: proporciona las interfaces necesarias para leer y escribir ficheros en formato NetCDF y permitirá la exportación de datos a formato este formato usando el lenguaje Java.

3. **Java property list library**[42] : permite la fácil lectura y escritura de ficheros en formato Apple Property List. En este proyecto permitirá la exportación de datos a éste formato.

4. **Java API for KML**[43]: librería que proporciona interfaces para el fácil acceso a ficheros KML. En este proyecto permitirá la exportación de datos a este formato.

5. **JAXB Reference Implementation**[44]: esta librería proporciona una implementación de la *Java Architecture for XML Binding*, y es un requisito para la librería anterior que permite el tratamiento de ficheros KML, debido a que estos ficheros no dejan de ser ficheros XML.

6. **PostgreSQL JDBC Driver**[45]: permite la conexión con bases de datos PostgreSQL usando el lenguaje Java. En este proyecto permitirá la consulta e inserción de datos en la base de datos del sistema.

La aplicación consta de los componentes mencionados en el Análisis de requisitos 7.

A continuación, en las figuras 16 y 17 se muestran los diagramas de las clases que se han implementado:

[40] Apache Commons Net: http://commons.apache.org/net
[41] NetCDF Java: http://www.unidata.ucar.edu/software/netcdf-java/
[42] dd-plist: http://code.google.com/p/plist
[43] Java API for KML: http://code.google.com/p/javaapiforkml/
[44] JAXB Reference Implementation: http://jaxb.java.net
[45] PostgreSQL JDBC Driver : http://jdbc.postgresql.org

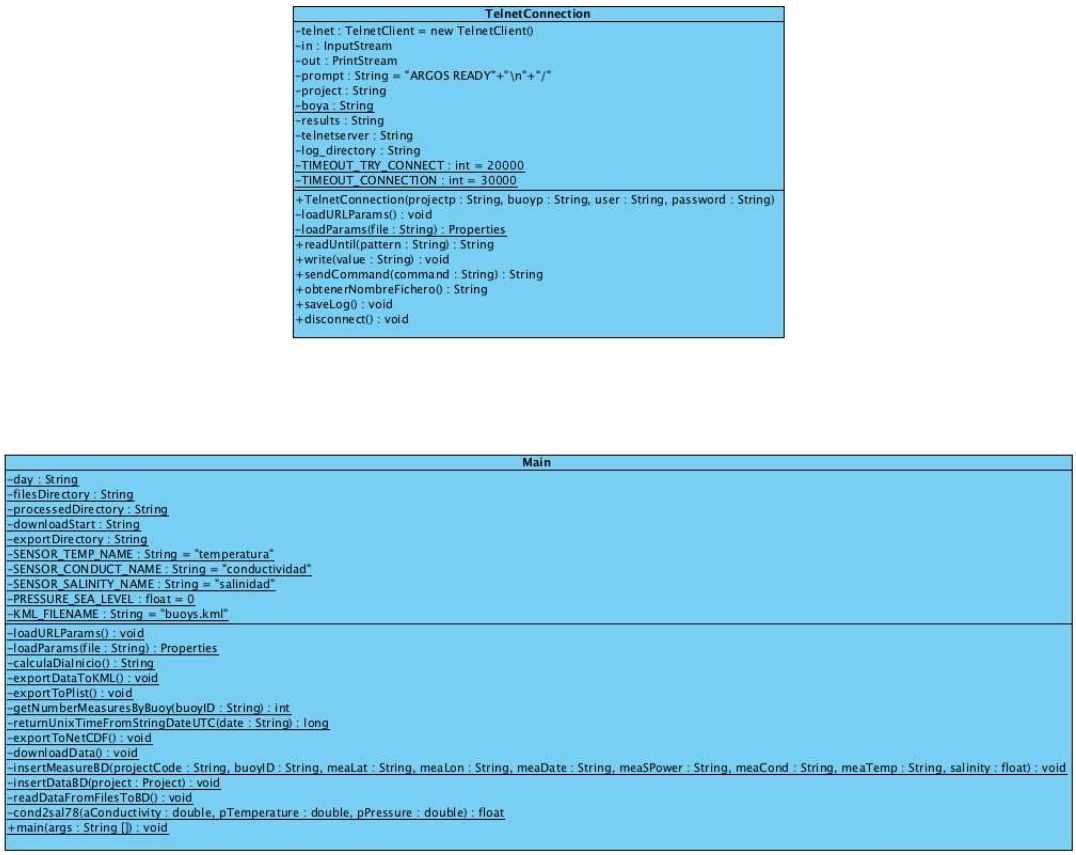

Figura 16: Diagrama de clases 1 Buoys ICM Server

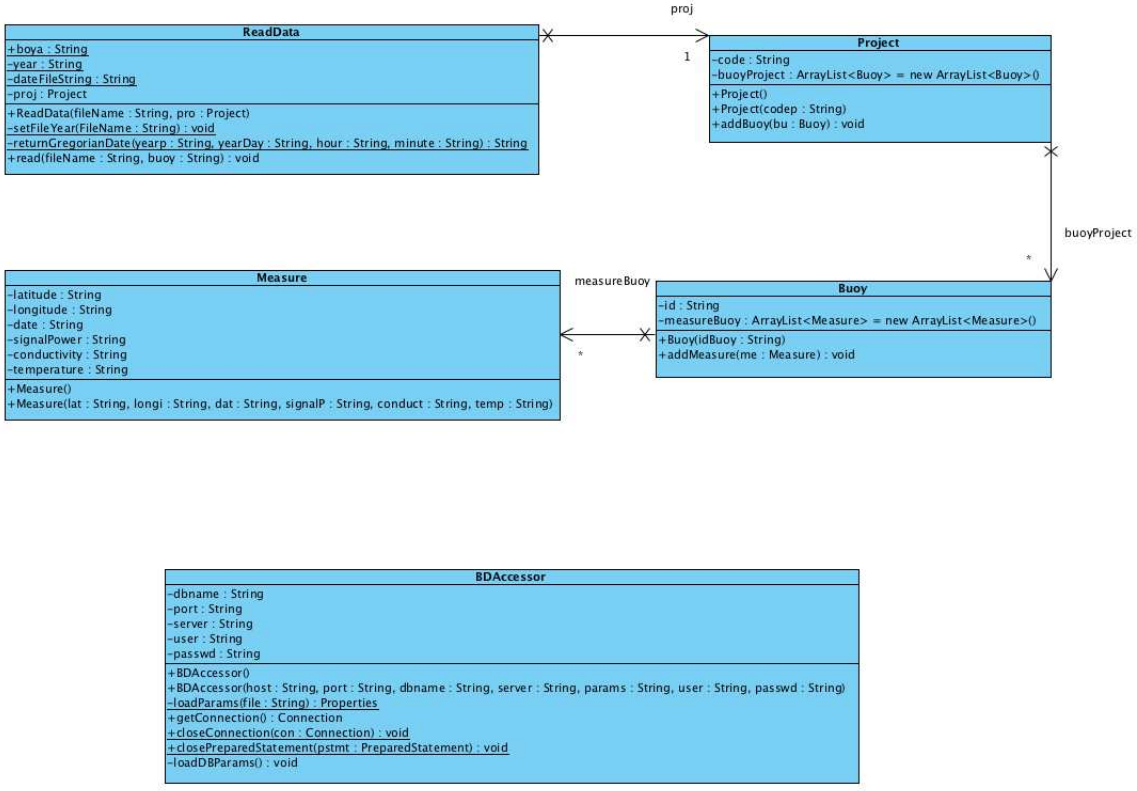

Figura 17: Diagrama de clases 2 Buoys ICM Server

En la Figura 18 se muestra cual será la iteracción entre todas las clases que forman los componentes mencionados anteriormente:

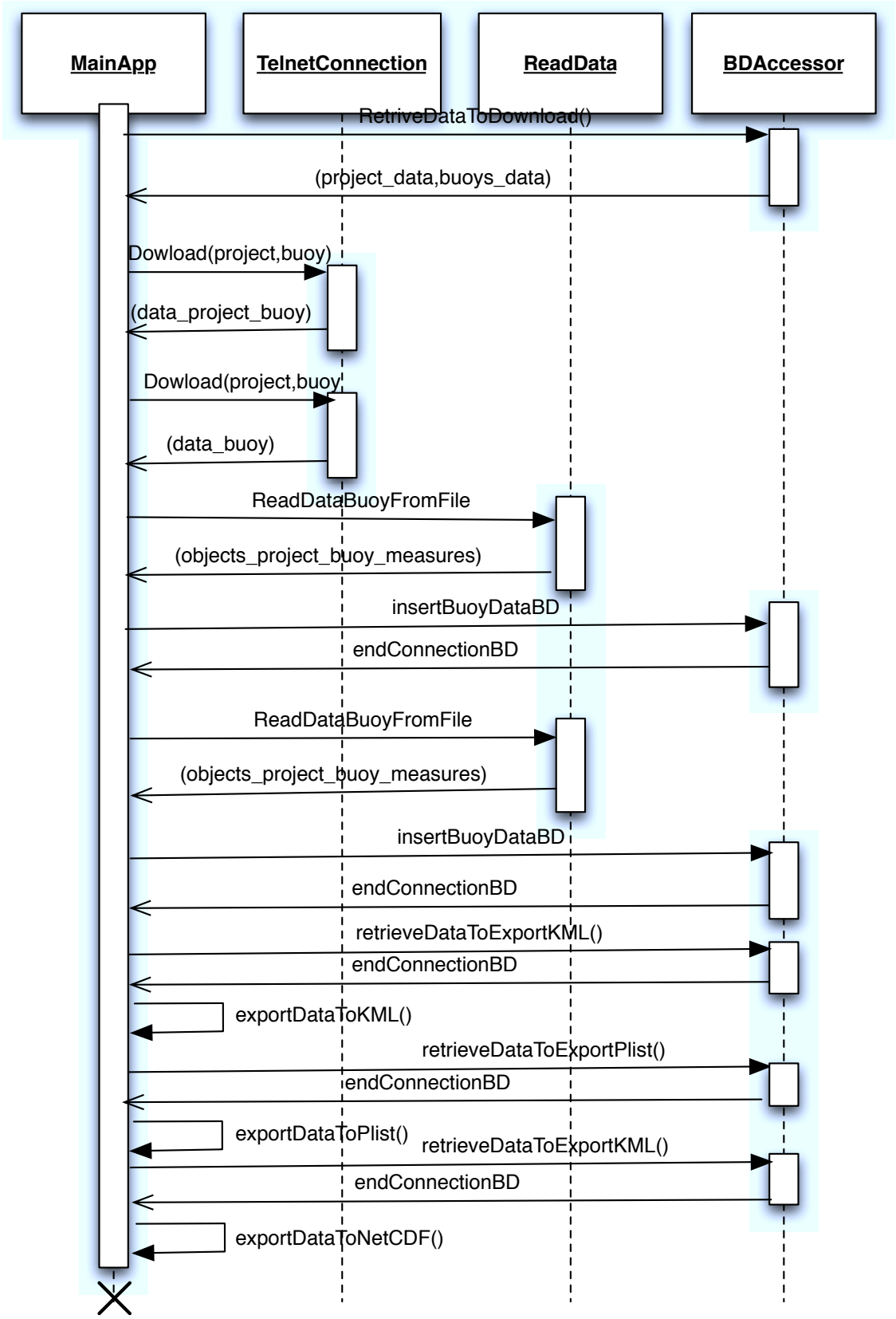

Figura 18: Diagrama de secuencia Buoys ICM Server

Seguidamente a continuación se procede a mencionar los aspectos más relevantes para cada uno de ellos:

5.3.1. Descarga de Datos

Para este componente se ha utilizado la librería *Apache Commons Net* anteriormente mencionada. Esta librería proporciona el acceso por Telnet de forma muy sencilla y transparente, en concreto la conexión se realiza sólo con una pequeña porción de código en la que se realiza la conexión, se fijan parámetros y se configura la entrada y salida de datos en la sesión Telnet:

```
TelnetClient telnet = new TelnetClient();

telnet.setConnectTimeout(TIMEOUT_TRY_CONNECT); //timeout intento conexion en ↩
    milisegundos


telnet.connect(telnetserver,23);
telnet.setSoTimeout(TIMEOUT_CONNECTION);  //timeout conexion


in = telnet.getInputStream();
out = new PrintStream( telnet.getOutputStream());
```

La implementación de este componente se ha hecho en una clase, **TelnetConnection**, que después es llamada desde la clase **Main**, para que este componente sea modular y reutilizable.

Se ha implementado de forma que se realiza una conexión para cada boya activa, y se descarga un fichero en el formato RAW previamente mencionado en secciones anteriores.

Añadir que, para que el programa principal no se pare si hay un fallo en la conexión Telnet, se captura la excepción *java.net.ConnectException*. Así de esta forma si hay ficheros pendientes de procesar por la interrupción de una ejecución previa del sistema, estos se procesarán aunque falle esta conexión.

5.3.2. Transformación de Datos

Este componente, una vez descargados todos los ficheros para las distintas boyas los lee e interpreta para crear los objetos mencionados en el diseño estático de clases en la figura 8. Para ello se ha creado una clase **ReadData** que actúa de parser y va leyendo cada trama de datos de cada uno de los ficheros descargados previamente.

Esta lectura se hará para cada fichero, y se irán creando los objetos correspondientes a los datos leídos e interpretados.

Para escanear el fichero se ha usado la clase *Scanner* de Java, junto con expresiones regulares para reconocer las distintas expresiones:

```
Scanner scanner = new Scanner(new FileReader(file));


String word = null;

try {

    Buoy boyaOb = new Buoy(boya);
    while (scanner.hasNextLine()) {



       word = scanner.next();

       if (word.matches("Prog")) {

            String projName = scanner.next();

            projName = Integer.toString(Integer.parseInt(projName));
            proj.setCode(projName);


            proj.addBuoy(boyaOb);


       }

       if (word.matches("[0]*"+boya)) {

            System.out.println("Boya: "+word);

            //leemos latitud
            String latitude = scanner.next();

            System.out.println("Latitud original "+latitude);
            //Transformamos latitud a formato con signo

            if (latitude.endsWith("N")) {

                latitude = latitude.substring(0, latitude.length()-1);

            }

            if (latitude.endsWith("S")){

                latitude = "-"+latitude.substring(0, latitude.length()-1);

            }
```

5.3.3. Inserción en Base de Datos

Una vez creados los objetos para cada boya, se procede a su inserción. Para ello se realiza una conexión a la base de datos y realizan inserciones para cada tipo de medida:

```
// Conexion BD para insertar datos

            BDAccessor bdIns = new BDAccessor();
            Connection bdconnectionIns = bdIns.getConnection();
            Statement stmtIns = bdconnectionIns.createStatement();

            String insertMeasure = null;
            String insertMeasureSensorTemp = null;
            String insertMeasureSensorCond = null;
            String insertSalinity = null;

            //Preparamos las operaciones de insercion

            insertMeasure = "INSERT INTO measure (datetime,latitude,longitude,↩
                signal_power,buoy,project) " +
                    "(SELECT '"+ meaDate +"','"+ meaLat +"','"+ meaLon+"','"+ meaSPower↩
                        +"'," +
                    "pk_buoy,pk_project FROM buoy,project WHERE " +
                    "platform_id='"+ buoyID + "' AND name='"+ projectCode +"');";

            System.out.println( "Insercion medida : \n"+ insertMeasure);

            insertMeasureSensorTemp = "INSERT INTO measure_sensor (value,measure,type) ↩
                " +

                      "(SELECT '"+ meaTemp +"',pk_measure,pk_sensor "+
                      "FROM measure,sensor_type WHERE "+
                      "datetime='"+ meaDate +"' AND name='"+  SENSOR_TEMP_NAME+"' "+
                      "AND buoy=(select pk_buoy "+
                      "FROM buoy where platform_id='"+ buoyID  +"'));";

            System.out.println( "Insercion temperatura : \n"+ insertMeasureSensorTemp);
```

A su vez, como petición después del análisis de requisitos, se ha implementado el cálculo de la salinidad antes de la inserción de datos, con lo cual nuestro sistema contendrá los datos de salinidad absoluta que dependen de la temperatura, la conductividad y la presión, usando el algoritmo mencionado en UNESCO Technical Papers in marine Science, 1983.

