Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Desarrollo de un servidor para misiones descritas en C-BML

Autor: Juan José Villanueva Borrego
Tutor: Jesús Iván Maza Alcañiz

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Desarrollo de un servidor para misiones descritas en C-BML

Autor

Juan José Villanueva Borrego

Tutor

Jesús Iván Maza Alcañiz

Profesor Contratado Doctor

Departamento de Ingeniería de Sistemas y Automática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2015

Agradecimientos

He de expresar mi profundo agradecimiento a todas aquellas personas que me han dado la oportunidad de desarrollarme, tanto personal como profesionalmente. Su influencia ha sido clave para llegar al punto en el que encuentro ahora mismo, finalizando una carrera como Ingeniero.

No puedo nombrar a todos, pero si quiero reconocer el valor de algunos de ellos.

En primer lugar quiero agradecer a mi tutor, Ivan Maza Alcañiz, el haberme dado la confianza y la posibilidad de trabajar en un proyecto de tal magnitud, que ofrece cantidad de posibilidades futuras.

Por otro lado, dar las gracias a Alfredo Vázquez Reyes por su dedicación y excelente trabajo en el diseño y desarrollo del cliente de misiones CBML, parte fundamental en la ejecución de este proyecto.

También quiero agradecer, por supuesto, a mis padres todo el apoyo y el cariño prestado estos años, en los buenos y en los malos momentos, así como la educación y la infinidad de ayuda recibida de su parte.

Un recuerdo también para mis abuelos, que seguro que se encuentran orgullosos de que su nieto por fin sea Ingeniero. Esto va especialmente por ustedes.

Agradecer a todos los profesores de la Universidad, del colegio, y amigos también su cariño y apoyo mostrados.

En último lugar, y no por eso menos importante, agradecer todo el apoyo incondicional recibido por parte de mi pareja, Roxanna Mariela, y por las numerosas peleas que no hacían más que empujarme a conseguir mis logros y acabar con las tareas que tenía pendientes de hace tiempo, entre las que se encontraba el graduarme como Ingeniero de Telecomunicaciones.

Sin todos ellos, esto nunca habría sido posible.

Gracias a todos.

Resumen

En este proyecto se tratará la casuística de interoperabilidad entre sistemas mediante el uso de un lenguaje de combate llamado C-BML (Coalition Battle Management Language). Este desarrollo lo lleva a cabo el GRVC (Grupo de Robótica Visión y Control) de la Universidad de Sevilla dentro del contexto del proyecto CITIUS (Command and control for Interoperability of Unmanned Systems).

Para el desarrollo de este proyecto se ha llevado a cabo una extensa investigación tanto en entornos de interoperabilidad como de software e infraestructura de comunicaciones que permitiesen obtener el mejor diseño según las restricciones que en el diseño se proponen.

C-BML es un lenguaje de combate recientemente publicado en su versión oficial (Abril 2014) por lo que los diseños existentes no son del todo maduros y llevarán a replantear el diseño en multitud de ocasiones durante el transcurso de los desarrollos. Para ser más exactos, el desarrollo de estas tareas se comenzó cuando todavía no se había publicado una versión terminada y oficial del sistema.

De manera resumida los pasos que se han llevado en este estudio son:

- Investigación de los sistemas existentes basados en C-BML.
- Implementación y pruebas de estos sistemas desarrollados hasta el momento.
- Implementación de un nuevo sistema basado en el diseño técnico y las características de los anteriores que permitiese culminar las características de diseño que se proponen en la tarea.

Índice

Índice de Imágenes

1 INTRODUCCIÓN

1.1 Antecedentes

Tras varios años de desarrollo centrados en la creación de un lenguaje de combate desambiguo y amplio para la especificación de todo tipo de misiones se llega a la necesidad de un sistema servidor que almacene las misiones y datos generados con este lenguaje.

Es entonces cuando se decide (en 2007) la creación del sistema JBML:

"El lenguaje Joint BML (JBML) se desarrolla como un lenguaje inequívoco para orders y reports. Este lenguaje es una contribución de los EEUU al actual lenguaje C-BML de la SISO y su a grupo de desarrollo.

Se centra en la aplicación de los principios bien conocidos de la BML en el contexto war fighting para el intercambio de misiones entre sistemas C2 y de simulación. El diseño de JBML se caracteriza por estar formado por diferentes capas que permiten soluciones configurables, no sólo desde la perspectiva del sistema de información, sino también desde una vista del intercambio de información de dominio específico.

La idea principal es usar un lenguaje desambiguo que además use el estándar JC3IEDM. Los servicios se implementan como servicios Web de soporte a C-BML Phase 1. La configuración de dominio utiliza un esquema motivado por el trabajo inicial en la gramática formal, destinado a apoyar C BML-Phase 2. El servicio Web se configura con este conocimiento específico definido en los archivos XML Schema. Las codificaciones de datos están estrechamente conectadas con el estándar JC3IEDM, aunque los niveles más altos de JBML introducen abstracciones que encapsulan la complejidad del modelo de datos subyacente. La intención es hacer la aplicación coherente a JBML con una interfaz que use un lenguaje directo."

Entrando en el diseño que implementa, se propone un diseño en 5 capas.

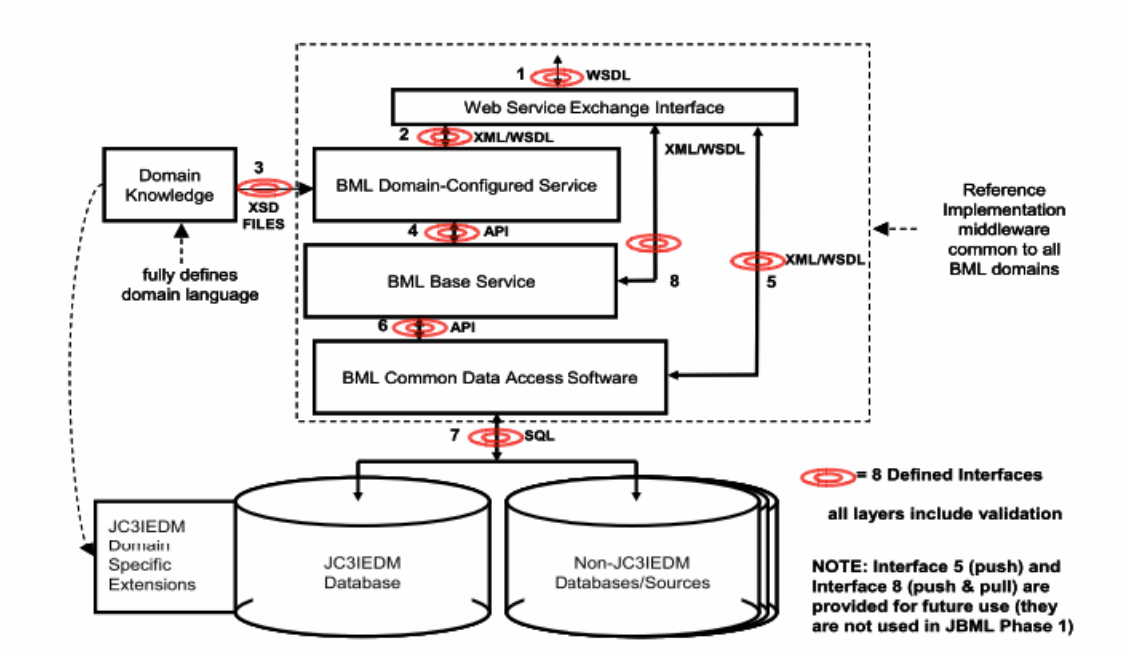

Imagen 1 - Esquema de capas de JBML

De manera resumida, las peticiones recibidas en XML son validadas en cada una de las capas siguiendo los XML Schema (XSD) asociados a estas. Estas peticiones son enviadas a una base de datos usando el lenguaje SQL. La base de datos implementa todos los tipos de datos definidos en el estándar JC3IEDM. A modo de ejemplo se ha comprobado la instalación de la base de datos MySQL y se ha comprobado que contiene los datos según el estándar JC3IEDM.

Para más información, se puede consultar el repositorio de archivos que define el lenguaje JBML

https://netlab.gmu.edu/JBML/

Desde el repositorio de archivos JBML se puede descargar el archivo **jc3iedm_jbml_v1_5.sql**

Puede descargar este archivo en

https://netlab.gmu.edu/JBML/OpenSource/jc3iedm_jbml_v1_5.sql

Una vez desarrolladas las primeras versiones, que solo incluían algunos tipos de Orders, se integran con otros sistemas como *geoBML* para ofrecer una mejor interoperabilidad, y se realizan las primeras pruebas de simulación.

En 2008 se lleva a cabo la *NATO MSG-048 Coalition Battle Management Initial* en las que se detalla las pruebas realizadas para conseguir interoperabilidad usando JBML unido a diferentes capas de adaptación de los diferentes sistemas de cada país. Este experimento pretende sobretodo mostrar la robustez del lenguaje BML como lenguaje intermedio en la traducción de datos entre diferentes sistemas. Si todos los sistemas propietarios pueden acabar presentando sus datos mediante el lenguaje BML, estos datos pueden ser almacenados en una base de datos común y ser traducidos de nuevo al formato propietario que se desee.

En 2009 se llevan a cabo una serie de mejoras y pruebas siguiendo de nuevo un enfoque de dar interoperabilidad entre diferentes países. Una de las novedades incluidas por EEUU en este experimento es el uso del lenguaje *Scripted BML (SBML)* el cual se nombra en la conclusión como un avance muy importante en el desarrollo así como un sistema prometedor.

Estos cambios y los resultados obtenidos provocan un cambio en la dirección del proyecto, ya que se sustituye JBML por SBML dejando únicamente la base de datos basada en JC3IEDM.

JBML no queda descartado, sino que se une a otras plataformas como *geoBML* y *Army OPORD* para formar *IBML (Integrated Battle Management Language)* un lenguaje que pretende proveer servicios web con datos basados en el estándar de gramática C2LG.

Además ahora pasa de estar basado en *IDEF1x diagrams* de JBML a estar basado también en el intercambio de archivos XML que propone SBML, evitando los formatos tediosos de JBML. Continúa usando JAVA pero ahora usa pársers para archivos XML genéricos fáciles de encontrar online y de código abierto.

Por último se sustituyen las capas de escritura en la base de datos que poseía JBML por un sistema de acceso que además incluye la posibilidad de publicador/subscriptor.

Con estos cambios en la base del proyecto se desarrolla el estándar *SBMLServer* que llega hasta nuestros días, siendo mejorado año tras año, añadiendo nuevas funcionalidades y mejorando su interoperabilidad.

1.2 Objetivos

El propósito de este proyecto es la realización de un sistema que permita el intercambio de mensajes C-BML entre los distintos sistemas de C2. Para ello debe existir una base de datos común que mantenga actualizado el estado de las misiones y permita el conocimiento al resto de C2.

Aprovechando que C-BML usa archivos XML para el intercambio de las misiones y que XML se define como un lenguaje para almacenamiento de datos, usualmente datos web, se va a implementar un servidor con una base de datos MySQL que mantenga actualizado el estado de las misiones y use servicios web para realizar envíos y peticiones.

La base de datos seguirá el estándar JC3IEDM definido por la NATO, ya que este estándar sirve como base para el estándar C-BML definido por la SISO.

Para la realización del procesamiento y gestión de las misiones C-BML y las peticiones de estas, se llevará a cabo una búsqueda de herramientas a tal fin. En caso de no encontrarla, se implementará una que cumpla todas las especificaciones que aquí se proponen.

También sería aconsejable el uso de mecanismos de publicador/subscriptor para que los C2 estén actualizados en todo momento y dispongan de la última información sobre el estado de las misiones que se están llevando a cabo.

Por tanto, a grandes rasgos se buscarán los siguientes objetivos:

- Base de datos MySQL que siga el sistema de tipos de datos JC3IEDM.
- Servidor para peticiones de actualización de la base de datos y realización de consultas.
- Mecanismos que simulen las funcionalidades de un publicador/subscriptor.

Algunas restricciones que han de tenerse en cuenta son:

- Deberá procesar correctamente todos los ejemplos y misiones C-BML desarrollados en el contexto del proyecto CITIUS. Entre ellos destacar los siguientes:
    - **Who_type.xml:** Define los elementos que se pueden encontrar en el espacio de misiones.
    - **Who_association.xml:** Define las posibles asociaciones entre distintas entidades.
    - **Organisation_structure.xml:** Proporciona una referencia a la entidad que se considera como raíz de la organización
    - **Candidate_target_list_search_and_rescue.xml:** Se utiliza para especificar listas de objetivos de cualquier tipo de misión.
    - **Candidate_target_list_reconnaisance.xml:** Se utiliza para especificar listas de objetivos de cualquier tipo de misión.
- Debe controlarse la consistencia de los datos para evitar problemas en la realización de misiones por diferentes C2.

1.3 Diseño propuesto

La arquitectura planteada se basa en el siguiente modelo de capas:

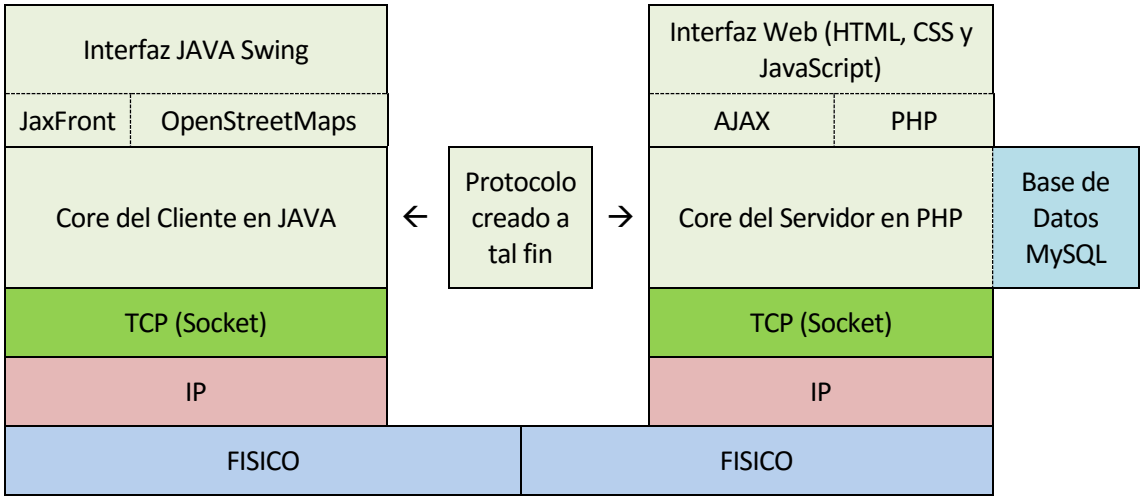

Imagen 2 - Modelo de Capas de la plataforma para interoperabilidad basada en C-BML

Analizando el modelo de capas aquí expuesto, se puede observar que está basado en un intercambio de mensajes TCP/IP. Los mensajes XML de C-BML así como otros posibles comandos para comunicarse con el servidor y realizar un control de los temas (topics) han sido encapsulados bajo un protocolo que ha sido creado específicamente para este propósito. Se basa en el intercambio de cadenas de caracteres (string), fácilmente soportables por los sockets TCP.

Como se puede deducir del esquema, cualquier cliente, escrito en cualquier lenguaje de programación que soporte la comunicación por Sockets podrá comunicarse con el servidor siempre que use el protocolo creado a tal fin, que como se acaba de comentar, está basado en el intercambio de cadenas de caracteres (string).

Para más información sobre el protocolo diseñado, consultar el Apéndice C.

---

2. ESTADO DEL ARTE

En este apartado se van a presentar las infraestructuras de telecomunicaciones usadas por los EEUU y la OTAN en los procesos de desarrollo del grupo de investigación MSG-085 (anteriormente llamado MSG-048).

Estos estudios se basan en el uso de BML (Battle Management Language), lenguaje desambiguo que ha sido creado tras el estudio durante más de 5 años de los diferentes formatos de misiones y órdenes que se producían en los mandos militares.

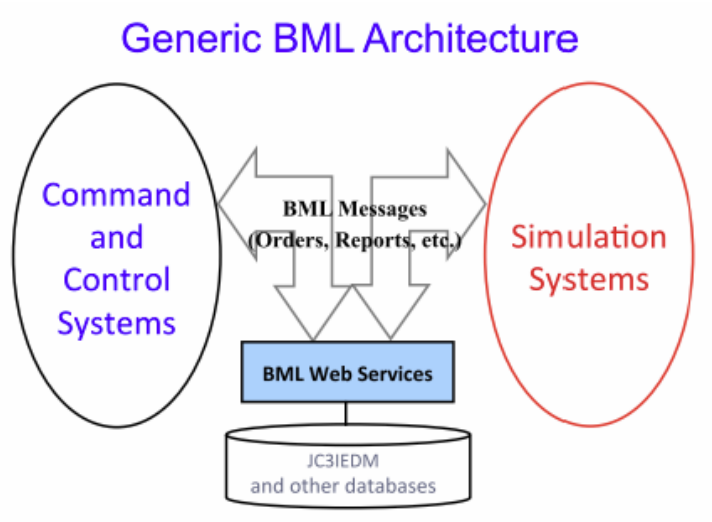

Imagen 3 - Arquitectura genérica a implementar

2.1 Servidores para C-BML y MSDL

2.1.1. Funciones principales

- Debe brindar la posibilidad de postear nuevos documentos XML: Order y Report en C-BML y escenarios en MSDL.
- Aceptar subscripciones a un cierto tema (Topic).
- Responder a peticiones de archivos llevada a cabo por clientes.

2.1.2. Otras funciones que debe implementar el Servidor

- Namespaces. Permiten el uso de diferentes formatos y lenguajes de manera segura.
- Validación de archivos. Gracias al archivo XSD los archivos XML pueden ser validados comprobándose la formalidad del lenguaje establecido y evitando problemas de incompatibilidades.
- Filtrado de datos. Restricciones impuestas por el criterio del usuario.
- Uso de multihilos. Para mejorar el rendimiento en el procesamiento de peticiones.
- Temas (topics) dinámicos para el publicador/subscriptor. Permite la modificación de los temas en tiempo de ejecución, al contrario que los temas estáticos creados en el arranque del servidor.
- Uso de múltiples interfaces. Como podrían ser SOAP como mecanismo tradicional o uso de sistemas más avanzados como RESTful.

2.1.3. Servidores actuales en el mercado

2.1.3.1. Scripted BML Server (SBML)

Ha sido desarrollado por GMU C4I Center para apoyar los desarrollos propuestos por el grupo de investigación MSG-048. Es de código abierto y promete ofrecer implementaciones para C-BML.