Un detalle de la implementación sería el siguiente:

```
private static float cond2sal78 (double aConductivity,double pTemperature,double ↩
    pPressure) {

    double xr = 0;

    double rt35 = 0;
    double c = 0;

    double b = 0;
    double a = 0;
    double dt = 0;
    double rt = 0;
    double aSalinity = 0;
    double salinity = 0;

    try {

    dt = pTemperature - 15;
    aSalinity = aConductivity/42.914;

     // FUNCTION RT35: C(35,T,0)/C(35,15,0) VARIATION WITH TEMPERATURE

    rt35 = (((1.0031E-9 * pTemperature - 6.9698E-7) * pTemperature + 1.104259E-4) *↩
        pTemperature
              + 2.00564E-2) * pTemperature + 0.6766097;


// C(XP) POLYNOMIAL CORRESPONDS TO A1-A3 CONSTANTS: LEWIS 1980

    c = ((3.989E-15 * pPressure - 6.370E-10) * pPressure + 2.070E-5) * pPressure;


    b = (4.464E-4 * pTemperature + 3.426E-2) * pTemperature + 1.0;

    //A(XT) POLYNOMIAL CORRESPONDS TO B3 AND B4 CONSTANTS: LEWIS 1980
      a = -3.107E-3 * pTemperature + 0.4215;

      rt        = aSalinity / (rt35 * (1.0 + c /(b+ a * aSalinity)));
      rt        = Math.sqrt(Math.abs(rt));


      xr = rt;

    //Ecuacion
      salinity = ((((2.7081*xr-7.0261)*xr+14.0941)*xr+25.3851)*xr
              - 0.1692)*xr+0.0080
              + (dt/(1.0+0.0162*dt))*(((((-0.0144*xr
              + 0.0636)*xr-0.0375)*xr-0.0066)*xr-0.0056)*xr+0.0005);

      // control, whether result
```

5.3.4. Acceso Base de Datos

Para este componente se ha implementado una clase auxiliar, **BDAccessor**, en la que están encapsuladas todas las operaciones de conexión con la Base de Datos por medio del driver JDBC.

De esta forma será más fácil realizar la conexión llamando a esta clase.

En esta clase se han implementado todas las operaciones relativas a la conexión a la base de datos usando el driver JDBC de PostgreSQL:

```
public Connection getConnection()
    throws SQLException, IOException, ClassNotFoundException {
    Connection con = null;

    try {

        // Lee el fichero de configuracion de la BD
        this.loadDBParams();


        // Carga el driver
        Class.forName( "org.postgresql.Driver" );

        // Conexion
        String url = "jdbc:postgresql:"+
        "//" + server + ":" + port + "/" + dbname;
         con = DriverManager.getConnection( url, user, passwd );


    } catch (IOException e) {
        System.err.println( "ERROR: No se ha podido leer el fichero de parametros" ↩
            );
        throw e;
    } catch (ClassNotFoundException e1) {
        System.err.println( "ERROR: No se ha podido cargar driver JDBC" );
        throw e1;
    } catch ( SQLException e2 ) {
        System.err.println( "ERROR: No se ha podido conectar con la Base de Datos" ↩
            );
        throw e2;
    }

    // Activamos autocommit para las inserciones
    con.setAutoCommit( true );

    return con;
}

/**
 * Cierra la conexion a la BD.
 * @param con Conexion a la BD.
 */
public static void closeConnection( Connection con ) {
    if ( con != null ) {
        try {
            con.close();
        } catch ( SQLException e ) {
            System.err.println( "Error cerrando la conexion a la BD" );
            System.err.println( e.getMessage() );
            System.exit( -1 );
        }
    }
}
```

5.3.5. Exportación de datos

Este componente debe permitir transformar los datos a tres formatos diferentes:

- **KML**[46]
- **NetCDF**[47]
- **Property List** [48]

Se exportarán todos los datos contenidos en la base de datos, con lo cual antes de hacer la exportación a cada formato se realizarán las conexiones necesarias a la base de datos para su posterior exportación.

El componente de Exportación de datos se implementará en forma de métodos dentro de la clase **Main** de la aplicación.

Seguidamente se describe la implementación de la exportación a cada uno de los formatos mencionados.

KML

Para realizar la exportación a KML se ha usado, tal como se ha mencionado en la sección 5.3, las librerías Java API for KML y JAXB Reference Implementation, que proporcionan una interfaz de alto nivel y sencilla para crear ficheros KML.

En realidad la implementación no entraña mucha dificultad una vez se ha cogido práctica en el uso de esta API. La forma en la que se exportan los datos es creando un archivo KML que contiene la información para todas las boyas, separada en distintas secciones y mostrando las trayectorias para cada boya en diferentes colores aleatorios.

Un detalle de la implementación sería:

[46] Keyhole Markup Language: http://www.unidata.ucar.edu/software/netcdf
[47] NetCDF: http://www.unidata.ucar.edu/software/netcdf
[48] Property Lists

```
// Creamos KML
            Kml kmltest =  new Kml();
            File kmlFile = new File(exportDirectory+KML_FILENAME);


            String buoyAux = null;
            //Creamos folder buoys
            Folder folder = kmltest.createAndSetFolder();
            folder.setName("Buoys");

            //Añadimos a folder buoys folder boya
            Folder fol = folder.createAndAddFolder();

            //Creamos trayectoria
            Placemark linePlacemark = new Placemark();
            //Creamos linea trayectoria
            LineString lineString = linePlacemark.createAndSetLineString();



            while (rs.next()) {


                //Sacamos todos los datos de la Base de Datos

                String project= rs.getString("project");
                String buoy = rs.getString("buoy");
                String latitude = rs.getString("latitude");
                String longitude = rs.getString("longitude");
                String date = rs.getString("date");
                String temperature = rs.getString("temperature");
                String conductivity = rs.getString("conductivity");
                String salinity = rs.getString("salinity");


                // vamos insertando datos en el KML
                //parte descripcion kml
                String description = "<![CDATA[<b><font color='blue'>ID:</font> " + ↩
                    buoy+
                                    "<br><font color='blue'>Latitude:</font> "+↩
                                        latitude+
                                    "<br><font color='blue'>Longitude:</font> "+↩
                                        longitude+
                                    "<br><font color='blue'>Date:</font> "+date+
                                    "<br><font color='blue'>Temperature:</font> "+↩
                                        temperature+
                                    "<br><font color='blue'>Conductivity:</font> "+↩
                                        conductivity+
                                    "<br><font color='blue'>Salinity:</font> "+↩
                                        salinity+"</b>";
```

El resultado que se obtiene con la exportación se puede ver en la siguiente Figura 19:

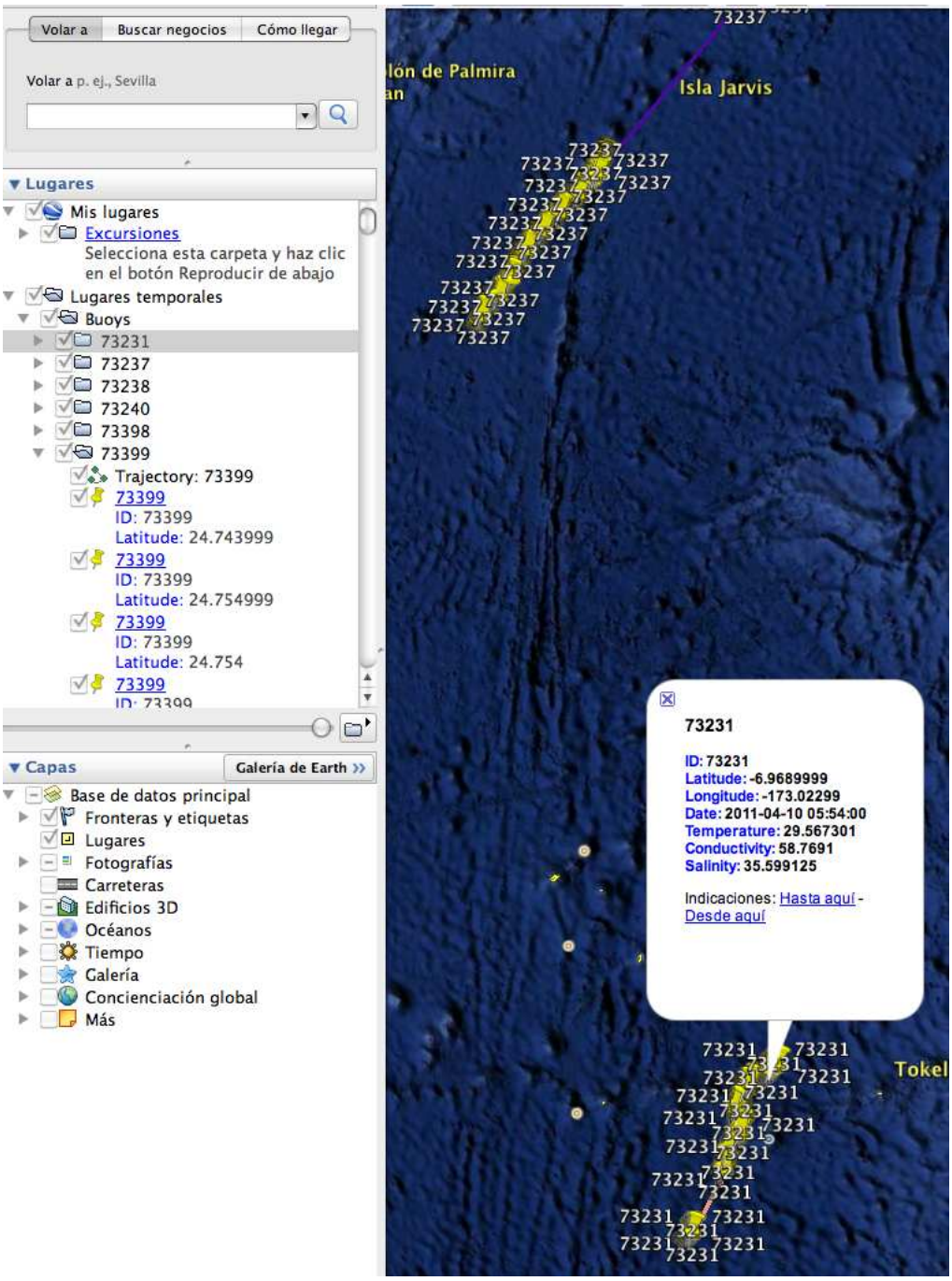

Figura 19: Fichero KML exportado en GoogleEarth

NetCDF

Para realizar la exportación a NetCDF se ha usado, tal como se menciona en la sección 5.3, la librería NetCDF Java, que proporciona una interfaz de alto nivel para crear ficheros NetCDF.

La mayor dificultad que entraña la exportación a este formato usando esta librería es que la inserción de Datos se realiza mediante arrays multidimensionales que han de

ser previamente inicializados, con lo cual es necesario conocer de antemano el número de medidas que tiene cada boya, para ello se han creado métodos auxiliares que devuelven estos datos.

La estructura que se ha decidido dar a los ficheros es la siguiente:

- Dos dimensiones: profundidad, que será siempre cero y tiempo en formato Unix Time[49]. Aunque para este caso la profundidad es siempre fija, es conveniente incluirla para su tratamiento masivo con datos procedentes de otros boyas que toman medidas a diferentes profundidades.
- Incluir como variable todos los datos referentes a cada medida: latitud, longitud, time en formato Unix Time, conductividad, temperatura y salinidad.
- Incluir como variables globales metadatos: identificador de la boya, información de la institución generadora de los datos, convenciones de datos usadas.

A su vez, en base a lo anteriormente mencionado, se ha tenido que implementar un método que calcule el Unix Time.

Esta estructura, aunque es la más usada en ámbito científicos para datos de boyas, representará un problema, y es que al no tener dimensiones de posición y longitud, estos datos no serán reconocidos por ncWMS[50] para ser geoposicionados.

Se ha decidido que se generará un fichero NetCDF para cada boya, que contenga todos los datos que se encuentren en la base de datos para cada una de ellas.

Un detalle de la implementación de la definición del fichero NetCDF sería:

[49]Unix Time: http://en.wikipedia.org/wiki/Unix_time
[50]ncWMS: http://www.resc.rdg.ac.uk/trac/ncWMS

```
//Definimos fichero netCDF
NetcdfFileWriteable nc = NetcdfFileWriteable.createNew(exportDirectory+buoy+".nc",true);

//Dimensiones
Dimension depthDimension = nc.addDimension("detph", 1);
Dimension timeDimension = nc.addDimension("time", medidasPorBoya);


//Define variables
ArrayList dims = new ArrayList();
dims.add(depthDimension);
dims.add(timeDimension);

nc.addVariable("depth", DataType.FLOAT,new Dimension[]{depthDimension});
nc.addVariableAttribute("depth", "units", "meters");
nc.addVariableAttribute("depth","positive", "down");

nc.addVariable("time", DataType.INT,new Dimension[]{timeDimension});
nc.addVariableAttribute("time", "units", "seconds since 1970-1-1 00:00:00");


nc.addVariable("latitude", DataType.FLOAT, new Dimension[]{timeDimension});
nc.addVariableAttribute("latitude", "units", "degrees north");
nc.addVariableAttribute("latitude","valid-min","-90.0f");
nc.addVariableAttribute("latitude","valid-max","90.0f");


nc.addVariable("longitude", DataType.FLOAT,new Dimension[]{timeDimension});
nc.addVariableAttribute("longitude", "units", "degrees east");
nc.addVariableAttribute("longitude","valid-min","-180.0f");
nc.addVariableAttribute("longitude","valid-max","180.0f");

nc.addVariable("temperature", DataType.FLOAT,dims);
nc.addVariableAttribute("temperature", "units", "degree Celsius");
nc.addVariableAttribute("temperature","valid-min","0");
nc.addVariableAttribute("temperature","valid-max","40");



nc.addVariable("conductivity", DataType.FLOAT,dims);
nc.addVariableAttribute("conductivity", "units", "msiemens/cm^3");
nc.addVariableAttribute("conductivity","valid-min","0");
nc.addVariableAttribute("conductivity","valid-max",100);

nc.addVariable("salinity", DataType.FLOAT,dims);
nc.addVariableAttribute("salinity", "units", "psu");
nc.addVariableAttribute("salinity","algorithm","UNESCO Technical Papers in marine Science, 1983");
nc.addVariableAttribute("salinity","valid-min","30");
nc.addVariableAttribute("salinity","valid-max","40");

nc.addGlobalAttribute("buoyID","Buoy ID "+buoy );
nc.addGlobalAttribute("institution","Institute of Marine Sciencies - CSIC, Barcelona, Spain");
nc.addGlobalAttribute("references","http://www.icm.csic.es");
nc.addGlobalAttribute("Conventions","CF-1.4");
```

El resultado para una boya se puede ver en consola o en el sistema THREDDS, en consola para una boya tendríamos los siguientes datos:

```
ncdump Documents/workspace/Boyas/Export/73399.nc
netcdf \73399 {
dimensions:
detph = 1 ;
time = 371 ;
variables:
float depth(detph) ;
depth:units = "meters" ;
depth:positive = "down" ;
int time(time) ;
time:units = "seconds since 1970-1-1 00:00:00" ;
float latitude(time) ;
latitude:units = "degrees north" ;
latitude:valid-min = "-90.0f" ;
latitude:valid-max = "90.0f" ;
float longitude(time) ;
longitude:units = "degrees east" ;
longitude:valid-min = "-180.0f" ;
longitude:valid-max = "180.0f" ;
float temperature(detph, time) ;
temperature:units = "degree Celsius" ;
temperature:valid-min = "0" ;
temperature:valid-max = "40" ;
float conductivity(detph, time) ;
conductivity:units = "msiemens/cm^3" ;
conductivity:valid-min = "0" ;
conductivity:valid-max = 100 ;
float salinity(detph, time) ;
salinity:units = "psu" ;
salinity:algorithm = "UNESCO Technical Papers in marine Science, 1983" ;
salinity:valid-min = "30" ;
salinity:valid-max = "40" ;

// global attributes:
:buoyID = "Buoy ID: 73399" ;
:institution = "Institute of Marine Sciencies - CSIC, Barcelona, Spain" ;
:references = "http://www.icm.csic.es" ;
:Conventions = "CF-1.4" ;
```

```
data:

 depth = 0 ;

 time = 1302311460, 1302318840, 1302324840, 1302326040, 1302330840,
    1302331740, 1302332820, 1302338520, 1302344340, 1302345120, 1302350160,
```

Property List

Para realizar la exportación al formato Property List (PLIST) se ha usado, tal como se menciona en la sección 5.3, la librería Java Property List Library, que proporciona una interfaz de alto nivel para crear ficheros PLIST.