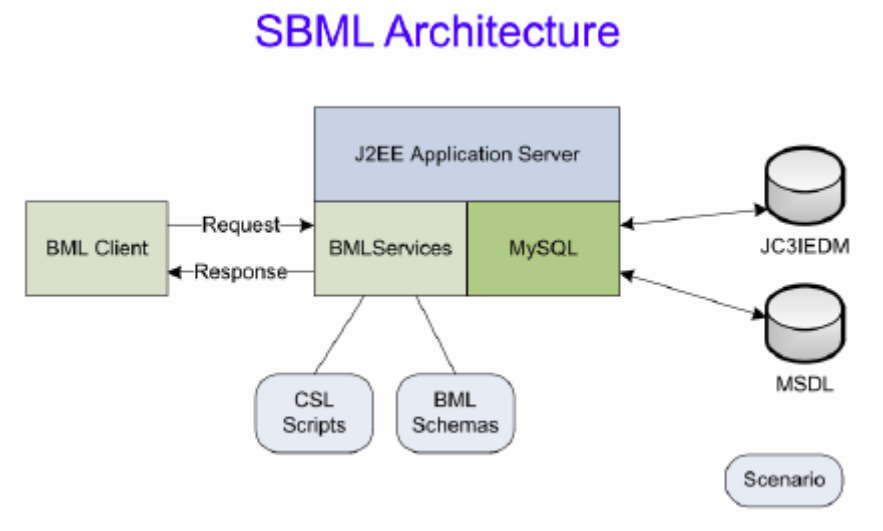

Imagen 4 - Arquitectura de SBML

Está basado en el concepto de Scripted Server donde los diferentes cambios que puedan sufrir BML o JC3IEDM pueden ser adaptados de manera rápida. Además provee una definición concisa de BML siguiendo mapeos de los modelos de datos. Por lo demás cumple todos los requisitos propuestos de diseño.

Su velocidad de procesamiento se encuentra entorno a las 10 transacciones por segundo.

2.1.3.2. Coalition Battle Management Services (CBMS)

Ha sido desarrollada por VMASC para JCW y permite servir documentos XML completos. Está basado en componentes Apache de código abierto y permite un alto rendimiento en torno a 100 transacciones por segundo.

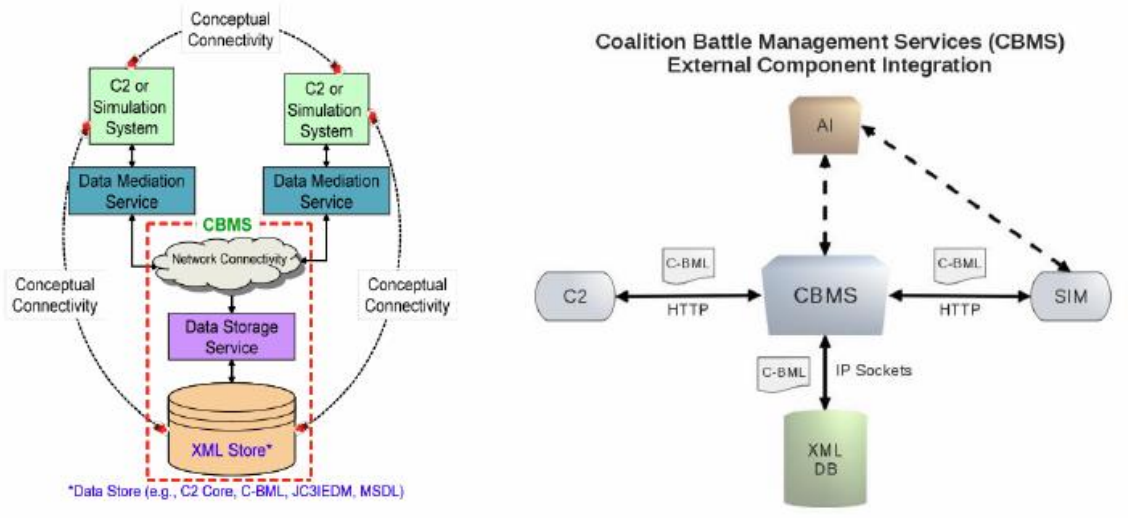

Imagen 5 - Modelo conceptual de CBMS y Arquitectura de CBMS

Los documentos XML que almacena no son validados a no ser que sea requerido. Su base de datos no es de tipo relacional y no divide sus datos en tablas sino que almacena los archivos XML de manera individual. Esto no evita que puedan usarse filtros para la búsqueda de un archivo concreto.

Se encuentra bajo licenciamiento OpenSource solo para USA, AUS, CAN, NZ, UK.

Cumple con la mayoría de requerimientos, incluyendo algunos como la posibilidad de subscripción. Permite el acceso mediante HTTP (en concreto xLightweb para el lado del cliente) y esto permite un acceso rápido a las misiones así como posibilita el acceso desde dispositivos móviles con acceso HTTP al servidor

2.1.3.3. FKIE BML server

Desarrollado por el Fraunhofer FKIE para su uso en experimentos de BML de Alemania y Francia. Implementa los esquemas básicos propuestos por el grupo MSG-048 pero no lo estándar de la SISO. Es de código abierto, pero solo se permite su uso en Alemania, Francia, Dinamarca, Holanda y España. Incluye uso de Namespaces, filtrado por temas (topics), Logs e interfaces SOAP y RESTful.

2.1.3.4. Saab WISE/SBML

Widely Integrated Systems Environment (WISE) es un software propietario de Saab para la integración de sistemas heterogéneos. Se trata de un servidor BML diseñado como transición de SBML. Saab está financiando a GMU para obtener los nuevos diseños.

Este sistema incrementa la integración con una suite de servicios de alto rendimiento. También existe una versión Lite usada para desarrollo y distribuida gratuitamente, incluyendo toda la actividad del MSG-085.

Entre otras novedades respecto a SBML incluye la posibilidad de usar servidores distribuidos.

2.2 Interfaces de Usuario - Graphical User Interfaces (GUIs)

2.2.1. Funciones principales

- Crear y editar documentos C-BML o MSDL posibilitando su serialización a XML.
- Validación de su gramática y sus esquemas.
- Envío o petición de un documento.
- Subscripción a un servidor para el envío automático siguiendo temas (topics).
- Obtención de los últimos reports.
- Mostrar mapas con la representación de la posición y geolocalización.

2.2.2. Interfaces de usuario en el mercado

2.2.2.1. C2LG GUI

Command & Control Lexical Grammar (C2LG) Graphical User Interface (GUI) está diseñado por el centro alemán de investigación FGAN (actualmente Fraunhofer FKIE). Usado principalmente para generar ordenes en sistemas JBML. Disponible en código abierto para los países que colaboran con el grupo MSG-085.

Incluye todas las características anteriores, exceptuando la total integración con C-BML y MSDL.

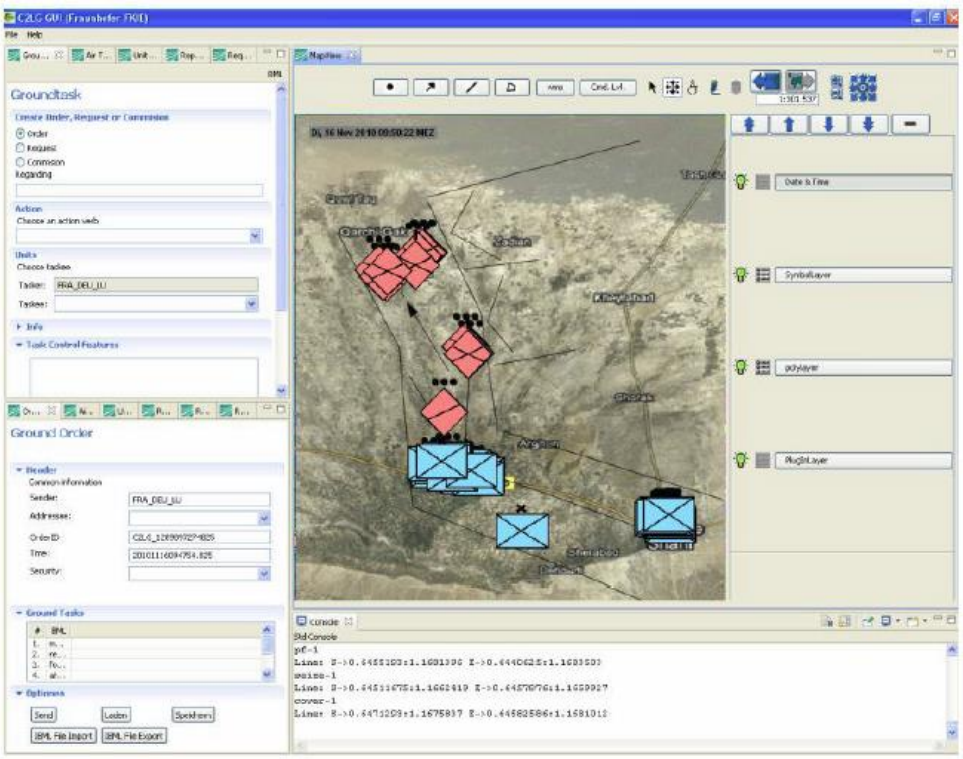

Imagen 6 - Interfaz gráfica de C2LG GUI

2.2.2.2. BML C2 GUI

Sigue el modelo de C2LG GUI y los consejos de Fraunhofer FKIE. Añade una visión completa como monitor y editor donde controlar ambas cosas de manera simultánea. Promete ser autoconfigurable siguiendo los XML Schemas del lenguaje, ya sea C-BML, MSDL, u otro sistema (posteriormente al hacerle el testeo se descubrirá que no es así). Permite dibujar iconos y marcar localizaciones sobre el mapa usando OpenMaps. Además, es de código abierto.

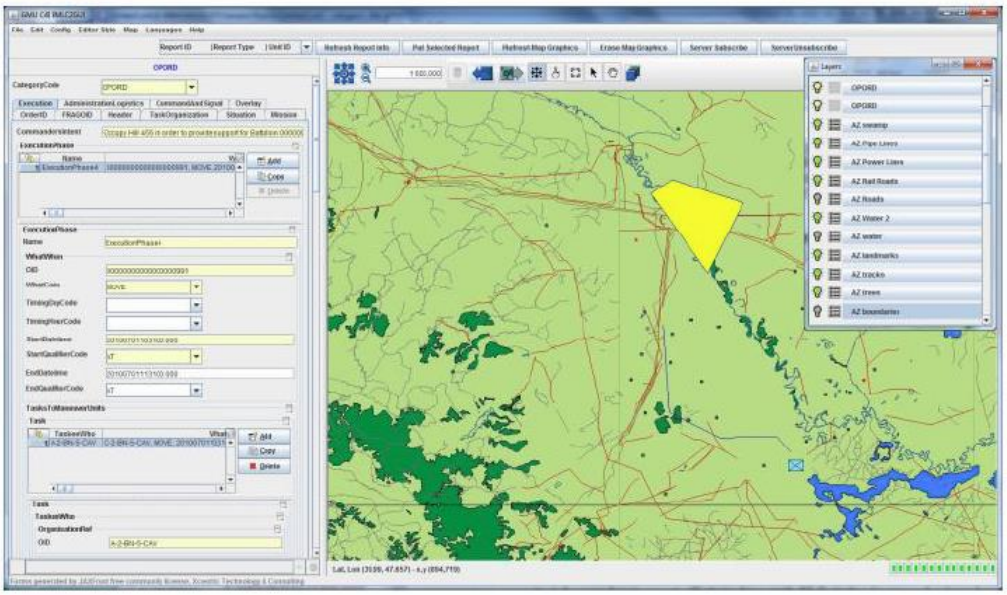

Imagen 7 - Interfaz gráfica de BML C2 GUI

3. CLIENTE

Se trata de una interfaz gráfica (GUI) donde es posible visualizar, abrir, crear y modificar las misiones de manera sencilla para el operador. También incluye un mapa donde consultar datos relacionados con posiciones geográficas. Una vez creada una misión es posible enviarla al servidor de misiones mediante un panel implementado.

3.1 Características básicas

En la siguiente imagen puede verse la interfaz gráfica que implementa el cliente y sus características básicas. El cliente ha sido programado en Java, usando las librerías JavaSwing para la creación de la interfaz.

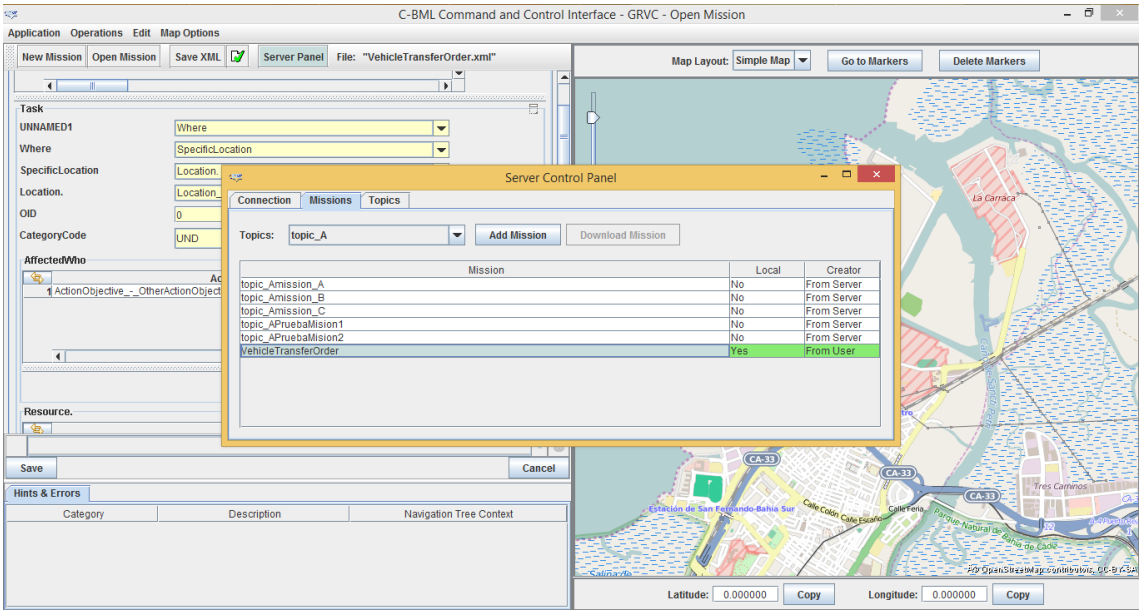

Imagen 8 - Partes básicas de la interfaz del cliente

La aplicación del cliente consta de 3 partes principales:

1) Zona para la creación, modificación y visualización de misiones C-BML

Se ha implementado la totalidad del protocolo C-BML, por lo que es posible generar *Order*, *Request* y *Report* de cualquier tipo. Estos mensajes C-BML pueden ser validados y exportados a XML para su posterior envío al servidor.

Para generar una nueva misión existe un formulario desarrollado gracias a la herramienta JaxFront.

2) Mapa donde consultar datos geográficos

Dispone de dos versiones del mapa: mapa simple y mapa geográfico. La tecnología utilizada para los mapas es OpenStreetMaps. Sobre el mapa es posible realizar marcas y conocer la Latitud y Longitud de cualquier punto.

3) Panel de control del Servidor.

Con este panel es posible controlar la conexión con el servidor, enviar y recibir misiones y subscribirse y eliminar subscripciones a los temas (topics). Además cuenta con un Log para conocer el estado del cliente. A su ver se ha implementado también la funcionalidad de pedir al servidor su Log, para de esta manera conocer todo lo que ocurre en el sistema.

3.2 Ejecución de la Interfaz

Para la ejecución de esta herramienta es necesario el uso de la máquina de java (JRE), con una versión igual o superior a la v8 (También llamada v1.8).

Para la ejecución de la interfaz solo es necesario hacer doble clic sobre el script de ejecución. Este será diferente para el sistema operativo Windows y Linux/Ubuntu.

- Windows

windows_exe.bat

- Linux/Ubuntu

linux_exe.sh

Estos scripts incluyen un arranque de la aplicación con una ampliación de la memoria virtual de la máquina Java. En Linux deberá añadir primero permisos de ejecución al archivo.

Algunos de los requisitos mínimos necesarios son:

- S.O. Windows, Linux, MAC OS
- **Máquina Java JRE v8**
- 2Gb de memoria RAM

IMPORTANTE: Es necesario el uso de la última versión de Java disponible para evitar errores inesperados.

Si no tiene instalada la última versión siga los pasos que aquí se indican para actualizarla.

Antes de instalar la última versión de java se recomienda desinstalar la versión que actualmente posea. Los pasos para instalar Java JREv8 en su PC son:

1) Windows. Descargar e instalar desde el siguiente enlace.

https://www.java.com/es/download

2) Linux. Seguir los siguientes pasos para instalar desde un terminal.

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer
$ sudo apt-get install oracle-java8-set-default
```

Para comprobar que la instalación es correcta, ejecutando el siguiente comando,

```
$ java –version
```

Debería aparecer lo siguiente:

```
java version "1.8.0_25"
Java(TM) SE Runtime Environment (build 1.8.0_25-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.25-b02, mixed mode)
```

3.3 Funcionalidades para la creación de misiones

La interfaz permite trabajar con archivos XML de misiones de C-BML según el formato definido por la SISO. La extensión de estos archivos es *".xml"*.

Las principales funcionalidades implementadas son:

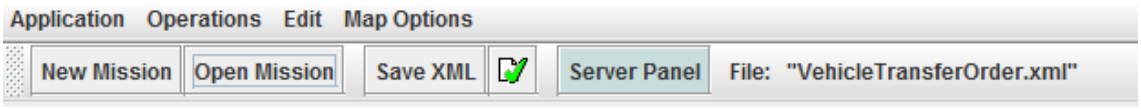

Imagen 9 - Botones para el manejo de las misiones

De izquierda a derecha,

1) Creación de nuevas misiones. Genera un lienzo en blanco donde crear la misión desde cero. *Nota: Los campos sombreados en naranja son aquellos obligatorios a cumplimentar.*
2) Abrir una misión. Permite abrir una misión desde un archivo XML. Solo aceptará misiones que sigan el estándar C-BML de la SISO. Esta funcionalidad es útil para hacer pequeñas modificaciones o ajustes sobre una misión existente.
3) Exportar Misión a XML. de esta manera se generará un mensaje C-BML en XML según el formato estándar C-BML de la SISO. Se recomienda "Validar" la misión antes de exportarla, comprobandose así la correcta sintaxis del mensaje creado.
4) Validar Misión. Comprueba que se siguen todos las limitaciones del estándar C-BML de la SISO. En caso de error, se indica el campo erroneo (recuadrándolo automáticamente en rojo) y la limitación que incumple (indicado en el bloque inferior) para ser corregido de manera sencilla.
5) Archivo Abierto. Aquí se muestra el nombre del archivo que se ha abierto. En caso de ser una nueva misión creada desde cero, no se mostrará nada.

Una vez abierta o creada una nueva misión, se podrán crear y modificar los diferentes campos que puede tener esta gracias a los menús desplegables que se crean.

Entre las opciones para crear las misiones se encuentran **todas las posibilidades descritas en el estándar C-BML**. Esto permite crear cualquier tipo de misión.

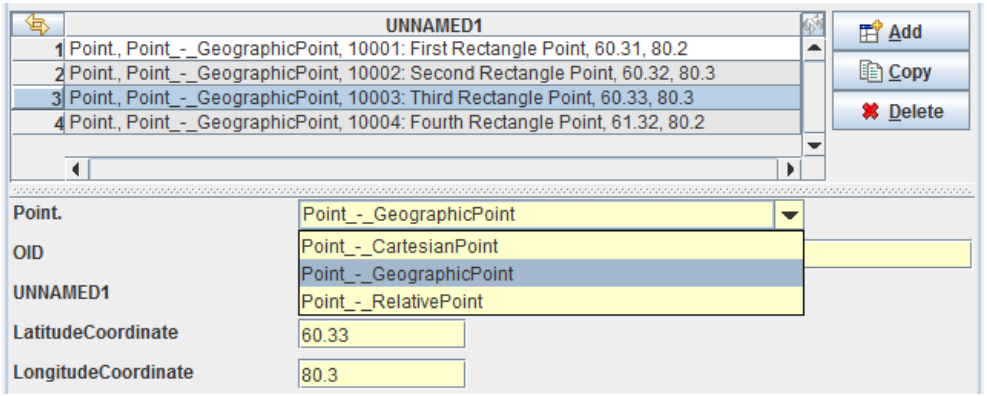

Imagen 10 - Diferentes opciones para crear un mensaje C-BML

Además, se incluye una ventana donde se notifican los posibles errores al validar la misión.

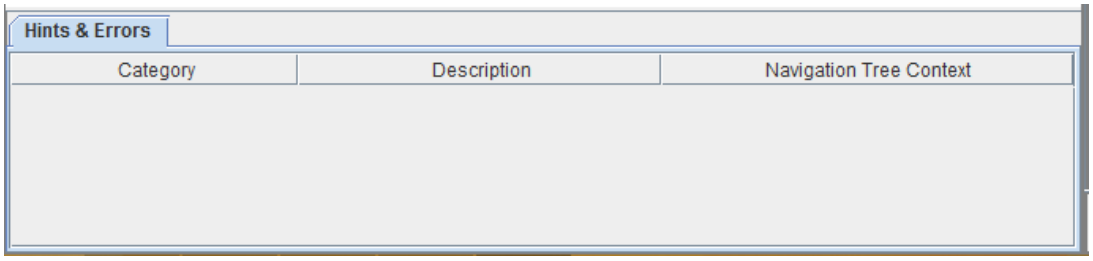

Imagen 11 - Panel donde se indican los posibles errores de formato

Para más información sobre la herramienta JaxFront, consultar el ***Error! Reference source not found.****A.*

3.4 Funcionalidades para el uso de mapas

Esta interfaz incluye un mapa con el que es posible consultar, fácilmente, posiciones geográficas.

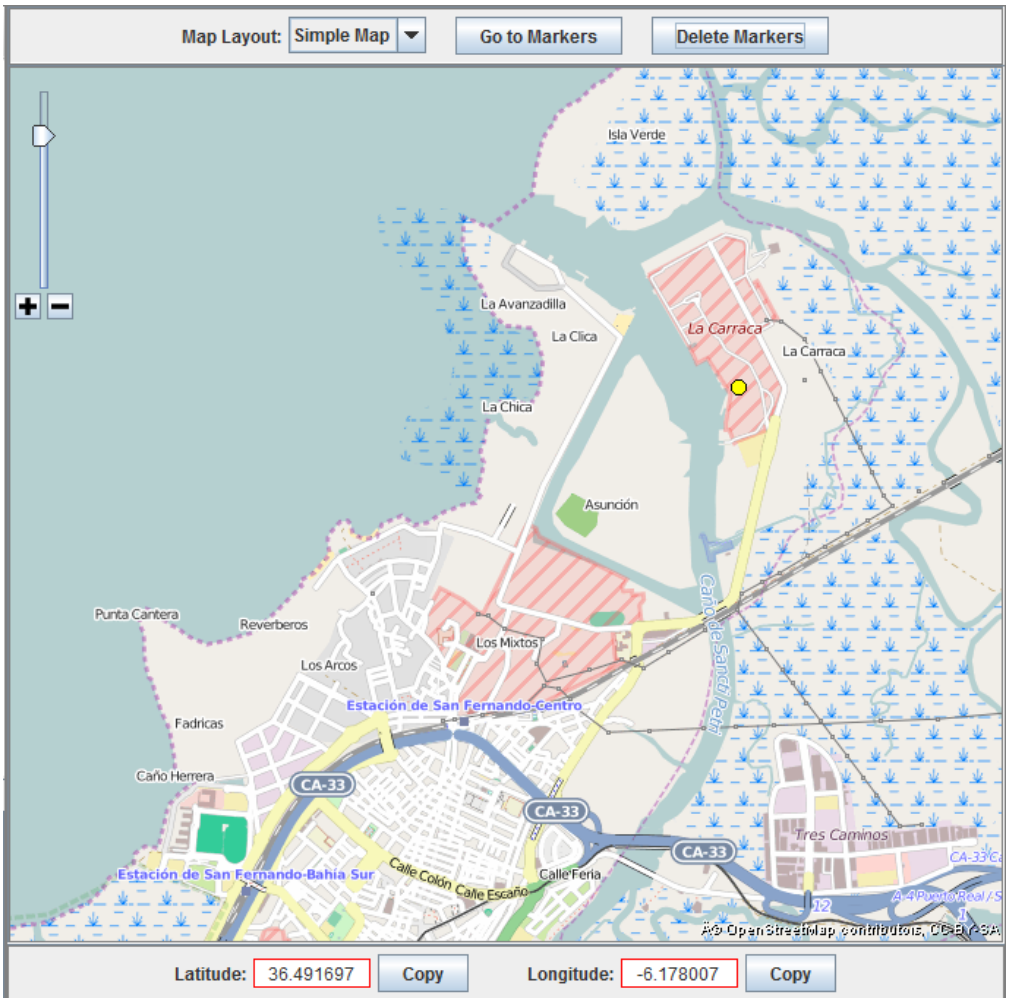

Imagen 12 - Panel de mapas incorporado en el cliente

Las funciones principales implementadas en esta versión son:

- Zoom. Posibilidad de hacer zoom sobre el mapa usando la rueda del ratón (scroll).
- Selección de puntos geográficos. Permite conocer la latitud y longitud de un determinado punto sobre el mapa. Al hacer doble clic con el ratón sobre un punto de este, aparecerá una marca amarilla indicando el punto seleccionado.

El punto seleccionado aparecerá en la parte inferior en los cuadros de texto enmarcados en rojo. Si se quiere añadir esta información al formulario de la misión, puede usarse la utilidad del botón "*Copy*" el cual colocará el valor en el portapapeles para su seguida inclusión en el citado formulario.

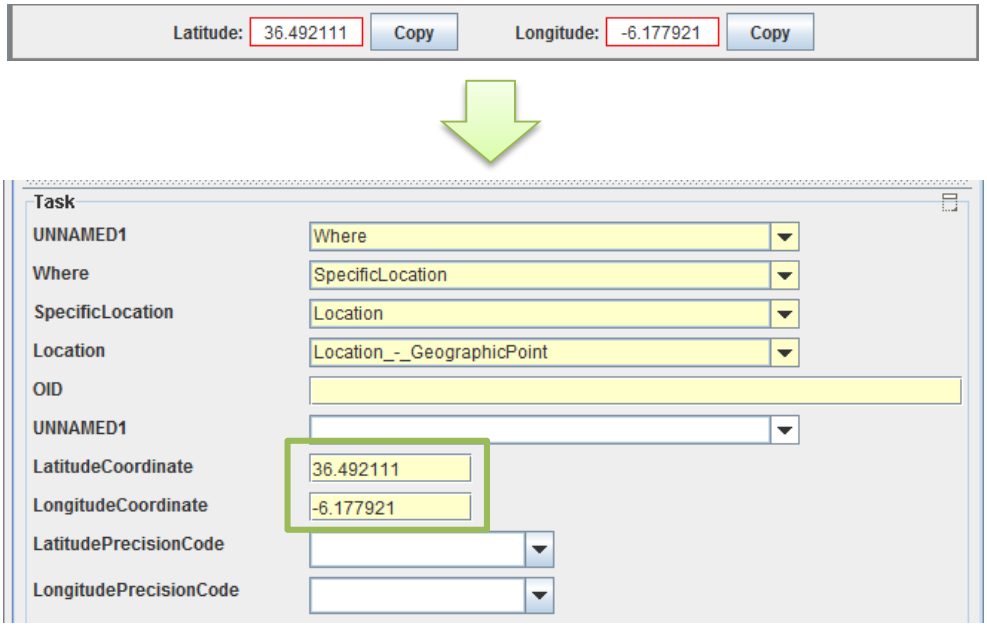

Imagen 13 - Funcionalidad de obtención de puntos geográficos para rellenar el formulario

Una vez seleccionados varios puntos en el mapa con las marcas amarillas, es posible focalizar el mapa en estos pulsando el botón superior "*Go to Markers*". En el caso de que se desee eliminar estas marcas, puede pulsarse sobre el botón "*Delete Markers*".

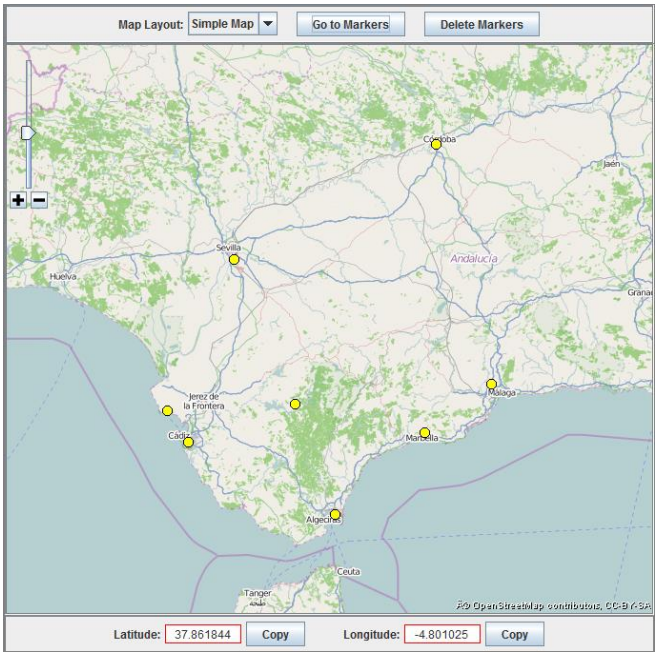

Imagen 14 - Funcionalidad para la selección de puntos en el mapa

- Map Layout. Sección donde se podrán escoger dos versiones diferentes del mapa.

Para obtener más opciones se incluye un apartado llamado "*Map Options*" en la parte superior izquierda, donde se pueden modificar parámetros sobre la visualización del mapa.

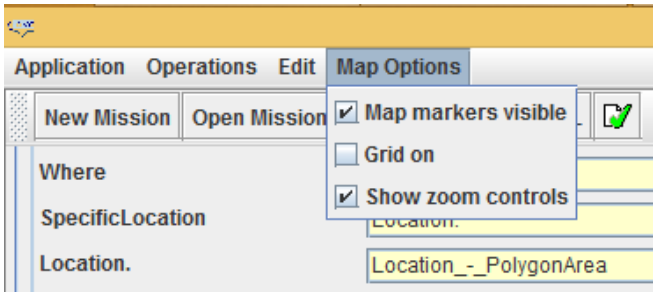

Imagen 15 - Panel secundario de opciones del mapa

Para más información sobre OpenStreetMaps, el funcionamiento de los mapas, los posibles errores de carga, etc. consultar el ***Error! Reference source not found.****B.*

3.5 Panel de control del Servidor

Para el control del servidor desde el cliente, se ha implementado una interfaz gráfica con la que realizar labores como la conexión, desconexión creación de temas, subscripción a estos, subida y descarga de misiones…

NOTA: El protocolo por el cual se estandariza el envío de estos datos no está estandarizado dentro del protocolo C-BML, por lo que desde el GRVC se ha diseñado un sistema de envío, el cual se detalla en el Apéndice C.

Para acceder a este panel, se ha añadido un botón a tal fin dentro de la barra de opciones:

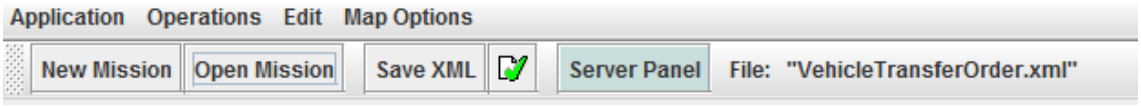

Imagen 16 - Botón Server Panel (sombreado)

Al pulsar, se abrirá el siguiente panel de control:

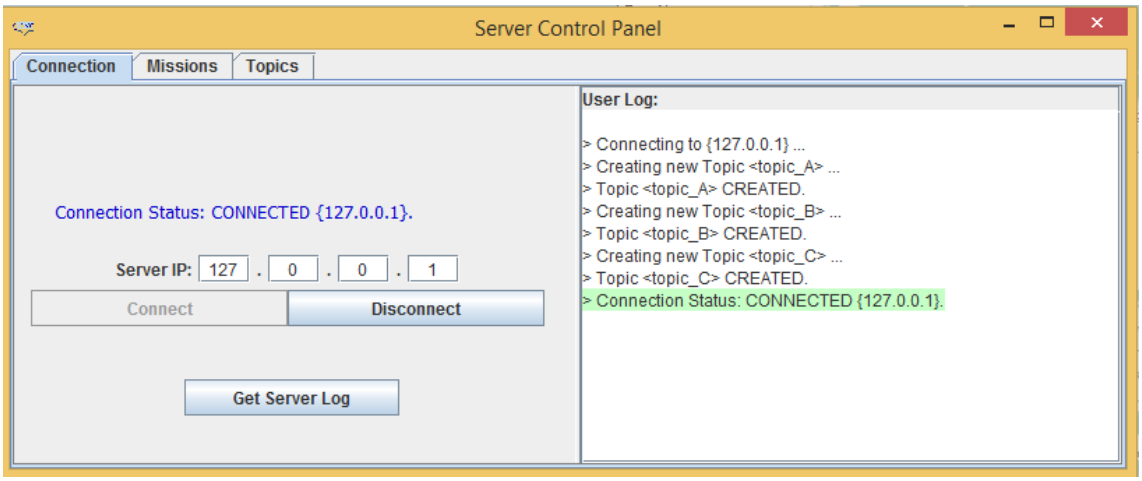

Imagen 17 - Opciones del panel de conexión con el servidor

3.5.1. Panel de conexión

- Estado de la conexión. Aquí se indicará si el cliente está conectado o no al servidor.
- Selección de IP. Aquí se indicará la IP donde ha de conectarse con el servidor y gracias al uso de los dos botones incluidos la conexión o desconexión se realizará automáticamente.
- Obtener el Log del Servidor. Con este botón se pide al servidor que envíe un log con el histórico de operaciones que se han llevado a cabo. De esta manera podrá supervisarse qué otros cliente hay conectados, qué temas se han creado en el servidor, etc.
- Log del cliente. Aquí se indicarán todas las operaciones relacionadas con el servidor que se lleven a cabo desde el panel de control.

3.5.2. Panel de temas

Las misiones han de ser publicadas en temas (topics). Gracias a este sistema te temas, otro cliente puede subscribirse a las misiones que sean de su interés. Algunos ejemplos de creación de temas son: un tema por cada contexto de ejecución, un tema por cada temática dentro de un contexto común, etc.

El panel mediante el cual se controlan estos mecanismos es el siguiente:

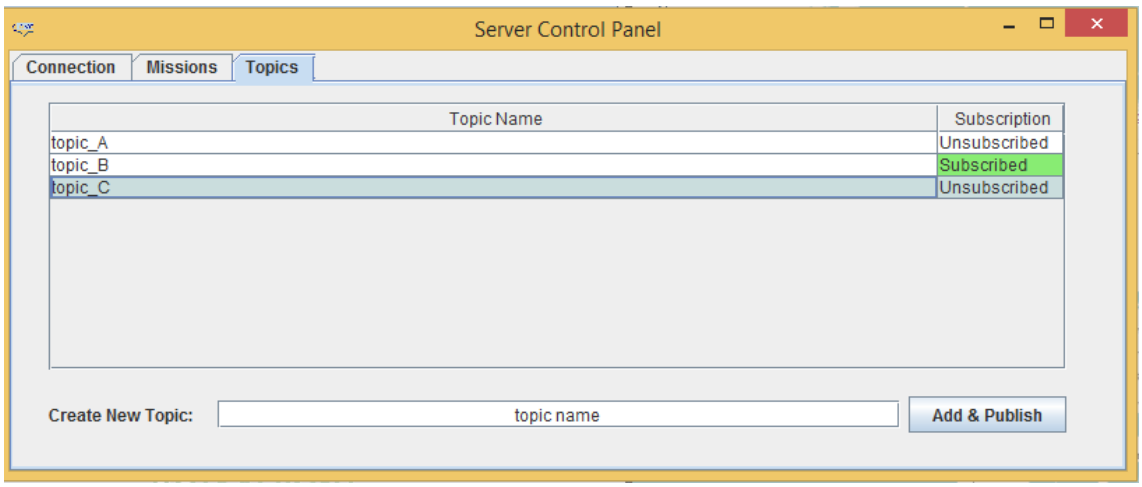

Imagen 18 - Opciones del panel de temas

Se pueden diferenciar las siguientes partes:

- Temas del sistema. En la parte central, se indican todos los temas que se han publicado en el servidor por todos los clientes de este. El usuario puede seleccionar que temas subscribirse o eliminar la subscripción haciendo doble clic sobre el tema deseado.
- Temas suscritos. Aquí se indica en color verde a qué temas está subscrito el cliente.
- Añadir un tema. Existe la posibilidad de crear nuevos temas en el servidor, independientemente de que el cliente desee subscribirse a él, es decir, es posible crear un tema y no llegar nunca a subscribirse a él. Esto puede tener sentido en ciertos contextos de coordinación.

3.5.3. Panel de misiones

En el panel de misiones es posible enviar al servidor las misiones creadas en el cliente y descargar del servidor las misiones que otros clientes han subido al servidor. Estas interacciones han de realizarse sobre un tema concreto, es decir, es posible añadir una nueva misión a un tema, o por lo contrario descargar una misión existente en un cierto tema.

Las características del panel de misiones son las siguientes:

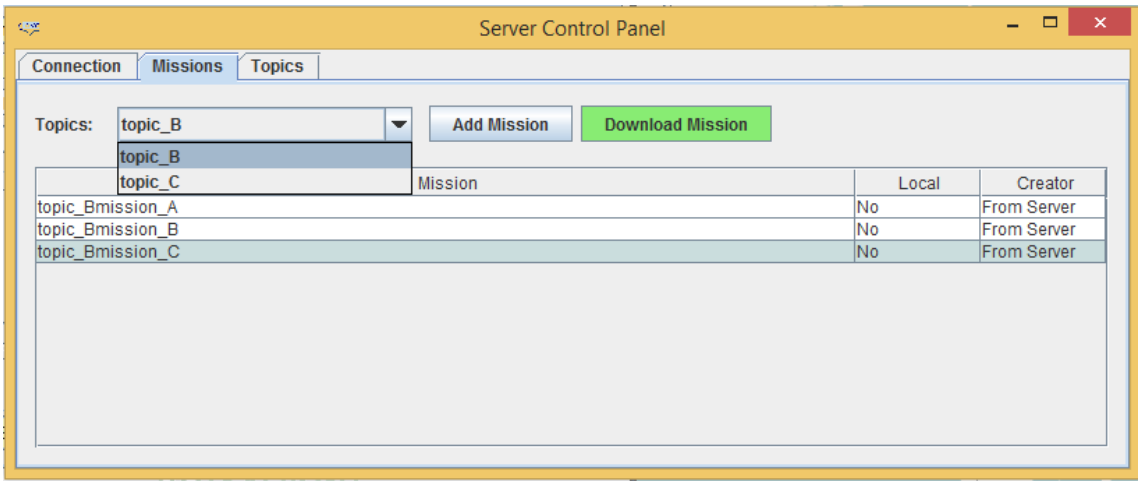

Imagen 19 - Opciones del panel de misiones

Se pueden diferenciar las siguientes partes:

- Selección de temas. En este selector es posible seleccionar el tema del que desean conocerse que misiones existen en el sistema. En él podrán incluirse también nuevas misiones. En este selector únicamente aparecerán las misiones a las que el cliente se encuentre subscrito, ya que para poder descargar y enviar misiones de un cierto tema es necesario haberse subscrito a él previamente.
- Misiones en el tema. Aquí se indica una lista de las misiones existentes en el tema escogido.
- Misiones en local. Aquí se indica si una cierta misión se encuentra en local. Por defecto solo las misiones añadidas por el cliente se encuentran en local. Para descargar a local una misión se usará la funcionalidad del botón “Download Mission”.
- Procedencia de la misión. Aquí se indica si la misión ha sido creada por otro cliente en el servidor (“From Server”) o por el contrario ha sido añadido por el mismo cliente (“From User”).
- Añadir una nueva misión. Mediante esta opción podrá añadirse una nueva misión al tema que se haya escogido. *Nota: Esta misión ha debido ser validada previamente, o sino el servidor la rechazará.*
- Descarga de una misión. Aquellas misiones que no estén en local es posible descargarlas usando este botón. En este caso se pedirá una localización donde guardar la misión descargada.

4. SERVIDOR

Se trata de un servidor socket PHP que gestiona todos los mensajes que se envían entre los diferentes clientes y el propio servidor. Para una mejor visualización de los procesos que se llevan a cabo, se ha desarrollado una interfaz web donde se muestran cada uno de los mensajes intercambiados entre cliente y servidor, una lista de los propios clientes conectados y el estado del servidor.

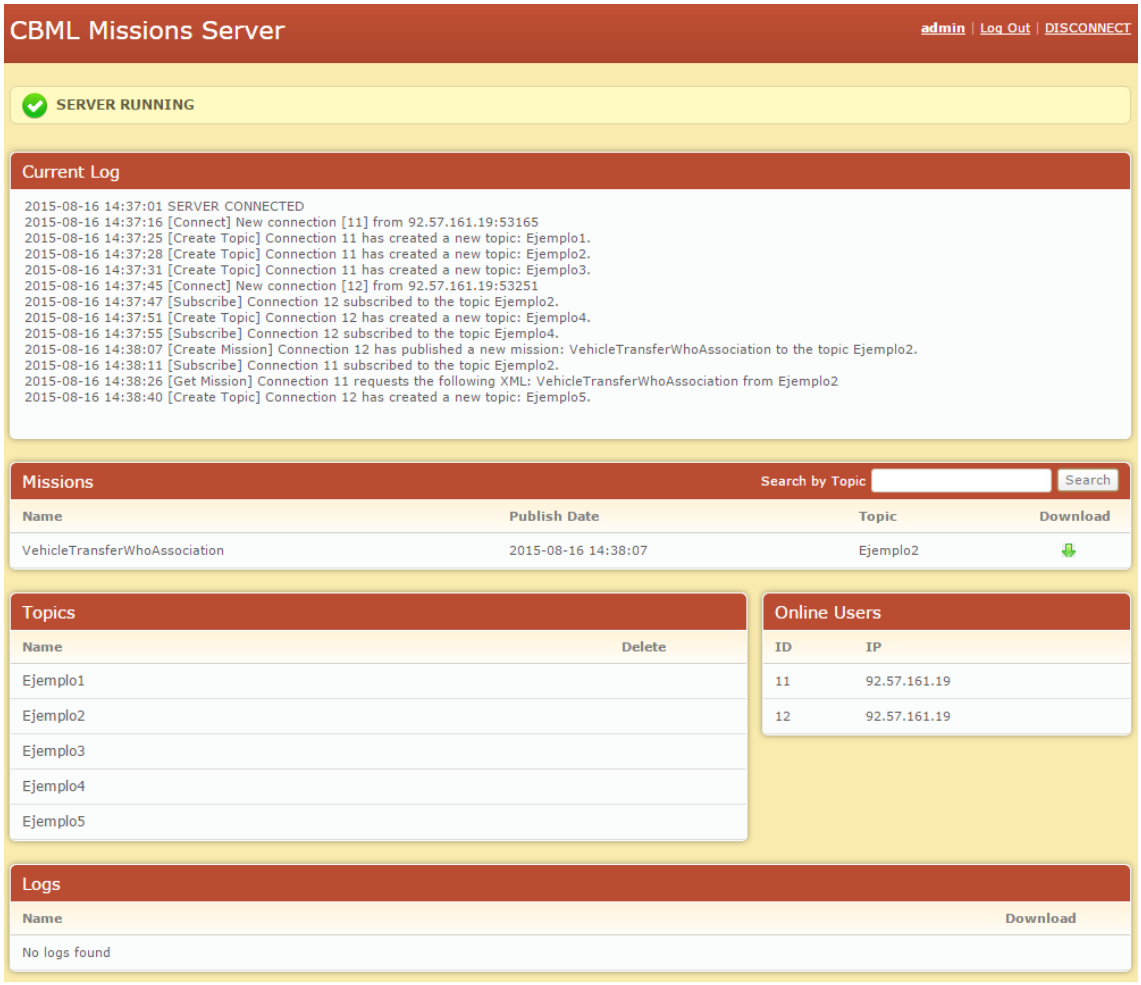

Imagen 20 - Interfaz web del servidor

Esta interfaz consta de las siguientes partes:

- Current Log. Aquí se observa, en tiempo real, los mensajes procesados por el servidor.
- Missions. Misiones publicadas en el servidor.
- Topics. Tópicos publicados en el servidor.
- Online Users. Usuarios conectados en el instante actual.
- Logs. Recopilación de logs creados por el servidor.

4.1 Características básicas

El servidor está programado en PHP y utiliza la librería *socket* propia del lenguaje para la comunicación con el cliente. Por otro lado, la interfaz donde se monitoriza el estado del servidor está en lenguaje web (HTML + CSS + JavaScript).

4.2 Ejecución del Servidor

Para la ejecución del servidor, es necesario tener instalado un sistema *LAMP* en el servidor. *LAMP* es el acrónimo utilizado para describir un sistema web que usa las siguientes herramientas:

- **L**inux, el sistema operativo.
- **A**pache, el servidor web.
- **M**ySQL, el gestor de base de datos.
- **P**HP, el lenguaje de programación.

Los pasos a seguir para instalar este sistema en el servidor se detallan a continuación.

4.2.1. Instalación de los servicios

4.2.1.1. Apache

Apache es un servidor de código abierto que corre en más del 50% de los servidores web del mundo. Para instalar *Apache*, desde un terminal, ejecutar lo siguiente:

```
$ sudo apt-get update
$ sudo apt-get install apache2
```

Para comprobar que *Apache* se ha instalado correctamente, escribir en el navegador la dirección IP del servidor (http://12.34.56.789, por ejemplo). La página debería mostrar el mensaje *"It works!".*

Para conocer la dirección IP del servidor,

```
$ ifconfig eth0 | grep inet | awk '{ print $2 }'
```

4.2.1.2. MySQL

MySQL es una potente herramienta para gestionar base de datos. Para instalar *MySQL* en el servidor, hay que ejecutar lo siguiente en un terminal:

```
$ sudo apt-get install mysql-server libapache2-mod-auth-mysql php5-mysql
```

Durante la instalación, *MySQL* preguntará para establecer una contraseña para el usuario *root*. Si por lo que fuera no pudiera establecerse en el momento o se quisiera cambiar en un futuro, puede hacerse desde la consola *MySQL*.

Una vez se ha instalado *MySQL*, debe activarse de la siguiente forma:

```
$ sudo mysql_install_db
```

Para terminar de configurar *MySQL*,

```
$ sudo /usr/bin/mysql_secure_installation
```

El script solicitará la contraseña del usuario root.

Después de esto, el script preguntará si se quiere cambiar la contraseña. Pulsar *N* (seguir con la misma contraseña) y continuar con los siguientes pasos. Para mantener la configuración recomendada, hay que pulsar *Y* en todas las opciones que nos muestra el script. Una vez hecho esto y finalizada la configuración, *MySQL* se reiniciará con los parámetros establecidos.

4.2.1.3. PHP

PHP es un lenguaje de código abierto adecuado para el desarrollo web. Para instalar *PHP* en el servidor, hay que ejecutar lo siguiente en un terminal:

```
$ sudo apt-get install php5 libapache2-mod-php5 php5-mcrypt
```

Una vez instalado *PHP*, se deben instalar los módulos estrictamente necesarios para la ejecución del servidor. Estos son:

- *php5-sockets*, para trabajar con sockets. (Depende de la versión viene ya por defecto instalado. En php5 no es necesario incluirla)
- *php5-mysql*, para trabajar con la base de datos.
- *php5-cli*, para ejecutar aplicaciones por terminal.

Para conocer los módulos *PHP* disponibles, desde un terminal escribir lo siguiente:

```
$ apt-cache search php5-
```

La terminal mostrará una lista de los posibles módulos disponibles.

```
php5-cgi - server-side, HTML-embedded scripting language
php5-cli - command-line interpreter for the php5 scripting language
php5-common - Common files for packages built from the php5 source
php5-curl - CURL module for php5
php5-dbg - Debug symbols for PHP5
php5-dev - Files for PHP5 module development
php5-gd - GD module for php5
php5-ldap - LDAP module for php5
php5-mysql - MySQL module for php5
php5-pgsql - PostgreSQL module for php5
php5-pspell - pspell module for php5
[...]
```

Para instalar un módulo concreto, ejecutar la siguiente instrucción:

```
$ sudo apt-get install nombre-del-modulo
```

Al finalizar la instalación de todos los servicios necesarios, ejecutar

```
$ sudo service apache2 restart
```

para reiniciar el servidor apache y que se apliquen todos los cambios realizados.

4.2.1.4. phpMyAdmin

phpMyAdmin es un software web de uso gratuito para trabajar con bases de datos *MySQL* en la web. Proporciona una interfaz web para administrar la base de datos.

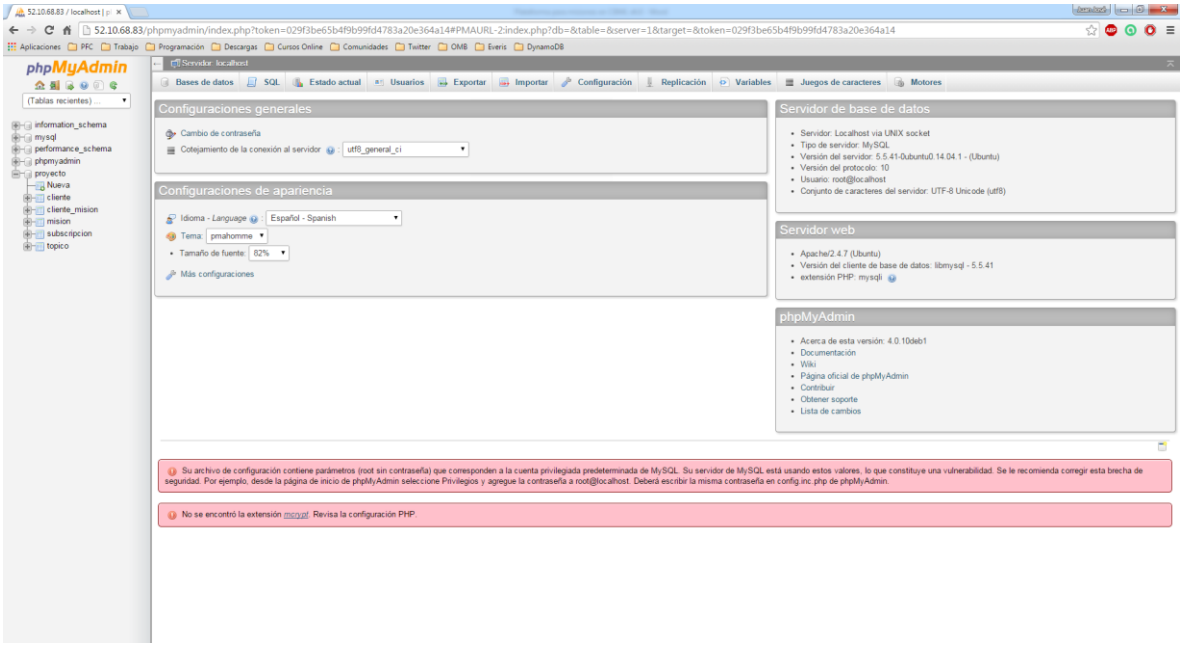

Imagen 21 - Interfaz de phpMyAdmin

Para instalar *phpMyAdmin*, ejecutar la siguiente instrucción desde un terminal:

```
$ sudo apt-get install phpmyadmin apache2-utils
```

Durante la instalación,

- Seleccionar *Apache2* para el servidor.
- Elegir *Yes* cuando se pregunte si se quiere configurar la base de datos con *dbconfig-common*.
- Introducir la contraseña *MySQL* cuando se solicite.
- Introducir la contraseña para *phpMyAdmin*.

Cuando finalice la instalación, ejecutar

```
$ sudo nano /etc/apache2/apache2.conf
```

para añadir *phpMyAdmin* a la configuración de *Apache*. Al final del archivo, escribir

```
Include /etc/phpmyadmin/apache.conf
```

Guardar y reiniciar nuevamente el servidor apache con la siguiente instrucción:

```
$ sudo service apache2 restart
```

Se puede acceder a la interfaz de *phpMyAdmin* desde el navegador a través de *http://direccion_ip/phpmyadmin*, siendo *direccion_ip* la dirección IP del servidor.

4.2.2. Base de datos

Una vez instalados todos los servicios, es el momento de importar la base de datos con la que trabajará el servidor *socket PHP*. Para ello, hay que seguir los siguientes pasos:

1) Acceder a *http://direccion_ip/phpmyadmin* desde el navegador web.
2) Loguearse con los datos definidos durante la instalación. Por defecto, el usuario es *root*.

3) Una vez dentro de *phpMyAdmin*, pulsar sobre Bases de datos (barra superior), introducir un nombre para la nueva base de datos y pulsar sobre Crear.

Imagen 22 - Crear una base de datos desde phpMyAdmin

Como se puede observar en la imagen, hay una base de datos con el nombre "proyecto", que es la base de datos que se ha creado para este caso. Una vez dentro de proyecto, se pulsa sobre la opción Importar.

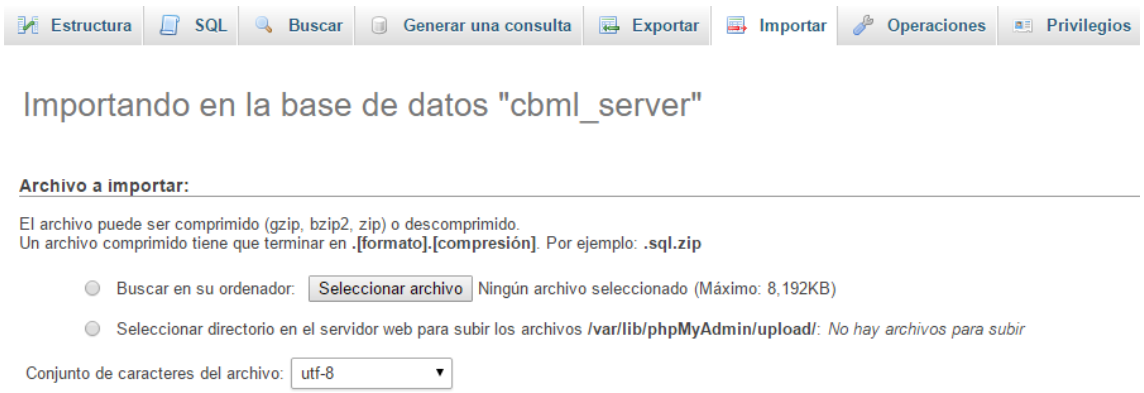

Imagen 23 - Importar un archivo SQL a phpMyAdmin

Se selecciona el archivo *proyecto.sql*, incluido en la documentación del proyecto, y se pulsa sobre Continuar. Una vez completado el proceso, la estructura de la base de daros se verá de la siguiente forma:

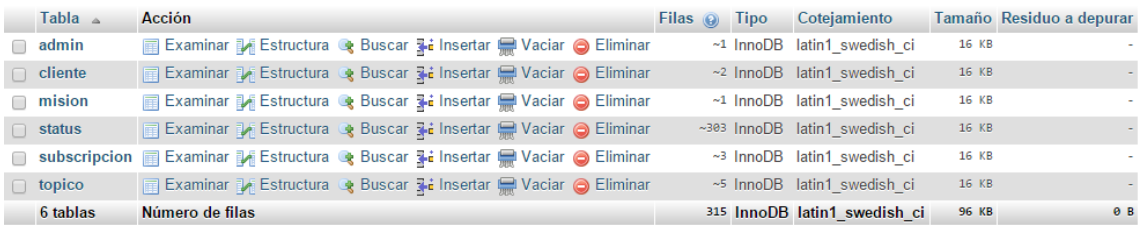

Imagen 24 - Estructura de la base de datos del servidor socket

4.2.3. Funcionamiento del Servidor

4.2.3.1. Estructura

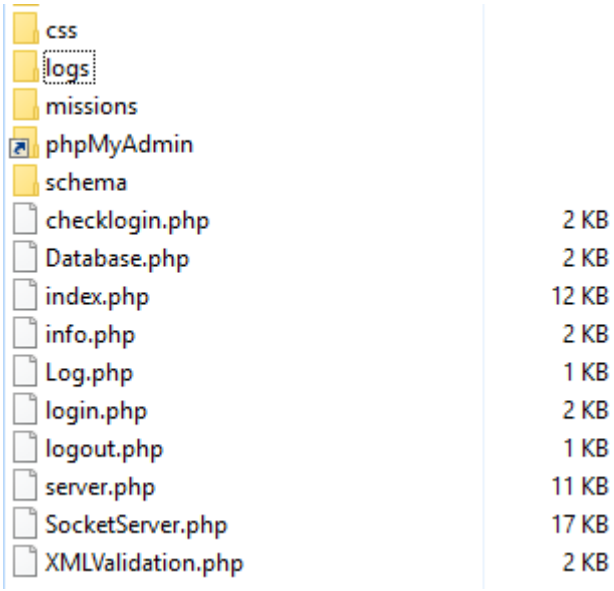

Imagen 25 - Estructura del servidor

Como se ha mencionado anteriormente, el servidor consta de 3 partes principales:

- Núcleo del servidor. Lógica necesaria para el funcionamiento propio del servidor. Este núcleo está formado por una serie de objetos y librerías programadas en PHP que controlan la ejecución del socket, la comunicación con la base de datos y la interfaz web.
- Interfaz web. Interfaz donde se monitoriza en tiempo real el estado del servidor.
- Base de datos MySQL. Sistema de almacenamiento del servidor.

Para un mejor entendimiento del servidor y su funcionamiento, en la imagen anterior se pueden ver los diferentes archivos que componen el núcleo del servidor. Entre todos ellos se prestará una atención especial al archivo **server.php** y a la librería **SocketServer.php**, donde se recoge toda la lógica en relación a los sockets.

Dentro del server.php, está definida la siguiente estructura:

```
$log = new Logging();
$name = "log_".date("Ymd_Hms").".txt";
$log->lfile('logs/'.$name);
$log->lwrite('SERVER CONNECTED');
noQuery("INSERT INTO status (status, fecha_accion) VALUES (1, '".date("Y-
m-d H:i:s")."')");
$motd = utf8_encode("<##connected##>\n");
$server = new SocketServer(AF_INET, SOCK_STREAM, SOL_TCP);
$server ->bind('0.0.0.0', 9300)
        ->setMotd($motd)
        ->setRequestHandler('handler')
        ->setOnOpenHandler('open_handler')
        ->setOnCleanupHandler('cleanup_handler')
        ->setOnCloseHandler('close_handler')
        ->setOnWriteErrorHandler('write_error_handler')
        ->run();
exit(0);
```

El funcionamiento es el que sigue:

1) En primer lugar, cuando se arranca el servidor, se crea log de nombre *"name"* haciendo uso de la clase Logging. En este log se recopilará toda la información procesada por el servidor desde este momento hasta su desconexión.
2) Se inserta en la tabla *status* de la base de datos un registro con el campo status a 1.

Esta tabla almacena la información de conexión/desconexión del servidor. Cuando un cliente hace una petición de conexión al servidor, se comprueba el último registro dentro de esta tabla. Si el valor de status en este registro es de 1, el cliente se puede conectar sin ningún inconveniente; si el valor es 0, el cliente asume que el servidor esta desconectado y no se procesa la conexión.

3) Una vez creado el log y el registro de conexión en la base de datos, es momento de arrancar el servidor como tal. Para ello, se definen las siguientes propiedades:
    a. *bind('0.0.0.0', 9300).* El servidor aceptará peticiones de conexión realizadas desde cualquier dirección web a través del puerto 9300.
    b. *setMotd($motd).* Mensaje de bienvenida del servidor en el momento de la conexión, es decir, cuando un cliente consigue conectarse al servidor, este será el mensaje de respuesta que reciba.
    c. *setRequestHandler('handler')*. Método que se ejecutará cuando se reciba una petición del cliente una vez establecida la conexión.
    d. *setOnOpenHandler('open_handler')*. Método que se ejecutará cuando se reciba una petición de conexión del cliente.
    e. *setCleanupHandler('cleanup_handler')*. Método que se ejecutará para liberar una conexión con el cliente una vez cerrada.
    f. *setOnCloseHandler('close_handler')*. Método que se ejecutará cuando se reciba una petición de desconexión del cliente.
    g. setOnWriteErrorHandler('write_error_handler'). Método para el control de errores.
    h. run(). Bucle principal de ejecución.

Una vez ejecutadas estas instrucciones, el servidor estará operativo para recibir peticiones y procesarlas de cuantos clientes estén conectados. A modo de ejemplo, se detalla a continuación los pasos que sigue el servidor cuando recibe una petición para la creación de un nuevo tópico.

- Como se ha visto anteriormente, el método encargado de procesar las peticiones de los clientes una vez establecida la conexión es *setRequestHandler('handler')*. El código de dicho método es el que sigue:

```
function handler($request, $id) {
 global $log;

 $broadclient = array();
 $broadcast = false;
   if ($request == "<##release##>") {
       delClient($id);
       return null;
 }

 $msc = microtime(true);
 $request_aux = substr(substr($request, 3), 0, -3);
 $action = explode(".", $request_aux, 2);
 switch($action[0]) {
       case 'createtopic':
             $response = createTopic($id, $action[1]);
             $broadcast = ($response == "") ? false : true;
             break;
       case 'subscribe':
             $response = subscribe($id, $action[1]);
```

```
                break;
        case 'unsubscribe':
                $response = unsubscribe($id, $action[1]);
                break;
        case 'createmission':
                list($response, $broadclient) = createMission($id,
$action[1], $broadclient);
                $broadcast = ($response == "") ? false : true;
                break;
        case 'getmission':
                $response = getMission($id, $action[1]);
                break;
        case 'serverlog':
                $response = serverLog($id, $action[1]);
                break;
        case 'disconnect':
                $response = "<##ack.disconnect##>";
                $log->lwrite("[Disconnect] Disconnect request from
connection $id");
                break;
        case 'shutdown':
                $response = false;
                shutdown($id);
                break;
        default:
                $response = "NOT_ACK";
                break;
 }

 if($request_aux != "alive?") {
        $msc = microtime(true) - $msc;
        // echo sprintf('Received: "%s"', $request_aux) . PHP_EOL;
        // echo sprintf('Sent: "%s"', $response) . PHP_EOL;
        // echo sprintf('Time: %d milliseconds', $msc*1000) . PHP_EOL;
 }

    return array($broadcast ,$response, $broadclient);
}
```

- Cuando llega una petición del cliente $request, hay que conocer el tipo.

Para más información sobre los tipos de peticiones, consultar el ***Error! Reference source not found.****C.*

- Una vez identificado el tipo de petición y el mensaje (si la petición lo requiere), el servidor ejecuta un procedimiento u otro. El servidor, en este caso de ejemplo, recibe una petición *<##createtopic.nombretopic##>*, donde *createtopic* es el tipo de petición, y *nombretopic* es el nombre que se le dará a ese nuevo tópico.
- Para este tipo de peticiones, el método encargado del procesamiento del mensaje es *createTopic($id, $message),* siendo *$id* la id de la conexión, *$message* el nombre del nuevo tópico.
- El servidor inserta el nuevo tópico en la base de datos

```
noQuery("INSERT INTO topico (topico) VALUES ('".$message."')");
```

- Y escribe en el log el mensaje definido para este tipo de peticiones

```
$log->lwrite("[Create Topic] Connection $id has created a new topic:
$message.");
```

La última parte del núcleo del servidor que merece ser comentada es el bucle principal, dentro del método run(), definido en la librería SocketServer.php. Su lógica se puede dividir en dos estados:

1) Servidor conectado (acepta conexiones).

```
foreach($pool as $conn_id => $conn) {
  if(!is_resource($conn)) {
    $this->__cleanup($conn_id);
    $this->__close($conn_id);
    unset($pool[$conn_id]);
   }
}

// Create a copy of pool for socket_select()
$active = $pool;

// Halt execution if socket_select() failed
if(socket_select($active, $w, $e, null) === false) {
  $this->__raiseError();
}

// Register new client in the pool
if(in_array($this->__socket, $active)) {
  $conn = socket_accept($this->__socket);
  if(is_resource($conn)) {
    $conn_id = (integer)$conn;
    $response = $this->__open($conn_id, $conn);
    if ($response) {
      if($response != 1)
        socket_write($conn, $response, strlen($response));
      $pool[$conn_id] = $conn;
      if($this->__motd !== null)
        @socket_write($conn, $this->__motd, strlen($this->__motd));
    }
    else {
      $this->__close($conn_id);
      @socket_close($conn);
    }
  }

  unset($active[array_search($this->__socket, $active)]);
}
```

Se diferencian varias partes dentro de la lógica de conexión:

- En primer lugar, se liberan las conexiones que han dejado de funcionar, ya sea por un error de comunicación y/o por desconexión del cliente.
- El servidor, entonces, se queda en “suspensión” esperando por una conexión. Cuando esta conexión llega y es correcta, se procesa (se crea la conexión, se le asigna una id y se agrega al array de conexiones existentes) y se le responde con el mensaje de bienvenida. Cuando la conexión no es la adecuada, se omite y se vuelve a la sentencia de espera de nueva conexión.

Una vez hay clientes conectados en el servidor, se pasa a un nuevo bloque de código que será el encargado de procesar las diferentes peticiones que llegue por cada cliente.

```
$conn_id = (integer)$conn;
$request = @socket_read($conn, $this->__readAmount, $this->__readMode);
$request_max = $request;

while(strlen($request_max) == 1460) {
  $request_max = @socket_read($conn, $this->__readAmount, $this-
>__readMode);
  $request .= $request_max;
}

// If connection is closed, mark it for clean and continue
if($request === false) {
  $pool[$conn_id] = false;
  continue;
}

$request  = trim($request);

// Skip to next if client tells nothing
if(strlen($request) == 0)
  continue;

list($broadcast, $response, $clients) = call_user_func($this->__handler,
$request, $conn_id);
```

El funcionamiento de las líneas que aparecen arriba es el siguiente:

- Por cada cliente conectado, se mira si hay peticiones pendientes en el buffer.
- Si no hay nada pendiente, se acaba la ejecución para dicho cliente y se pasa al siguiente; de haber algo en el buffer, se procesa. Para ello, se lee la petición y se llama a la función que procesa todas las peticiones que llegan al servidor de clientes ya conectados, detallada con anterioridad.
- Esta función realiza la operativa correspondiente a cada tipo de petición y devuelve la respuesta a este bucle. En función a dicha respuesta, se actúa de una forma u otra. Estas son:
    - Si la respuesta es igual a *null*, se cierra la conexión, pues es la respuesta que se espera por parte del servidor cuando un cliente solicita la desconexión del mismo.

```
// Request handler asks to close connection
if($response === null) {
  @socket_close($conn);
  $this->__close($conn_id);

  unset($pool[$conn_id]);
  continue;
}
```

    - Si la respuesta es igual a false, significa que el servidor está procesando su desconexión, por lo que se cierran todas las conexiones existentes y se le envía a los clientes el mensaje <##shutdown##>.

- Para cualquier otro tipo de respuesta diferente de “NOT_ACK” (esta es la respuesta a los mensajes de isAlive por parte del cliente, que no se responden), éste se procesa de forma diferente si es de tipo broadcast (se envía a todos los clientes conectados o a ciertos clientes seleccionados), o de forma individual.

```
else if($response != "NOT_ACK") {
  $response = utf8_encode($response."\n");
  if($broadcast == FALSE)
    @socket_write($conn, $response, strlen($response));
  else {
    // Tell subscribed clients to the topic the new mission
    if(!empty($clients)) {
      foreach($pool as $broadname) {
        $id = (integer)$broadname;
        if(in_array($id, $clients))
          @socket_write($broadname, $response, strlen($response));
      }
    }
    // Tell everyone the new topic
    else {
      foreach($pool as $broadname) {
        @socket_write($broadname, $response, strlen($response));
      }
    }
  }
}
```

2) Servidor desconectado (no acepta conexiones).

```
$i = 0; $closed = $pool;
foreach($closed as $conn) {
  if($i != 0) {
    @socket_close($conn);
    $this->__close((integer)$conn);
   }

   $i++;
}

$pool = $pool_ini;
$this->__close("DISCONNECT");
while(!$this->__checkstatus()) {
 sleep(2);
}