La mayor dificultad que entraña la exportación a este formato usando esta librería es que la inserción de Datos se realiza mediante arrays y, al igual que en la exportación a formato KML, estos arrays han de ser previamente inicializados, con lo cual es necesario conocer de antemano el número de medidas que tiene cada boya, para ello se han creado métodos auxiliares que devuelven estos datos.

Se exportarán a este formato los datos de todas las boyas que estén activas.

La estructura que se ha decidido dar a los ficheros es la que se ha mencionado en la figura 10, con lo cual se tendrá un array con las boyas y dentro de cada boya su identificador y un array con las diferentes medidas.

Un detalle de la implementación de la creación del fichero sería:

```
//Creamos plist
            NSArray buoysArray = new NSArray(numBoyasContador);

            NSDictionary buoyDictionary = new NSDictionary();

            NSArray measuresArray = new NSArray(endMeasureFirst);
```

Y de la escritura de las medidas dentro del PLIST:

```
// creamos e inicializamos diccionario y array nueva boya
                buoyDictionary.put("Measures", measuresArray);
                buoysArray.setValue(j, buoyDictionary);

                System.out.println("Buoys Array indice: "+j);

                buoyDictionary = new NSDictionary();
                buoyDictionary.put("Buoy", new NSString(buoy));
                measuresArray = new NSArray(numMedidas);
                j++;

            }

            NSDictionary measureDictionary = new NSDictionary();

            // Insertamos datos medida en diccionario de medida de la boya
            measureDictionary.put("Conductivity", new NSString(conductivity));
            measureDictionary.put("Date", new NSString(date));
            measureDictionary.put("Temperature", new NSString(temperature));

            //measureDictionary.put("Latitude", new NSString(latitude));
            //measureDictionary.put("Longitude", new NSString(longitude));

            measureDictionary.put("Latitude", new NSNumber(Double.parseDouble(↩
                latitude)));
            measureDictionary.put("Longitude", new NSNumber(Double.parseDouble(↩
                longitude)));

            measureDictionary.put("Salinity", new NSString(salinity));

            //metemos diccionario en array de medidas boya
            measuresArray.setValue(i, measureDictionary);
```

5.3.6. Interfaz aplicación móvil

La interfaz con la aplicación móvil será mediante un fichero Property List que generará el sistema y se hará público en el servidor web en una URL oculta.

Este fichero se irá renovando cada vez que se haga una descarga de datos, y se genere un fichero con datos más actuales.

5.4. Aplicación Buoys dispositivo móvil iOS

Para el desarrollo de la aplicación móvil se utiliza XCode 4 y el iOS SDK 4. A continuación, en la Figura 20 y Figura 21 , se muestra el esquema de clases de toda la aplicación y seguidamente se mencionan los aspectos más relevantes de la implementación de las diferentes secciones de la aplicación.

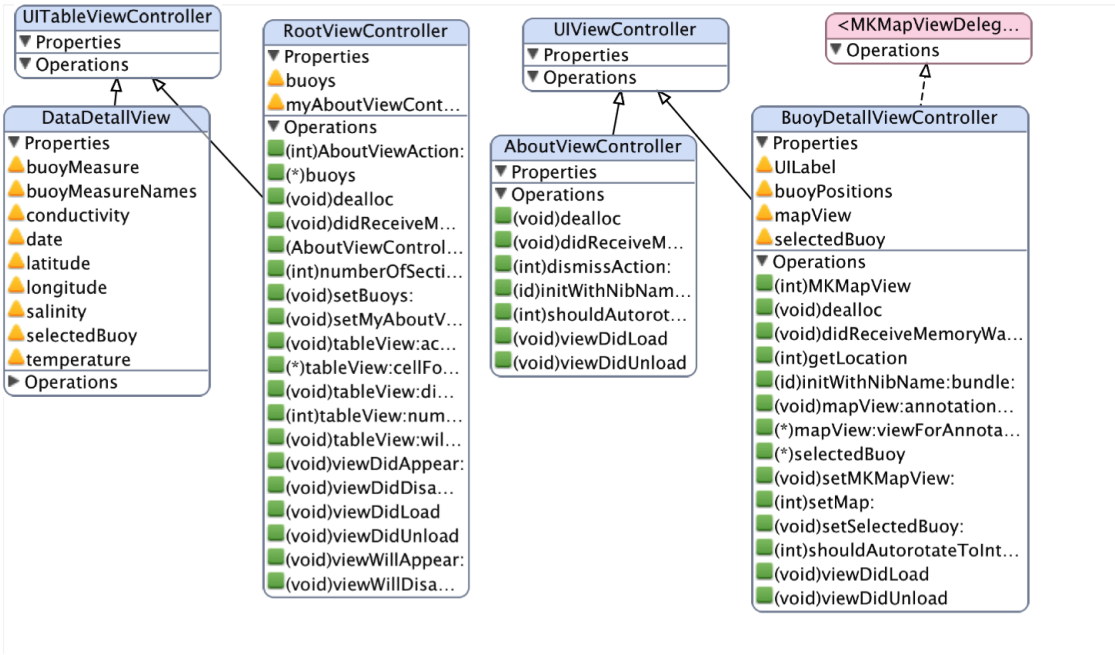

Figura 20: Diagrama de clases 1 Buoys iOS

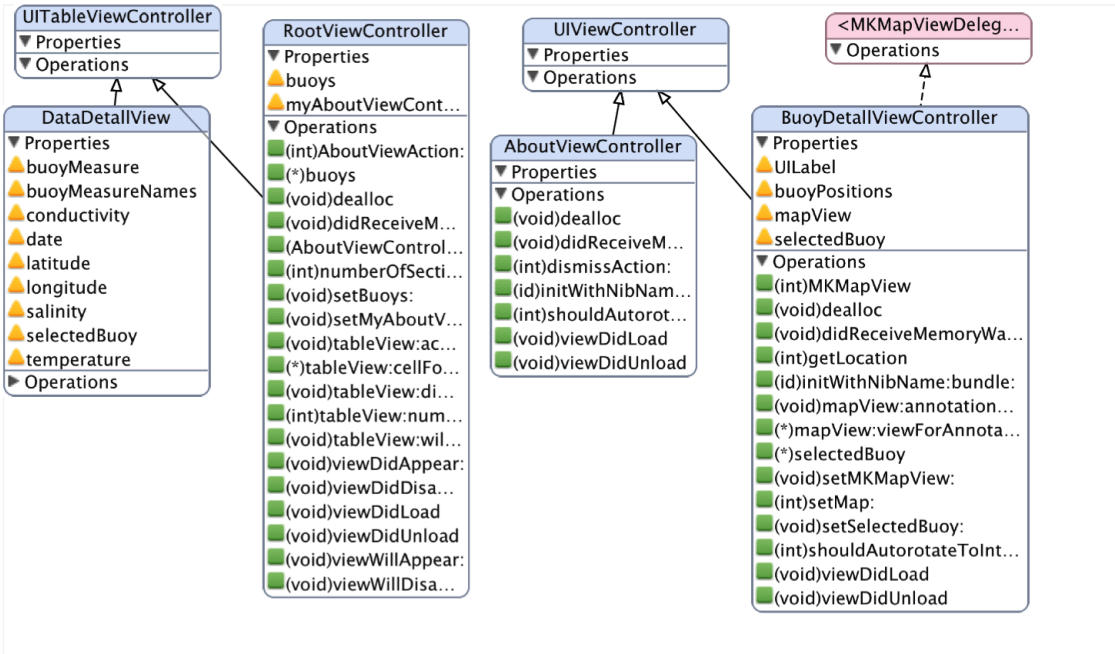

Figura 21: Diagrama de clases 2 Buoys iOS

5.4.1. Select Buoy

Esta será la primera sección de la aplicación, con lo cual, cuando se crea el proyecto, utilizando la plantilla *Navigation-Based Application*, y se procede a crear **RootView-Controller** que generará tres ficheros: un fichero con extensión .m y otro con extensión .h, que serán el modelo y el controlador de esta vista, y otro fichero con extensión .xib que contendrá la vista de la aplicación.