$this->__open("CONNECT");
```

Cuando se pulsa sobre el botón de desconexión del servidor en la interfaz, éste cierra y libera todas las conexiones existentes, escribe en el log el mensaje definido para esta acción y se queda en un bucle en el que cada x tiempo (en este caso ha sido programado en 2 segundos) comprueba el estado de conexión. Cuando el administrador vuelve a conectar el servidor, éste sale del bucle y continúa con su funcionamiento normal.

4.2.3.2. Ejecución del servidor

Una vez instalados todos los servicios y teniendo una visión general de cómo funciona el servidor, el siguiente paso es hacerlo andar. Para ello, basta con dirigirse al directorio del servidor y ejecutar la siguiente línea por consola:

---

php server.php

4.2.4. Interfaz Web

Al principio de este capítulo se ha visto una captura de la interfaz web que controla el servidor. En este apartado se profundizará desde un punto de vista técnico en su funcionamiento.

4.2.4.1. Login

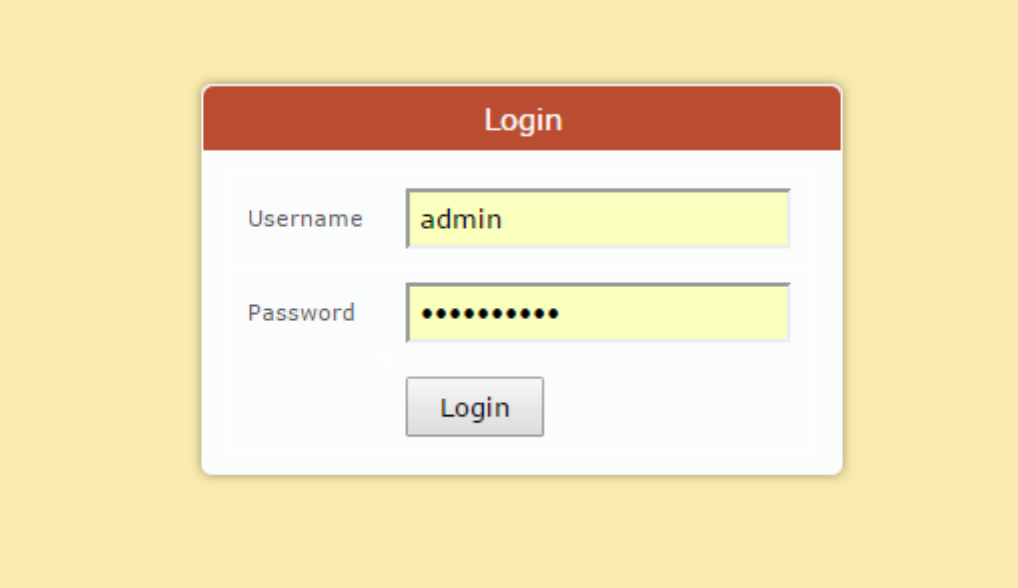

Imagen 26 - Login

Esta es la pantalla que se muestra la primera vez que se accede a la interfaz. Los parámetros de conexión del administrador están definidos en la tabla *admin* de la base de datos. Por defecto,

- Usuario: admin
- Contraseña: cbmlserver

4.2.4.2. Barra de herramientas

Una vez dentro de la interfaz, en el menú superior se encuentran las siguientes opciones:

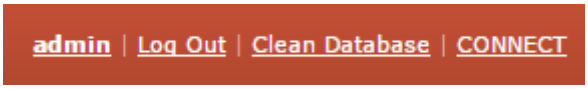

Imagen 27 - Menú superior de la interfaz

- admin. Nombre del usuario conectado.
- Log Out. Botón para cerrar la sesión actual.
- Clean Database. Este botón, por seguridad, solo estará activo cuando el servidor esté desconectado. Su función es como su nombre indica borrar la base datos al completo. Las únicas tablas que permanecen intactas son *admin* y *status*.
- CONNECT/DISCONNECT. Conectar/Desconectar el servidor.

4.2.4.3. Información

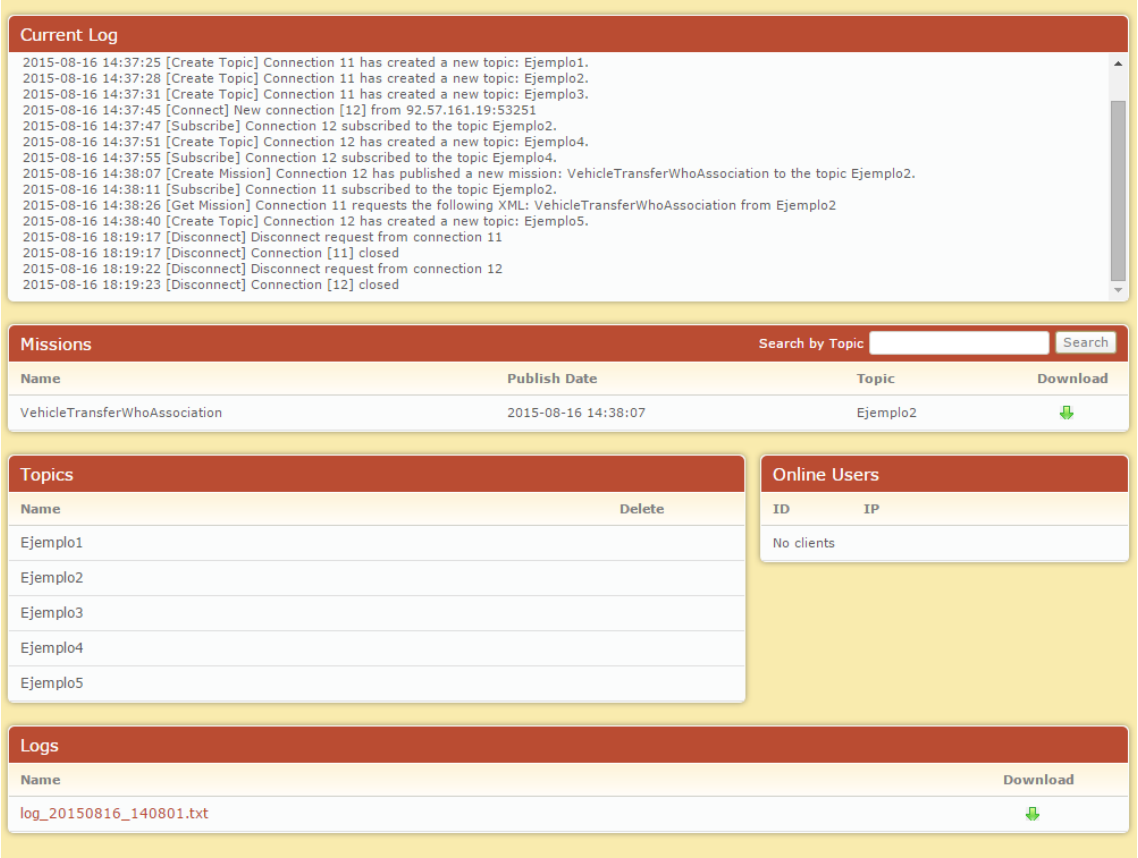

Imagen 28 – Opciones de la interfaz web

Se pueden destacar varios apartados dentro de la interfaz web.

- Current Log. En el log del servidor se puede seguir en tiempo real las instrucciones que están siendo ejecutadas en el servidor, es decir, las peticiones que está intercambiando con los diferentes clientes conectados. En cada desconexión del servidor se cierra el log sobre el que se está trabajando y se crea uno nuevo cuando se vuelve a conectar. Para que esto se produzca, la desconexión tiene que ser causada por una caída del servidor a nivel de servicio, no a nivel de usuario.
- Missions. En este apartado se pueden visualizar todas las misiones publicadas en el servidor en la sesión actual. Esta tabla ofrece la opción de filtrar las misiones por tópicos y descargar la misión que se necesite.
- Topics. Tópicos publicados en la sesión actual. En la columna *delete* aparecerá un botón para eliminar el tópico en cuestión siempre que el servidor esté desconectado, para así no alterar el funcionamiento del mismo.
- Online Users. Clientes conectados.
- Logs. Aquí aparecen todos los logs que ha ido creando el servidor en las diferentes sesiones que han sido creadas. Como se ha dicho en un apartado anterior, estas desconexiones no son las que el usuario fuerza desde la interfaz web, sino caídas de servicio a nivel de aplicación. Al igual que las misiones, estos pueden ser descargados.

5. CONCLUSIONES Y LÍNEAS DE DESARROLLO FUTURAS

5.1 Conclusiones

La elaboración de este Proyecto Fin de Carrera ha supuesto un coste apreciable en tiempo que ofrece alternativas a los protocolos de comunicaciones presentes en el mercado en la actualidad. Se ha desarrollado una plataforma intuitiva y de fácil instalación que reúne las principales características de los sistemas desarrollados por la competencia.

El servidor de misiones CBML desarrollado en este proyecto está compuesto por un cliente Java y un servidor PHP, dos de los lenguajes más extendidos y consolidados en el mercado, compatibles en la mayoría de los sistemas informáticos. A su vez, estos lenguajes tienen excelentes librerías para el manejo de archivos XMLs, principal elemento de información para este tipo de plataformas.

Haciendo referencia al segundo apartado de este documento, un servidor de misiones CBML debe cumplir con las siguientes especificaciones:

- Brindar la posibilidad de postear nuevos documentos XML: Order y Report en C-BML y escenarios en MSDL.
- Aceptar subscripciones a un cierto tema (Topic).
- Responder a peticiones de archivos llevada a cabo por clientes.

Así pues, se ha elaborado un sistema de comunicaciones siguiendo las especificaciones marcadas por los principales organismos reguladores y cubriendo, a su vez, las pautas marcadas por el cliente. Este sistema tiene las 3 características mencionadas anteriormente, además de una serie de mejoras (como la interfaz web) que facilitan el funcionamiento y entendimiento del mismo.

El servidor de misiones CBML aquí desarrollado está capacitado para procesar y almacenar una gran cantidad de peticiones desde el momento de su instalación, sin necesidad de sistemas y plataformas adicionales más allá de las mencionadas en este documento técnico.

5.2 Líneas de desarrollo futuras

Durante el diseño del cliente y del servidor se han incorporado todas aquellas funcionalidades y mejoras para cumplir los requisitos mínimos de diseño y además aquellas que se han creído conveniente para una experiencia de usuario mejorada.

Sin embargo, siempre es posible mejorar el diseño actual. Teniendo en cuenta otros diseños de plataformas parecidas y recomendaciones dadas por la SISO o la OTAN, se proponen las siguientes mejoras:

1) Compresión de los archivos antes del envío de estos.

Mediante esta funcionalidad podría disminuirse el consumo de ancho de banda que puedan producir el envío de mensajes XML completos. Para ello, se recomienda el uso de algún compresor "XML-concious", que son aquellos que tienen en cuenta la estructura de los archivos XML para optimizar su compresión.

2) Uso de MSDL como lenguaje para la inicialización del sistema.

MSDL permite determinar un estado inicial para una misión de C-BML, con lo que se podría mejorar el proceso de inicialización del servidor.

3) Sistema de servidores distribuidos.

Actualmente el uso de un solo servidor centralizado hace que este sea un elemento crítico en la red. Mediante el uso de una arquitectura de servidor distribuida podría ganarse robusted ante posibles incidencias en el servidor principal.

4) Añadir inteligencia artificial al servidor.

Si el servidor interpretase el contenido de todas las misiones que aloja, podría generar otro tipo de servicios añadidos a los actuales. Para ello necesitaría unos mecanismos de algoritmia nada sencillos, con los que aplicar mecanismos de análisis del contexto de las misiones.

5) Seguridad en el paso de mensajes.

La codificación que se utiliza actualmente en el paso de mensajes entre el servidor y los clientes es muy sencilla e intuitiva para cualquier usuario con un mínimo de conocimientos técnicos. En un futuro se podría añadir un sistema de claves (tokens) al que sólo tuvieran acceso los clientes autorizados o una codificación en el propio cuerpo del mensaje.

Bibliografía

[1] Dr. J. Mark Pullen, Dr. Michael R. Hieb, Dr. Stan Levine, Dr. Andreas Tolk y Curtis Blais, "*Euro Simulation Interoperability Workshop"*, 2007

[2] Dr. Mark Pullen, Dr. Stanley Levine, Nico de Reus, Nanne Le Grand, Paul de Krom, Dr. Kevin Heffner, Ole Martin Mevassvik, Anders Alstad, Lionel Khimeche, Ricardo Gomez-Veiga, Sergio Galan Cubero, Dr. Ulrich Schade, Dr. Miloslaw Frey y Adam Brook, "*IEEE/SISO Simulation Interoperability Workshop"*, 2010

[3] Dr. Mark Pullen, Douglas Corner, Samuel Singapogu, Bhargava Bulusu y Mohammad Ababneh, "*Proceedings of the IEEE 2010 European Simulation Interoperability Workshop*", 2010

[4] Mohammad Ababneh y Dr. J. Mark Pullen, "*Proceedings of the IEEE Fall 2010 Simulation Interoperability Workshop"*, 2010

[5] Dr. J. Mark Pullen, Douglas Corner y Lisa Nicklas "*16th CCRTS"*, 2011

[6] Alfredo Vázquez Reyes, *"Ideas para la colaboración entre Sistemas C-BML"*, 2015

APÉNDICE A. HERRAMIENTA JAXFRONT

Una de las herramientas usadas para la creación de la interfaz es JaxFront. Esta herramienta genera un formulario a partir de un archivo XSD (XML Schema) así como algunas otras funciones disponibles en su API.

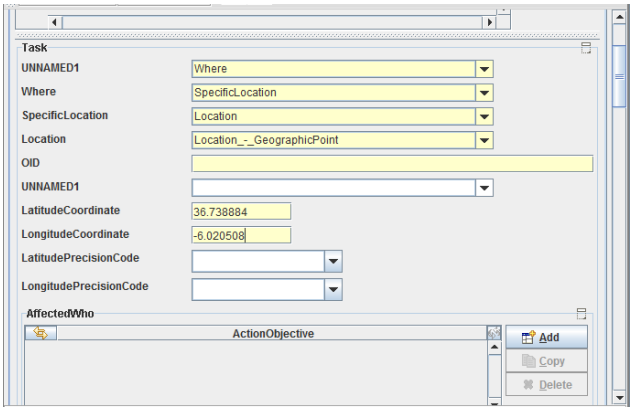

Imagen 29 - Tipo de formulario generado por JaxFront

1. Fundamentos

Esta herramienta, desarrollada en JAVA permite generar un formulario a partir de los siguientes archivos:

- XSD [Obligatorio]. Este archivo XML Schema define la estructura que tendrá el formulario, los campos que lo forman, las posibilidades que se darán a escoger al usuario, las restricciones de un determinado campo…
- XUI [Opcional]. Se trata de un archivo XML con extensión ".xui" que define aquellas modificaciones visuales que se quieren realizar sobre la interfaz. Al cargar el archivo XSD se genera una interfaz genérica, y gracias a estos archivos "xui" es posible realizar pequeños cambios. De esta manera se puede customizar la interfaz agrupando diferentes campos en tablas, o cambiando el color de un elemento. Estos archivos "xui" se crean con la herramienta "*XUIeditor*" que permite seleccionar los cambios disponibles. Estos cambios son muy limitados, ya que no permiten rediseñar la interfaz, solo añadir pequeños cambios visuales.
- XML [Opcional]. Si se incluye un archivo XML, la interfaz que se genere en vez de aparecer con todos sus campos vacíos, contendrá los datos que en este archivo se definan.

Esta API también permite algunas funciones como:

- Validar el contenido del formulario según el XSD que lo define.
- Serializar el formulario y generar un archivo XML con su contenido, siguiente el XML Schema con el que se definió.

2. Problema de implementación

A la hora de escoger una herramienta que facilitase la labor de parsear archivos para generar una interfaz con un formulario se descubrió que esta era de las más completas, aunque posee una limitación determinante en el diseño, la cual ha hecho replantear algunas partes del diseño.

La limitación que aquí se trata es la incapacidad que tiene JaxFront de procesar archivos XSD con hijos y herencias. Es bastante común encontrar en el lenguaje C-BML casos donde un tipo hereda de un padre, por ejemplo:

<Location xsi:type="OtherLocation"> → Tipo = OtherLocation hijo de AbstractLocation.

Esto provocaba que gran parte del esquema XSD de C-BML no fuese reconocido por la aplicación.

Finalmente se optó por usar una versión modificada de este donde se eliminan aquellas estructuras "padre-hijo" y se sustituye por una versión funcionalmente similar.

En este punto del diseño, pasó de usarse el XML Schema Original (XSD-SISO) y comenzó a usarse una versión modificada (XSD-GRVC). Este cambio provoca que los ficheros generados no sean fieles al modelo C-BML propuesto por la SISO, por lo que se añadió unos bloques de pre-procesado y post-procesado que permitiese hacerlos "estándar".

En la siguiente figura se seguirá la siguiente nomenclatura:

1) XSD-SISO. XSD estándar usado por C-BML.
2) XSD-GRVC. XSD modificado para el correcto funcionamiento de JaxFront.
3) XML-SISO. Misión C-BML que sigue el estándar de C-BML y su esquema oficial XSD-SISO.
4) XML-GRVC. Misión C-BML que no sigue el estándar, ya que sigue el XSD modificado.

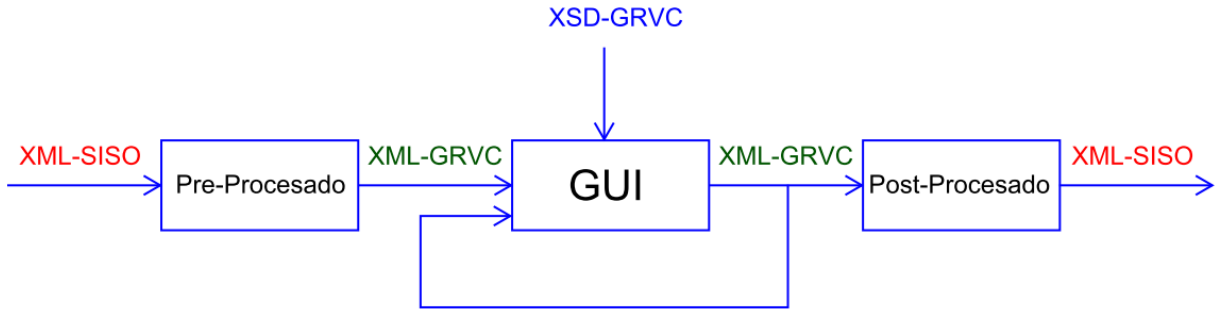

Imagen 30 - Recorrido de los mensajes XML en la aplicación del cliente

Como se ve en el esquema, el núcleo del sistema funciona tomando como referencia el XSD-GRVC modificado. Aquí se procede a explicar un poco el funcionamiento del sistema:

1) Apertura de una misión C-BML estándar. El archivo de entrada sería un XML que seguiría el esquema estándar XSD-SISO, por lo que será de tipo XML-SISO. Este mensaje se procesa para adaptarlo al XSD-GRVC y así poder ser abierto por el sistema.
2) Creación de una misión C-BML estándar. El archivo que crea la interfaz es un XML de tipo no oficial (XML-GRVC) por lo que debe ser procesado para que siga el estándar, creándose por tanto un archivo XML-SISO.
3) Reapertura de un archivo creado por la interfaz. Para abrir un archivo anteriormente creado, no es necesario post-procesar en la salida y luego pre-procesar a la entrada. Como se ve en la figura mediante un lazo de realimentación, la interfaz puede abrir un XML-GRVC creado sin necesidad de ningún tipo de procesamiento.

APÉNDICE B. OPENSTREETMAPS

1. Introducción

Para mejorar la usabilidad de la interfaz de creación de misiones, se ha creído conveniente la inclusión de un mapa donde poder buscar las coordenadas de Latitud y Longitud en las cuales han de llevarse a cabo las misiones.

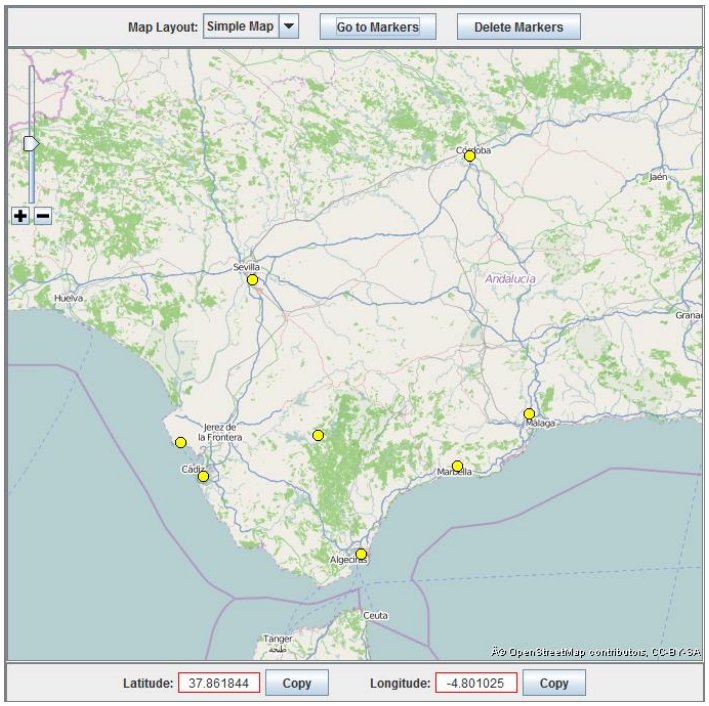

Imagen 31 - Mapa incorporado al cliente

2. Implementación

Los mapas escogidos han sido los de "*OpenStreetMaps*" por tener carácter gratuito y libre. Para ello se ha implementado dentro de la interfaz un panel con los mapas y sus principales funcionalidades. Entre ellas hay que destacar que existen dos tipos de mapas en la aplicación: mapa cartográfico y mapa foto-satélite.

La herramienta corre diferentes hilos en paralelo para la actualización de los mapas, el dibujo de estos, y la captura de eventos de ratón.

Hay que destacar que se ha llevado a cabo una implementación de una *caché de mapas*, que evita que los mapas tengan que ser actualizados continuamente mientras se usan. Si una zona del mapa todavía no se ha visualizado, los mapas son descargados vía internet de la página de OpenStreetMap usando su API. Sin embargo, para el caso en que esa zona del mapa haya sido visitada con anterioridad, se usará la imagen en caché almacenada tras la última visita, disminuyendo el consumo del tráfico de red. Además esto permite que ante cortes en la conexión, sea posible trabajar con los últimos mapas cargados. Esto se ha valorado positivamente para entornos móviles donde los microcortes podrían provocar errores en la carga de mapas, pero con este sistema se aumenta la robustez ante este tipo de incidencias.

En la siguiente imagen es posible ver como ante una pérdida de la conexión, se mantiene en la caché la imagen con mejor resolución disponible. Se observa como en el área marcada con el cuadro de puntos la resolución es mejor, mientras que en el resto del mapa la última imagen disponible es de peor resolución. Además se puede ver el mensaje indicando la problemática existente al usuario mediante el aviso *"Error: Check Internet Connection"*.

Imagen 32 - Imagen del mapa en caso de un error del conexión y una carga parcial de la imagen

Para solventar lo antes posible esta situación, se ha creado un hilo que espera la reconexión a internet y tan pronto como la obtiene, carga las zonas del mapa que no existían en la caché. Puede verse en la siguiente imagen como el mensaje de error desaparece y la zona inferior derecha pasa a tener la resolución necesaria para ser visualizada:

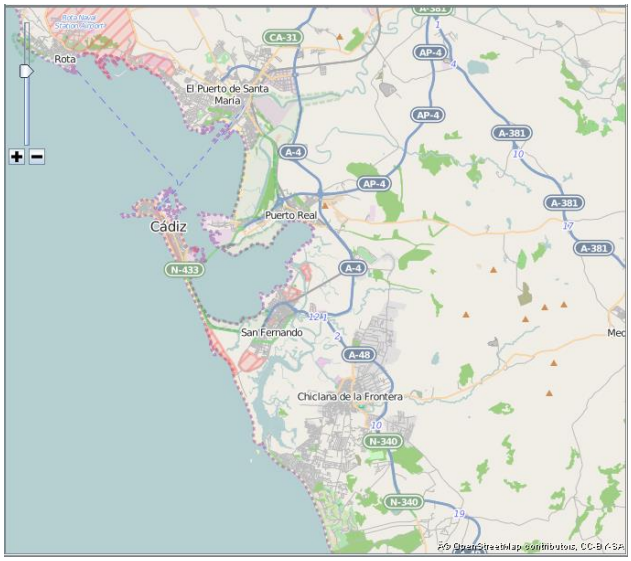

Imagen 33 - Imagen del mapa cargado completamente tras la restitución de un error de conexión

3. Errores corregidos

Además de los errores de posible pérdida de la conexión o microcortes que se han solucionado gracias al uso de una caché de mapas, se han gestionado la problemática que existe cuando se hace zoom en una zona del mapa donde existe muy poca resolución. Esto ocurre sobretodo en la fotografía satélite, donde hay zonas como el océano donde la resolución de las imágenes no es de calidad, dada la "poca utilidad" de estas áreas para el uso general. En estos casos se ha implementado un gestor de zoom que reconoce estas zonas e indica al usuario la limitación cuando este intenta insistir en "hacer zoom" sobre ese área. Además implementa un sistema por el que lleva al usuario hacia la mejor imagen disponible en caso de ocurrir esta casuística. En la siguiente figura se puede ver como en la zona central de la selva amazónica el zoom máximo está limitado, y esto se ha indicado al usuario mediante el mensaje *"There is no image for this Zoom level".*

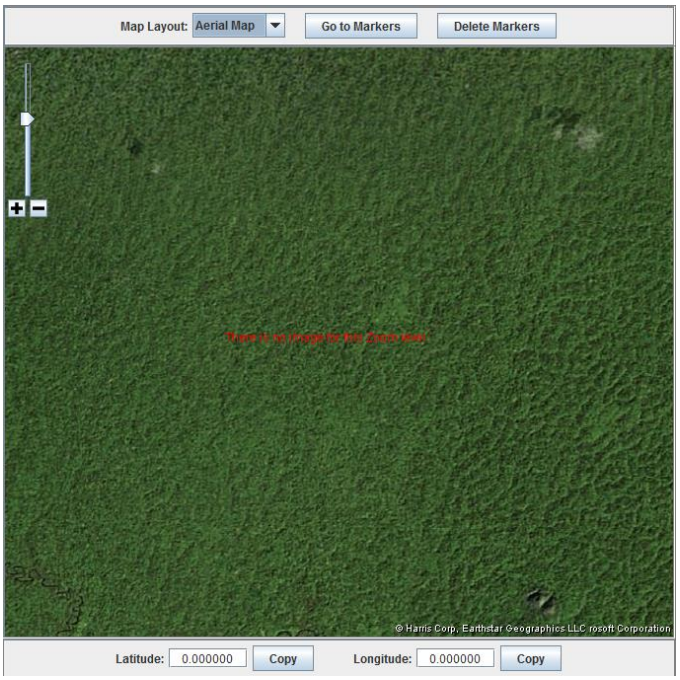

Imagen 34 - Error producido al intentar acceder a una sección del mapa con insuficiente resolución

APÉNDICE C. PASO DE MENSAJES

1. Formato de datos

Todos los mensajes serán enviados por el socket siguiendo el siguiente formato:

<##mensaje##>

De esta manera se puede evitar confundir un mensaje con el siguiente cuando estos se acumulen en el buffer de entrada del cliente.

*<##mensaje1##><##mensaje2##><##mensajeConXML**<cbml>mission</cbml>**##>*

Nota: Cuando el servidor transmita información al cliente deberá añadir un retorno de carro (\n) al final de la transmisión, para mejorar los procesos de lectura en los clientes, permitiendo que estos estén escritos en cualquier lenguaje de programación que permita conexiones vía socket.

2. Intercambio de mensajes

2.1. Conexión y Desconexión

Para la **conexión**, el cliente deberá crear un socket en la dirección IP donde se encuentre el servidor y en el puerto 9300. La creación del socket supone el diálogo automático que se crea para establecer la conexión, por lo que no es necesario que el servidor responda expresamente esta petición.

Si existe error durante el establecimiento de la conexión en C++ se devuelve el valor "-1" y en Java se produce una excepción.

Una vez establecida la conexión el cliente quedará a la espera de datos por parte del servidor, es decir, a que este le envíe el nombre de los temas existentes en ese momento en el servidor.

Este intercambio se hará con el mensaje:

<##topiclist.topic1.topic2.topic3.topic4##>

donde se ha expuesto un caso con 4 temas en el sistema. La lista aumentaría dependiendo del número de temas.

Tras este mensaje enviará una aceptación final de la conexión:

<##connected##>

Si no hubiera ningún tema en el sistema, solo deberá enviarse este último mensaje de aceptación final de la conexión.

Posibles errores en el CLIENTE:

- *IP incorrecta. Provocará un error al intentar conectar ya que no podrá crear el socket.*
- *Nunca llega el mensaje de aceptación de conexión (ACK). Cerrará la conexión siguiendo el esquema de desconexión.*

Por otro lado, la **desconexión** se lleva a cabo con el paso de 3 mensajes.

Antes de cerrar el socket el cliente deberá transmitir un mensaje de desconexión:

<##disconnect##>

Tras esto el servidor responderá con el mensaje:

<##ack.disconnect##>

Una vez que el cliente verifique la respuesta del servidor, enviará un mensaje de liberación:

<##release##>

E inmediatamente después, dispondrá los mecanismos de cierre del socket.

En caso de problemas de conexión donde el cliente no consiga dialogar con el servidor, el cliente forzará un cierre del socket sin previo intercambio de mensajes de desconexión.

El servidor una vez acabada la sesión con el cliente borrará los datos de sesión que este creó como pueden ser las subscripciones o el identificador de cliente. Los temas y misiones creadas permanecerán en el servidor hasta su apagado. El servidor verifica que la sesión ha acabado cuando se produce el mensaje de liberación y no antes. Si se corta la conexión antes de la liberación, quiere decir que alguna anomalía en el sistema ha hecho que se pierda la conexión con este, y por tanto se mantendrá toda la información de sesión a la espera de volver a recuperarla.

Posibles errores en el CLIENTE:

- *Nunca llega el mensaje de aceptación de desconexión (ACK). Preguntará si forzar la desconexión o no. En caso afirmativo cortará la conexión de manera rotunda sin envío de mensaje de liberación.*

2.2. Temas

Los nombres escogidos para los temas deberán estar formados por los rangos de caracteres (A-Z, a-z, 0-9) además del carácter “barra baja” (_). Además ha de añadirse la restricción de que no se permiten nombres duplicados.

2.2.1. Crear un nuevo tema

Cuando el cliente crea un nuevo tema este es enviado al servidor y desde allí se envía a todos los clientes conectados (Broadcast).

Que un cliente cree un tema no significa que se subscriba a él. La subscripción se realiza de manera separada con su mensaje correspondiente.

Cuando el cliente crea un tema se envía el nombre de este al servidor con el siguiente mensaje:

<##createtopic.nombretopic##>

Una vez el servidor lo recibe y comprueba que es posible añadirlo, responde en broadcast a todos los clientes con el mensaje:

<##newtopic.nombretopic##>

El cliente que envió la petición recibe por tanto el mismo mensaje que el resto de sus similares, sirviendo como mensaje de aceptación del tema.

Posibles errores en el CLIENTE:

- *Nunca llega el mensaje de aceptación del tema (newtopic). Indicará el error y no ejecutará ninguna acción más, descartando la petición.*
- *El tema que se intenta crear ya existe. No ejecutará ninguna acción, se mostrará un mensaje de aviso y se descartará la petición.*

2.2.2. Subscribirse a un tema

Cuando un cliente decide subscribirse a uno de los temas existentes en el servidor enviará el mensaje:

<##subscribe.nombretopic##>

El servidor responderá enviando el nombre de todas las misiones que tiene ese tema. Este intercambio se hará con el mensaje:

<##missionlist.nombretopic.mission1.mission2.mission3.mission4##>

---

donde se ha expuesto un caso con 4 misiones en ese tema. La lista aumentaría dependiendo del número de misiones.

Tras este mensaje, el servidor enviará una aceptación final de la subscripción:

<##ack.subscribe.nombretopic##>

Si no hubiera ninguna misión en el tema al que se subscribe, solo deberá enviarse este último mensaje de aceptación final de la subscripción.

Posibles errores en el CLIENTE:

- *Nunca llega el mensaje de aceptación de la subscripción (ACK). Indicará el error y no ejecutará ninguna acción más, descartando la petición de subscripción.*
- *Se subscribe un tema que no existe.*

2.2.3. Eliminar la subscripción a un tema

De esta manera el cliente comunica al servidor que no quiere recibir más información sobre ese tema. Se lleva a cabo con el siguiente mensaje:

<##unsubscribe.nombretopic##>

El servidor responderá con el mensaje:

<##ack.unsubscribe.nombretopic##>

Posibles errores en el CLIENTE:

- *Nunca llega el mensaje de aceptación de eliminación de la subscripción (ACK). Indicará el error y no ejecutará ninguna acción más, descartando la petición de eliminación de la subscripción.*
- *Se elimina la subscripción de un tema que no existe. Este error nunca podrá ocurrir ya que en la interfaz solo se reflejan aquellos que sí existen en el servidor y por tanto serán los únicos que podrán seleccionarse para eliminar la subscripción.*

2.3. Misiones

Los nombres escogidos para las misiones deberán estar formados por los rangos de caracteres (A-Z, a-z, 0-9) además del carácter “barra baja” (_).

2.3.1. Añadir una nueva misión XML:

Cuando el cliente desea incluir una misión creada por él mismo en un cierto tema, usará el siguiente formato de mensaje:

<##createmission.nombretopic.nombremision.<XMLdeCBML>##>

Donde <XMLdeCBML> será exactamente todo el contenido dentro del archivo XML (Recuerde que puede incluir en su interior puntos, retornos de carro, tabulaciones, espacios y toda aquella sintaxis permitida en un XML).

Una vez recibido y validado por el servidor, este enviará un mensaje en difusión únicamente a todos los clientes subscritos al tema donde se ha añadido la misión. Este mensaje contendrá el nombre de la misión y el tema donde se encuentra. El mensaje será:

<##newmission.nombretopic.nombremision##>

El cliente que envió la petición recibe por tanto el mismo mensaje que el resto de sus similares, sirviendo como mensaje de aceptación del envío.

Posibles errores en el CLIENTE:

- *Nunca llega el mensaje de aceptación de la misión (newmission). Indicará el error y no ejecutará ninguna acción más, descartando la petición.*
- *La misión que el cliente intenta incluir en ese tema ya existe. No ejecutará ninguna acción, se mostrará un mensaje de aviso y se descartará la petición.*
- *La misión que el servidor intenta incluir en ese tema ya existe. No ejecutará ninguna acción y se descartará la petición.*

2.3.2. Pedir el XML de una misión:

Cuando un cliente se subscribe a un tema recibe el nombre de todas las misiones que este contiene, pero no recibe el archivo XML de cada una de estas misiones, ya que estas han de ser pedidas una a una bajo demanda con el siguiente mensaje:

<##getmission.nombretopic.nombremision##>

El servidor procesará esta petición y responderá con el contenido del archivo XML. El formato del mensaje será:

<##ack.getmission.nombretopic.nombremision.<XMLdeCBML>##>

En este momento el cliente podrá escoger donde guardar la misión C-BML en XML que acaba de recibir.

Posibles errores en el CLIENTE:

- *Nunca llega el mensaje de aceptación de la descarga de la misión (ACK). Indicará el error y no ejecutará ninguna acción más, descartando la petición.*

2.4. Petición del log del Servidor

Existirá una funcionalidad con la que un cliente podrá pedir el estado del servidor. De esta manera, podrá hacerse una idea del entorno y de cómo están trabajando los otros clientes.

La petición se llevará a cabo con el mensaje:

<##serverlog##>

El servidor enviará de vuelta el contenido del log como un texto plano según el siguiente mensaje:

<##ack.serverlog.datos_del_log##>

Una vez recibido el log, el cliente podrá guardarlo como archivo de texto plano para su consulta.

Posibles errores en el CLIENTE:

- *Nunca llega el mensaje de aceptación de la descarga del log (ACK). Indicará el error y no ejecutará ninguna acción más, descartando la petición.*

2.5. Servicio "IsAlive"

Para conocer el estado de la conexión se llevarán a cabo las siguientes actuaciones:

- El servidor deberá verificar que el estado de la conexión con todos los clientes que aceptó se mantiene correctamente evaluando los parámetros de la conexión.
- El cliente deberá verificar que el estado de la conexión con el servidor es correcta. Esta se llevará a cabo con un refresco de 2 segundos y mediante el envío del mensaje:

<##alive?##>

Si este mensaje no da error en el envío quiere decir que la conexión es correcta. El servidor lo recibirá y descartará, ya que no es necesaria su respuesta.

APÉNDICE D. MENSAJES DEL CLIENTE

1. Errores relacionados con el panel de misiones

ERROR_0000 = "Unexpected Error - It is recommended to restart the application.

Error producido por cualquier causa extraña

ERROR_0001 = "It's necessary to load an XML Schema (XSD) file."

Se produce cuando se intenta crear un formulario o abrir un archivo XML sin llegar a seleccionar un archivo XSD.

ERROR_0002 = "The file selected is not valid."

Se produce cuando al crear el árbol DOM se da una excepción. Normalmente por seleccionar un archivo con errores de formato con el cual es imposible crear un objeto java con su información.

ERROR_0003 = "The file has invalid syntax."

Se produce cuando el archivo se abre, se modifica y se crea el objeto java, siendo este último erróneo y medio vacío al haber habido errores durante el parseo. Este error se implementa por la falta de una excepción que indique este error.

ERROR_0004 = "The file has invalid namespace syntax."

Se produce cuando al comprobar el target namespace dentro de la cabecera este se omite o posteriormente cuando se descubre que es null.

ERROR_0005 = "XML file not compliant with XSD file."

Se produce cuando comprobando el namespace obtenido del XML y el obtenido del XSD son diferentes.

ERROR_0006 = "The file selected is not valid."

Se usa para todos los casos donde el archivo está vacío o simplemente no es válido.

ERROR_0007 = "The XML file has invalid syntax."

Se produce cuando al ir parseando el archivo XML se encuentra algo que no debería ocurrir.

ERROR_0008 = "The XSD file has invalid syntax."

Se produce cuando al ir parseando el archivo XSD se encuentra algo que no debería ocurrir.

ERROR_0009 = "Interface unexpected error."

Se produce cuando ocurre un error inesperado en la interfaz.

ERROR_0010 = "JaxFront unexpected error."

Se produce cuando ocurre un error inesperado en la librería Jaxfront.

ERROR_0011 = "Stack Overflow - It was impossible to open the file"

Se produce cuando de manera inesperada se satura la memoria de la máquina java. Este error depende de la máquina java. suele ocurrir en versiones anteriores a java 8 por utilizar mecanismos menos eficientes.

ERROR_0012 = "You must use Java JRE 1.8 or higher!"

Se lanza cuando no cumple las especificaciones mínimas de ejecución, en concreto la versión mínima de la máquina java.

INFO_0001 = "Exportation cancelled"

Se produce al pulsar sobre exportar archivo y en vez de seleccionar el nombre del archivo de importación, se pulsa sobre cancelar. Se avisa para que el usuario sepa que el archivo no ha sido guardado.

2. Errores relacionados con el panel de control del servidor

ERROR_0001 = "Server ERROR: Not ready to create a Topic."

Error producido cuando se crea un nuevo tema (topic) y no se recibe ACK.

ERROR_0002 = "Server ERROR: Connection to server was lost."

Error producido cuando de repente se pierde la conexión con el servidor y no se consigue reconectar.

ERROR_0003 = "Connection ERROR: Invalid IP."

Cuando la IP es incorrecta o no hay nadie que responda a esa IP.

ERROR_0004 = "Server ERROR: Not ready to create a Connection."

Error producido cuando se establece una conexión y no se recibe ACK.

ERROR_0005 = "Server ERROR: Not ready to subscribe to a topic."

Error producido cuando se subscribe un tema (topic) y no se recibe ACK.

ERROR_0006 = "Server ERROR: Not ready to unsubscribe from a topic."

Error producido cuando se elimina la subscripción de un tema (topic) y no se recibe ACK.

ERROR_0007 = "Server ERROR: Not ready to disconnect."

Error producido cuando se intenta desconectar y no se recibe ACK.

ERROR_0008 = "Server ERROR: Not ready to add a mission."

Error producido cuando se intenta desconectar y no se recibe ACK.

ERROR_0009 = "Server ERROR: Not ready to download a mission."

Error producido cuando se intenta descargar una misión y no se recibe la descarga.

ERROR_0010 = "Server ERROR: Not ready to download the Server Log."

Error producido cuando se intenta descargar el Log del servidor y no se recibe la descarga.

INFO_0001 = "Impossible to add. This topic already exists!"

Se muestra cuando se añade un tema (topic) que ya existe.

INFO_0002 = "Impossible to add. This mission already exists!"

Se muestra cuando se añade un tema (topic) que ya existe.

APÉNDICE E. MENSAJES DEL SERVIDOR

"Error_0000: [Create Topic] Topic $message already exists."

Error producido cuando se quiere crear un tema ya existente.

string $message Nombre del tema.

"Error_0001: [Subscribe] This subscription $id - $message already exists."

Se produce cuando se intenta crear una relación cliente – tema ya existente.

string $id ID de la conexión.

string $message Nombre del tema.

"Error_0002: [Subscribe] The topic $message does not exist."

Se produce cuando el cliente se intenta suscribir a un tema no existente.

string $message Nombre del tema.

"Error_0003: [Unsubscribe] The subscription $id - $message does not exist."

Se produce cuando se intenta eliminar una suscripción no existente.

string $id ID de la conexión.

string $message Nombre del tema.

"Error_0004: [Create Mission] The mission $nombre_mission from $topic_name already exists."

Se produce cuando se intenta publicar una misión ya existente en el tema.

string $nombre_mission Nombre de la misión.

string $topic_name Nombre del tema.

"Error_0005: [Create Mission] The topic $topic_name does not exist."

Se produce cuando se intenta publicar una misión en un tema no existente.

string $topic_name Nombre del tema.

"Error_0006: [Create Mission] The XML $nombre_mission has not a valid format."

Se produce cuando la misión no tiene un formato válido.

string $nombre_mission Nombre de la misión.

"Error_0007: [Get Mission] The mission $nombre_mission does not exist."

Se produce cuando se intenta descargar una misión que no existe.

string $nombre_mission Nombre de la misión.

"Error_0008: [Error] Write error to [$id]."

Se produce cuando el servidor no consigue enviar la respuesta al cliente.

string $id ID de la conexión.

APÉNDICE F. CASO PRÁCTICO

En este último apéndice, se va a ver una demostración completa de la comunicación entre cliente y servidor para los diferentes tipos de mensajes definidos anteriormente.

5.3 Conexión/Desconexión

1.1 Conexión

Para conectarse al servidor, se crea un *socket* con un ID identificativo de la conexión, que será es el ID que usará el servidor para el resto de mensajes entre ambos, se añade una entrada en la base de datos para la nueva conexión y se envía el asentimiento de conexión al cliente. Antes del asentimiento, se comprueba si existe algún tema (topic) creado en la base de datos. De ser así, se le envía una lista con sus nombres.

Así pues,

1) Se indica la IP del servidor y se pulsa sobre el botón *Connect* en el cliente.

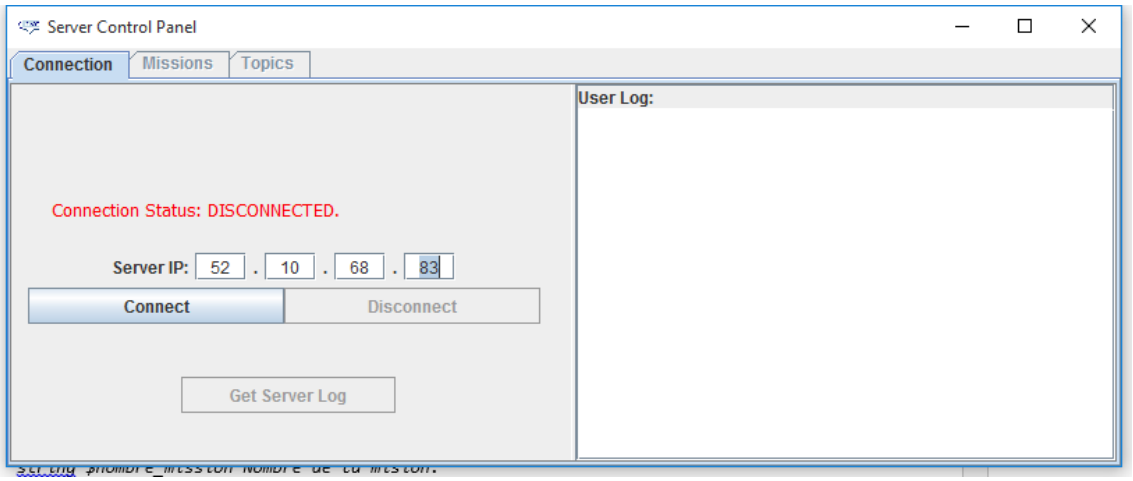

Imagen 35 - Conexión del cliente

2) En el servidor,

Imagen 36 - Notificación de cliente conectado

3) El cliente recibe el asentimiento.

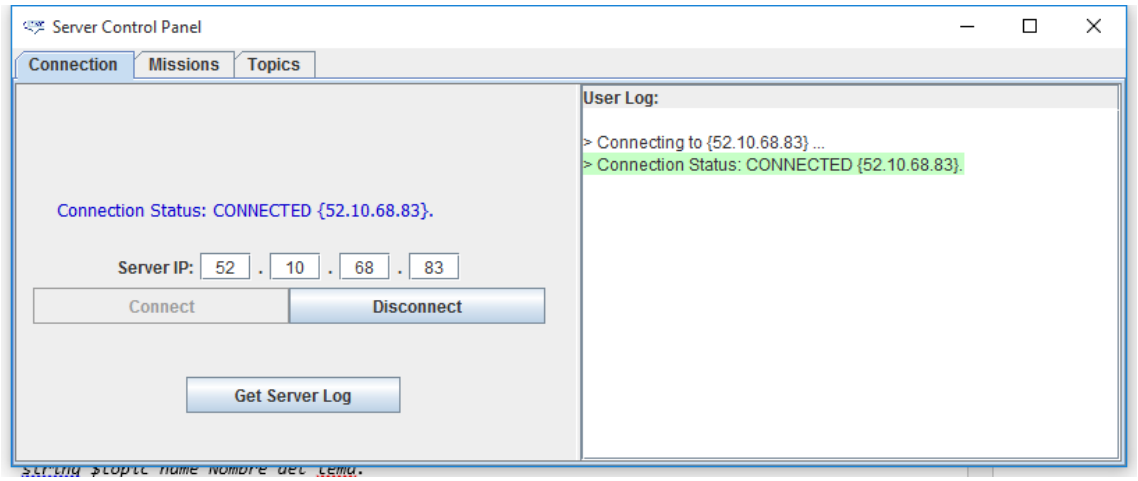

Imagen 37 - Cliente conectado

Si en el servidor hubiera temas ya creados, éstas aparecerían en la pestaña *Topics* del cliente tras la conexión.

1.2 Desconexión

El cliente envía la petición de desconexión al servidor. Este asiente y espera la petición de *release* para liberar el socket y borrar el cliente de la base de datos.

Así pues,

1) Se pulsa sobre *Disconnect* en el cliente.

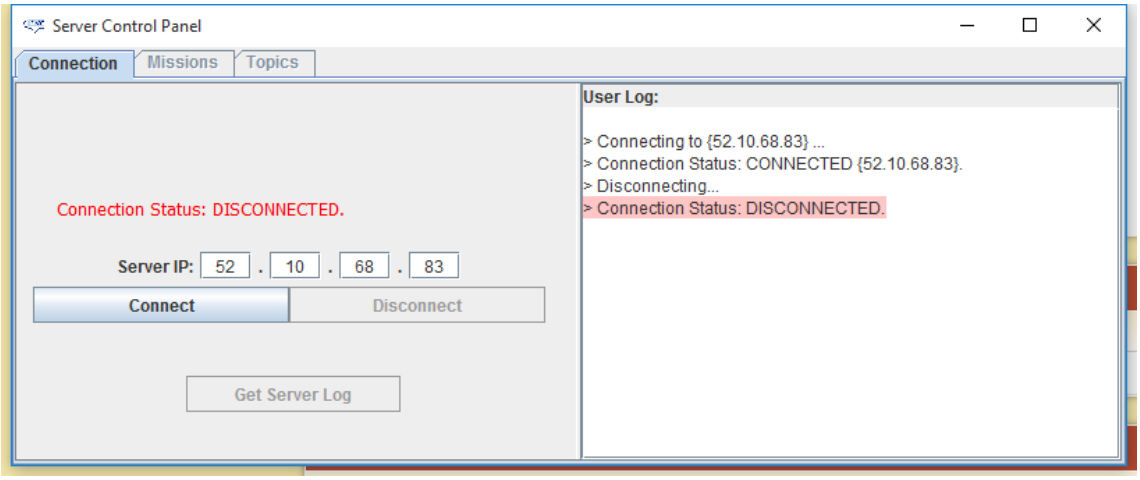

Imagen 38 - Desconexión del cliente

2) El estado del cliente queda actualizado.