A continuación se escoge el fichero xib que mostrará el XCode *Interface Builder*, y se inserta una tabla en la vista:

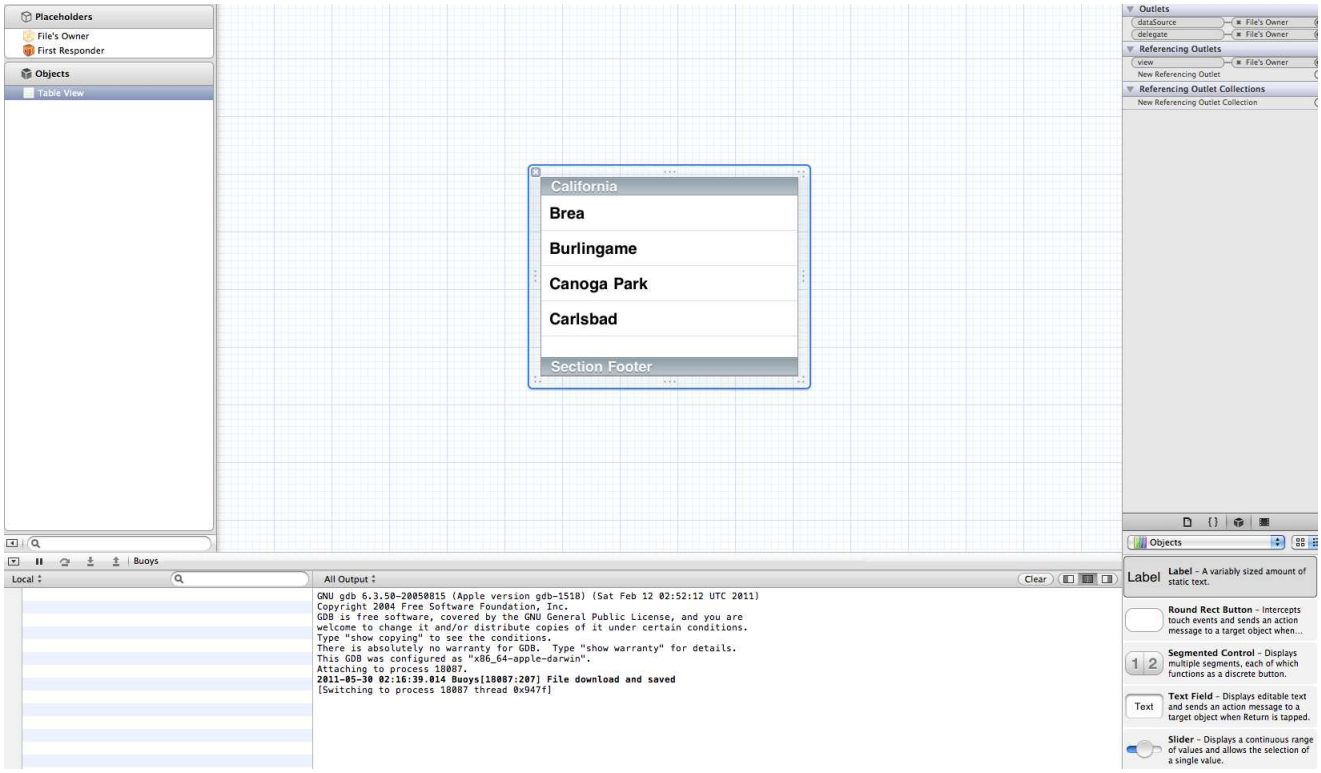

Figura 22: Implementación Select Buoy: XCode Interface Builder

Ahora que ya se tiene la vista, se define la forma en la que se descargan los datos de las boyas, para ello se crea un array que contendrá estos datos en las clases, tanto en la cabecera como en su implementación.

Lo primero es definir como propiedad un array que contendrá los identificadores de las boyas, que será lo que muestre esta vista, primero lo defino en la cabecera:

```
#import <UIKit/UIKit.h>
#import "AboutViewController.h"

@interface RootViewController : UITableViewController {

    NSMutableArray *buoys;
    AboutViewController *myAboutViewController;

}

@property (nonatomic, retain) NSMutableArray *buoys;
@property (nonatomic, retain) AboutViewController *myAboutViewController;

@end
```

Los principales métodos a implementar, y que contienen la lógica de la sección serán:

- **viewDidLoad**: método del controlador de la sección que se encarga de inicializar la vista que se mostrará al usuario. Desde este método se descargarán los datos de las boyas y guardarán los estos datos en su caché, y se encargará de cargar la tabla con los identificadores de las boyas.
- **cellForRowAtIndexPath**: método del protocolo UITableViewDelegate que es llamado automáticamente para cada una de las celdas que se van a mostrar en la tabla de nuestra vista. Este método obtiene los datos de las boyas, cargados por el método viewDidLoad, y muestra sus identificadores en cada una de las celdas de la tabla.
- **didSelectRowAtIndexPath**: método del protocolo UITableViewDelegate que es llamado cuando se pulsa una de las celdas por el usuario. En este método se pasa el identificador pulsado a la siguiente vista, que mostrará la boya seleccionada geoposicionada.

También se implementan otros métodos, que permiten cambiar la apariencia y fijar la navegación de la aplicación, como puede ser por ejemplo **willDisplayCell**, que cambia el color a celdas alternas, es decir, da un color a las celdas pares y otro a las impares.

Un detalle del método **viewDidLoad** sería:

```
@synthesize buoys,myAboutViewController;

- (void)viewDidLoad
{
    [super viewDidLoad];

    //Inicializamos array


    buoys = [[NSMutableArray alloc] init];


    // Actualizamos datos desde URL

    NSString *docsDir = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, ↩
        NSUserDomainMask, YES) lastObject];

     NSString *fileToDownload = @"buoys.plist";
     NSString *hostURLString = @"http://tarod.cmima.csic.es/iOSFiles";
     hostURLString = [hostURLString stringByAppendingPathComponent: fileToDownload];
     NSURL *pListURL = [NSURL URLWithString: hostURLString];
     NSData *pListURLData = [NSData dataWithContentsOfURL: pListURL];


     NSString* filepath = [docsDir stringByAppendingPathComponent: fileToDownload];

    BOOL ok = [pListURLData writeToFile: filepath atomically: YES];
```

En este detalle se ve que para las propiedades que se han fijado en la cabecera se crean sus *getters* y *setters* con *@synthesize*, y se intentará descargar los datos desde una URL. Se avisará al usuario mediante una ventana emergente si se han podido actualizar los datos, y, en caso de que no se puedan descargar o actualizar, se avisará al usuario y éste podrá seguir consultando los datos que hayan sido descargados en anteriores conexiones y que están almacenados en la caché de la aplicación.

5.4.2. About

Para implementar esta sección, igual que en la anterior, se utiliza una plantilla de XCode, creando una nueva subclase de **UIViewController**, llamada **AboutViewController**.

Una vez creada, desde *Interface Builder*, se le añade un campo de texto, y un botón que conectará con la acción **dismissAction**, que se creará en las clases de la sección.