```
2015-08-16 19:41:45 SERVER CONNECTED
2015-08-16 19:43:40 [Connect] New connection [11] from 92.57.161.19:57754
2015-08-16 19:45:13 [Disconnect] Disconnect request from connection 11
2015-08-16 19:45:14 [Disconnect] Connection [11] closed
```

Imagen 39 - Notificación de cliente desconectado

1.3 Temas

1.3.1 Creación de un nuevo tema

El cliente envía el mensaje para crear un nuevo tema al servidor. Éste crea la entrada en base de datos y envía el asentimiento al cliente.

Así pues,

1) Se vuelve a conectar el cliente. En la pestaña *Topics*, se le proporciona un nombre al nuevo tema (topic) y se pulsa sobre *Add & Publish* .

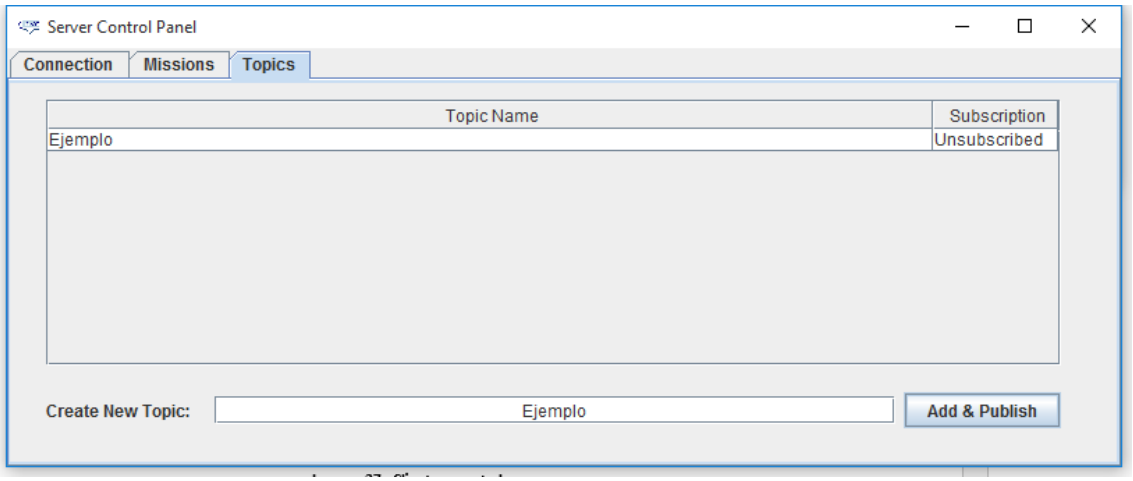

Imagen 40 - Publicación de un nuevo tema

El tema aparece ahora en la lista de temas existentes (*Topics*).

2) En el servidor se comprueba que, efectivamente, el nuevo tema se ha creado.

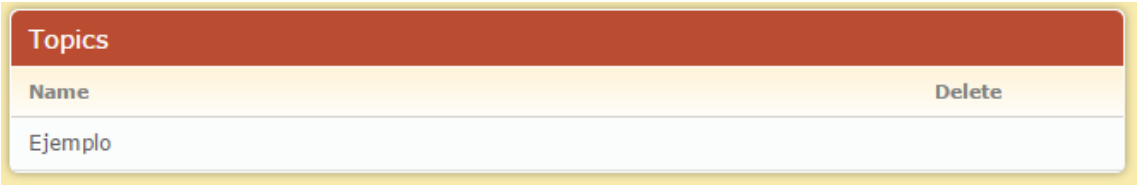

Imagen 41 - Nuevo tema creado en el servidor

A su vez, se puede comprobar cómo, efectivamente, el cliente sigue conectado.

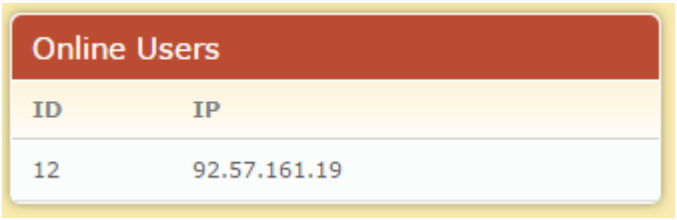

Imagen 42 - Clientes conectados

1.3.2 Suscripción de un tema

El cliente envía la solicitud de suscripción a un tema (topic) existente en el sistema. El servidor crea la relación en la base de datos y envía posteriormente un asentimiento al cliente.

Así pues,

1) Hacer doble click sobre el tema al que se desea suscribir dentro de *Topics.*

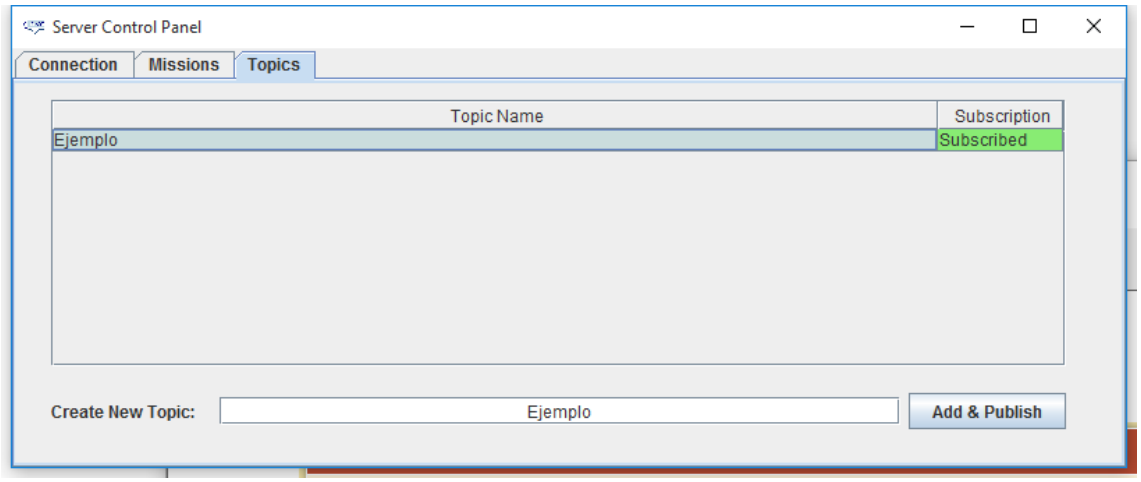

Imagen 43 - Suscripción a un nuevo tema

El tema aparece en verde. Esto quiere decir que la suscripción ha sido realizada con éxito.

2) En el servidor,