Seguidamente hay que fijar desde donde se quiere llamar esta sección, se decide llamarla desde **RootViewController**, para ello la llamamos desde el método **viewDidLoad**, creando un botón para tal fin:

```
// About Buttom
    UIButton* AboutViewButton = [UIButton buttonWithType:UIButtonTypeInfoLight];
    [AboutViewButton addTarget:self action:@selector(AboutViewAction:) forControlEvents↩
        :UIControlEventTouchUpInside];
    UIBarButtonItem *AboutBarButtonItem = [[UIBarButtonItem alloc] initWithCustomView:↩
        AboutViewButton];
    self.navigationItem.rightBarButtonItem = AboutBarButtonItem;
    [AboutBarButtonItem release];
```

Y seguidamente se crea la acción que mostrará esta sección, dentro de RootViewController, usando un Modal View Controller [51]:

```
- (IBAction)AboutViewAction:(id)sender
{
    if (self.myAboutViewController == nil)
        self.myAboutViewController = [[[AboutViewController alloc] initWithNibName:
                                        NSStringFromClass([AboutViewController class]) ↩
                                            bundle:nil] autorelease];

    [self.navigationController presentModalViewController:self.myAboutViewController ↩
        animated:YES];
}
```

De esta forma se mostrará un botón a la derecha de la barra de navegación, dentro de la sección *Select Buoy*.

5.4.3. View Map

Para implementar esta sección lo primero a realizar es importar los frameworks que permiten mostrar datos geoposicionados en un mapa, para ello se ha de importar al proyecto los frameworks MapKit y CoreLocation.

Seguidamente se añaden nuevas clases de la misma forma que las secciones anteriores con el nombre **BuoyDetallViewController**, con la diferencia de que será necesario que se importe en estas clases MapKit.

También será necesario definir varias propiedades o variables globales:

- **selectedBuoy**: NSString que contiene el identificador de la boya seleccionada por el usuario.
- **mapView**: MKMapView que contiene el mapa a mostrar.

[51] Modal View Controller

- **buoyPositions**: array que contendrá las posiciones de la boya seleccionada por el usuario.

Llegados a este punto hará falta una clase auxiliar para mostrar las posiciones, por lo tanto se crea, usando una plantilla de Xcode, una nueva Objective-C class, llamada **DisplayMap** que creará sólo los ficheros de clase, cabecera y controlador sin fichero xib de vista. Los objetos de esta clase serán los que permiten mostrar los datos sobre el mapa.

En la clase **BuoyDetallViewController** se inicializa el mapa dentro del método **viewDidLoad** cargando los datos desde el fichero PLIST:

```
buoyPositions = [[NSMutableArray alloc] init];


//Cargamos plist desde local
// NSString *path=[[NSBundle mainBundle] pathForResource:@"buoys.ºfType:@"plist"];

//add para descargado de la web
NSString *docsDir = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, ↩
    NSUserDomainMask, YES) lastObject];
NSString *file = @"buoys.plist";
NSString* path = [docsDir stringByAppendingPathComponent: file];

NSMutableArray  *plistData = [[NSMutableArray arrayWithContentsOfFile:path]retain];

//leemos plist
for(NSDictionary *dict in plistData) {


    NSString *value;
    value = [dict objectForKey:@"Buoy"];

    //Si encontramos boya seleccionada leemos datos
    if ([value isEqualToString:selectedBuoy]) {

        buoyPositions = [dict objectForKey:@"Measures"];
        double minLat = [[buoyPositions valueForKeyPath:@"@min.Latitude"] ↩
            doubleValue];
        double maxLat = [[buoyPositions valueForKeyPath:@"@max.Latitude"] ↩
            doubleValue];
        double minLon = [[buoyPositions valueForKeyPath:@"@min.Longitude"] ↩
            doubleValue];
        double maxLon = [[buoyPositions valueForKeyPath:@"@max.Longitude"] ↩
            doubleValue];

        MKCoordinateRegion region;
        region.center.latitude = (maxLat + minLat) / 2.0;
        region.center.longitude = (maxLon + minLon) / 2.0;
        region.span.latitudeDelta = (maxLat - minLat) * 3.05;
        region.span.longitudeDelta = (maxLon - minLon) * 3.05;
        mapView.region = region;
```

Y seguidamente se crean los objetos **DisplayMap** que contendrán las diferentes me-

didas:

```
            //Mostramos datos medidas
             for (NSDictionary *dictMeasures in buoyPositions) {

                 DisplayMap *measure = [[DisplayMap alloc] initWithDictionary:↩
                     dictMeasures];
                 [mapView addAnnotation:measure];
                 [measure release];


             }
```

Que serán mostrados en el mapa, junto con las anotaciones o callOuts (título y subtítulo):

```
- (MKAnnotationView *)mapView:(MKMapView *)mapView viewForAnnotation:(id <MKAnnotation↩
    >)annotation{

    MKPinAnnotationView *pinView = nil;
    if(annotation != self.mapView.userLocation)
    {
        static NSString *defaultPinID = @"com.invasivecode.pin";
        pinView = (MKPinAnnotationView *)[self.mapView ↩
            dequeueReusableAnnotationViewWithIdentifier:defaultPinID];
        if ( pinView == nil ) pinView = [[[MKPinAnnotationView alloc]
                                          initWithAnnotation:annotation reuseIdentifier↩
                                              :defaultPinID] autorelease];

        pinView.pinColor = MKPinAnnotationColorRed;
        pinView.canShowCallout = YES;
        pinView.animatesDrop = YES;

        pinView.rightCalloutAccessoryView = [UIButton buttonWithType:↩
            UIButtonTypeDetailDisclosure];


        self.mapView.delegate=self;
    }
    else {
        [self.mapView.userLocation setTitle:@"I am here"];
    }
    return pinView;
}
```

Que se fijen en los objetos **DisplayMap**:

```
@implementation DisplayMap

@synthesize coordinate,title,subtitle,latitude,longitude,date,temperature,conductivity,↩
    salinity;



- (id) initWithDictionary:(NSDictionary *) dict
{
    self = [super init];
    if (self != nil) {
        coordinate.latitude = [[dict objectForKey:@"Latitude"] doubleValue];
        coordinate.longitude = [[dict objectForKey:@"Longitude"] doubleValue];
        self.title = [NSString stringWithFormat: @"Date: %@",[dict objectForKey:@"Date"↩
            ]];

        NSString *latAux = [NSString stringWithFormat: @"Lat:  %@",[dict objectForKey:@↩
            "Latitude"]];

        NSString *lonAux = [NSString stringWithFormat: @"Lon:  %@",[dict objectForKey:@↩
            "Longitude"]];
        latAux = [latAux stringByAppendingString:@" "];


        // Fijamos variables que vamos a pasar
        self.latitude = [NSString stringWithFormat: @"%@",[dict objectForKey:@"Latitude↩
            "]];
        self.longitude = [NSString stringWithFormat: @"%@",[dict objectForKey:@"↩
            Longitude"]];;
        self.date = [dict objectForKey:@"Date"];
        self.temperature = [dict objectForKey:@"Temperature"];
        self.conductivity = [dict objectForKey:@"Conductivity"];
        self.salinity = [dict objectForKey:@"Salinity"];


        self.subtitle = [latAux stringByAppendingString:lonAux];
    }
    return self;
}
```

Mencionar también que en el objeto se cargan todos los datos que contiene el diccionario de la medida, para después pasarlos a la siguiente sección, cuando el usuario pulse la llamada callOut en la anotación de una medida que llamará al método **calloutAccessoryControlTapped**.

Añadir que se implementan también otros métodos que cambian las propiedades del mapa, entre ellos:

- **getLocation**: método que cambia la región del mapa y muestra la posición del usuario. Irá asociada a un botón en la vista.
- **setMap**: permite al usuario escoger el tipo de mapa, entre satélite o mapa normal.

Comentar que no se ha implementado el tipo satélite híbrido porque para mostrar datos marinos no tiene ninguna utilidad, el mapa normal la tiene si el usuario visualiza datos con poco ancho de banda disponible.

5.4.4. Measure Detail

Al igual que para anteriores secciones se utiliza una plantilla de XCode, creando una nueva subclase de **UIViewController**, llamada **DataDetallView**.

Se inicializa el objeto en el método **viewDidLoad** con los datos que se le han pasado con el método **calloutAccessoryControlTapped** de **BuoyDetallViewController**.

Seguidamente se muestran los datos en una tabla creada con el *Interface Builder* en la vista, que recibe los datos a través del método **cellForRowAtIndexPath**, donde se cargan en las celdas los arrays, creados previamente, con los datos y las leyendas:

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(↩
    NSIndexPath *)indexPath {

    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier↩
        ];
    if (cell == nil) {
        //cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero reuseIdentifier:CellIdentifier] autorelease];
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleValue1 ↩
            reuseIdentifier:CellIdentifier] autorelease];




    }


    // Set up the cell...
    NSString *cellValue = [buoyMeasure objectAtIndex:indexPath.row];
    NSString *cellDetail = [buoyMeasureNames objectAtIndex:indexPath.row];
    //cell.text = cellValue;
    cell.textLabel.text = cellDetail;
    cell.detailTextLabel.text = cellValue;
    cell.textLabel.textColor = [UIColor blueColor];
    cell.textLabel.font = [UIFont systemFontOfSize:16];
    cell.detailTextLabel.font = [UIFont systemFontOfSize:14];

    return cell;
}
```

También se implementan otros métodos que fijan diferentes propiedades de apariencia

en la visualización de los datos y la navegación, que se pueden ver en el código fuente de la aplicación.

6. Conclusiones

En este proyecto se ha conseguido trabajar con tecnologías muy dispares: desde aplicaciones de servidor, bases de datos, virtualización, programación en Java, diferentes APIs y frameworks, Objective-C e iOS SDK para dispositivos móviles y una gran variedad de formatos de datos de los cuales no se tenían conocimientos previos, integrándolos todos ellos en un mismo sistema.

También creo importante mencionar que se ha conseguido producir un documento usando LaTeX, lo cual también era para el autor un objetivo importante.

En base a todo esto creo que se han cumplido los principales objetivos mencionados en la sección 1.4 de Objetivos Iniciales.

Por contrapartida creo que se ha minusvalorado el tiempo de implementación de todo el sistema, con lo cual a partir de la fase de implementación se ha producido una desviación importante respecto a la planificación inicial. Esto es debido a que ha sido necesario emplear mucho más tiempo del inicialmente estimado para la finalización de este proyecto, debido a que los objetivos planteados para la aplicación de servidor han sido demasiado ambiciosos y no se tenía ninguna experiencia previa en programación para dispositivos móviles con iOS.

A pesar de todo esto la experiencia ha sido muy satisfactoria y ha permitido desarrollar aplicaciones en tecnologías que, aunque tienen una alta curva de aprendizaje, a parte del enriquecimiento personal, permiten ampliar las perspectivas de futuro.

Finalmente destacar que el bagaje y la base dada en los estudios de Ingeniería Informática de la UOC ha sido, sin duda, un factor clave para el empleo con éxito de las diversas tecnologías usadas para este proyecto.

7. Trabajos futuros

Como trabajos futuros, en cuanto a las aplicaciones del ICM Server, para próximos proyectos se plantea lo siguiente:

1. Crear una aplicación web que contenga los formularios necesarios para administrar la base de datos del sistema y así poder delegar la función de administrar las boyas a usuarios sin conocimientos avanzados de bases de datos.

2. Crear unas herramientas que permitan desde consola:

   a) Descargar datos en formato NetCDF y csv para cada boya, escogiendo rangos temporales, datos que se desea contenga el fichero, etc.

   b) Descargar datos en formato KML personalizado por el usuario: por boya, por rangos temporales, seleccionar medidas de interés.

   c) Mostrar datos seleccionados por diferentes criterios en pantalla.

3. Crear una aplicación web que permite visualizar los datos en Google Maps.

4. Implementar en el sistema nuevos sistemas de satélite, lo cual supone implementar nuevos parser que interpreten los datos/tramas de estos sistemas.

5. Integrar los datos del sistema en SIG [52] institucional.

Respecto a la aplicación móvil se plantea lo siguiente para futuros proyectos:

1. Implementar vista por regiones personalizables en las que se muestren las diferentes boyas que se encuentren en esta región.

2. Implementar estadísticas para los datos las diferentes boyas: estas estadísticas han de realizarse con gráficas y el usuario debe poder visualizar estadísticas para las boyas por él seleccionadas.

3. Implementar otras vistas para los datos de las boyas por rangos de fechas y de posición.

4. Crear alertas para eventos destacados, como pueden ser por ejemplo que se recojan datos oceanográficos singulares o destacados.

5. Mejorar la interfaz de usuario creando nuevos iconos y una nueva imagen de lanzamiento (splash image).

6. Añadir soporte multi-idioma que incluya más idiomas aparte del inglés.

[52] Sistema de información geográfica

8. Bibliografía

Libros

Beginning iPhone 4 Development, Dave Mark & Jack Nutting & Jeff LaMarche, Apress.

The Objective-C Programming Language, Apple Inc.

Webs

http://en.wikibooks.org/wiki/LaTeX: Wiki y tutorial de LaTeX.

http://download.oracle.com/javase/6/docs/api: Java™ Platform, Standard Edition 6 API Specification.

http://code.google.com/p/plist: dd-plist Library Site.

http://labs.micromata.de/display/jak/Home: Documentación Java API for KML.

http://code.google.com/intl/es-ES/apis/kml/documentation: Documentacion KML.

http://www.unidata.ucar.edu/software/netcdf-java/tutorial: Documentación NetCDF Java.

http://developer.apple.com/library/ios/navigation: iOS Developer Library.

http://www.iphonedevsdk.com: Foro de desarrolladores de iOS.

http://cocoadevcentral.com: Web con gran cantidad de tutoriales de iOS.

http://stackoverflow.com: Foro sobre programación con gran cantidad de temas sobre iOS.

Multimedia

http://itunes.apple.com/us/itunes-u/developing-apps-for-ios-hd/id395605774: Developing Apps for iOS, Paul Hegarty, Stanford University.

Glosario

ARGO programa de boyas a la deriva (transportadas por la corriente) dotadas de sensores de medida de propiedades físico-químicas del agua.

ARGOS sistema de monitorización y localización por satélite.

CLS Acrónimo de Collecte Localisation Satellites. Servicio que almacena y distribuye los datos del sistema de satélites ARGOS.

KML Acrónimo de Keyhole Markup Language. Es un lenguaje de marcado basado en XML para representar datos geográficos en tres dimensiones, estandarizado por el Open Geospatial Consortium.

MVC Acrónimo de Model–view–controller. Patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos..

ncWMS Servicio Mapas Web para datos geoespaciales que son almacenados en ficheros que cumplen los estándares NetCDF.

NetCDF Acrónimo de Network Common Data Form. Conjunto de librerías que permiten el acceso, creación y compartición de ficheros que contienen datos científicos multimensionales. Estas librerías son multiplaforma y siguen un standard abierto .

NOAA Acrónimo de National Oceanic and Atmospheric Administration. Agencia científica del Departamento de Comercio de los Estados Unidos cuyas actividades se centran en las condiciones de los océanos y la atmósfera.

Objective-C Lenguaje de programación orientado a objetos creado como un superconjunto de C pero que implementase un modelo de objetos parecido al de Smalltalk. Originalmente fue creado por Brad Cox y la corporación StepStone en 1980. En 1988 fue adoptado como lenguaje de programación de NEXTSTEP y en 1992 fue liberado bajo licencia GPL para el compilador GCC. Actualmente se usa como lenguaje principal de programación en Mac OS X y GNUstep..

OGC Acrónimo de Open Geospatial Consortium. Fundación cuyo fin es la definición de estándares abiertos e interoperables dentro de los Sistemas de Información Geográfica y de la World Wide Web.

PostgreSQL Sistema de gestión de base de datos relacional orientada a objetos y libre, publicado bajo la licencia BSD.

Property List Ficheros usado en frameworks de programación de Mac OS X, iOS, NeXTSTEP y GNUstep que almacenan datos serializables.Tiene formato XML y contienen diferentes tipos de datos en arrays y estructuras llamadas Dictionary .

THREDDS Acrónimo de Thematic Realtime Environmental Distributed Data Services. Middleware que permite el acceso a datos NetCDF.