```
2015-08-16 19:41:45 SERVER CONNECTED
2015-08-16 19:43:40 [Connect] New connection [11] from 92.57.161.19:57754
2015-08-16 19:45:13 [Disconnect] Disconnect request from connection 11
2015-08-16 19:45:14 [Disconnect] Connection [11] closed
2015-08-16 19:47:07 [Connect] New connection [12] from 92.57.161.19:58222
2015-08-16 19:47:16 [Create Topic] Connection 12 has created a new topic: Ejemplo.
2015-08-16 19:48:34 [Subscribe] Connection 12 subscribed to the topic Ejemplo.
```

Imagen 44 - Suscripción realizada con éxito

1.3.3 Eliminar la suscripción de un tema

El cliente envía la solicitud para eliminar la suscripción a un tema. El servidor elimina la relación en la base de datos y envía posteriormente un asentimiento al cliente.

Así pues,

1) Se hace doble clic sobre el tema del que se quiere dejar de formar parte.

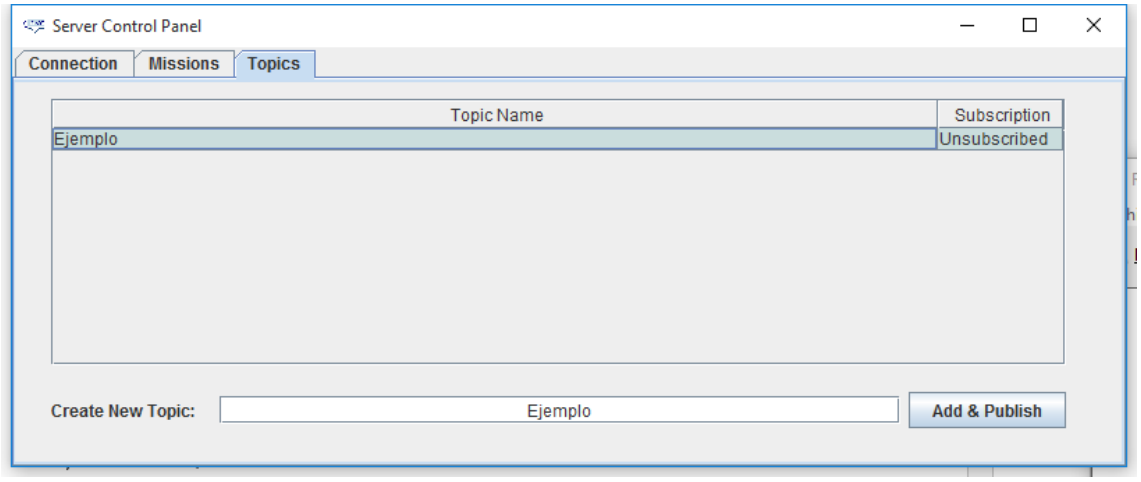

Imagen 45 - Eliminar la suscripción de un tema

Se puede ver como en el tema aparece nuevamente *Unsubscribed.*

2) En el servidor,

```
2015-08-16 19:41:45 SERVER CONNECTED
2015-08-16 19:43:40 [Connect] New connection [11] from 92.57.161.19:57754
2015-08-16 19:45:13 [Disconnect] Disconnect request from connection 11
2015-08-16 19:45:14 [Disconnect] Connection [11] closed
2015-08-16 19:47:07 [Connect] New connection [12] from 92.57.161.19:58222
2015-08-16 19:47:16 [Create Topic] Connection 12 has created a new topic: Ejemplo.
2015-08-16 19:48:34 [Subscribe] Connection 12 subscribed to the topic Ejemplo.
2015-08-16 19:51:49 [Unsubscribe] Connection 12 has been unsubscribed from the topic Ejemplo.
2015-08-16 19:51:50 [Subscribe] Connection 12 subscribed to the topic Ejemplo.
2015-08-16 19:51:51 [Unsubscribe] Connection 12 has been unsubscribed from the topic Ejemplo.
```

Imagen 46 - Suscripción eliminada en el servidor

1.4 Misiones

1.4.1 Añadir una misión XML

Cuando un cliente quiere añadir una misión XML a un tema (topic) ya existente, debe enviar la petición al servidor y éste enviará su nombre a todos los clientes conectados a dicho tema como respuesta.

Así pues,

1) En el cliente, volver a suscribirse al tema *Ejemplo* creado previamente.
2) En la pestaña *Missions*, con el tema *Ejemplo* seleccionado como única opción, se pulsa sobre *Add Mission* y se busca la misión a publicar en dicho tema.

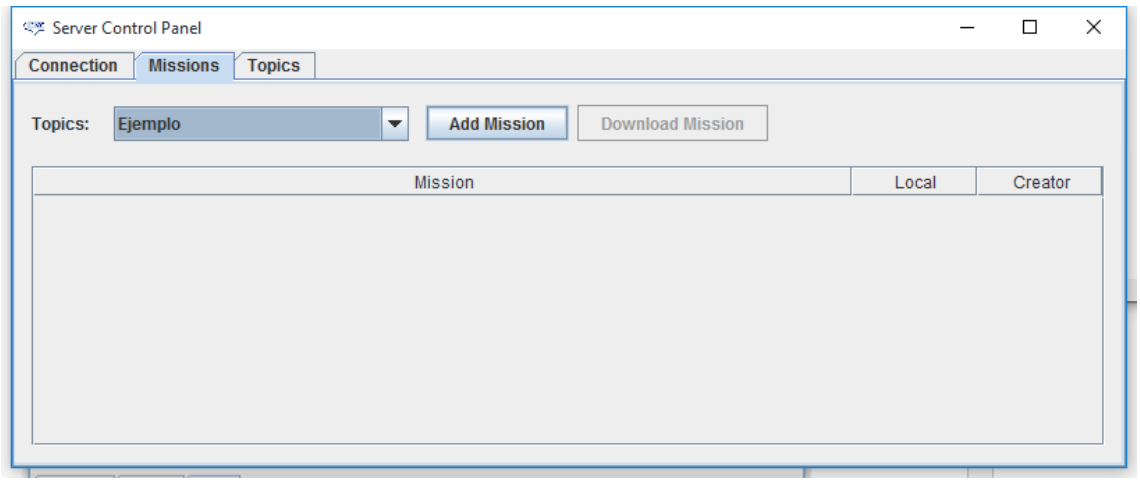

Imagen 47 - Publicar una misión

Tras pulsar sobre *Open Valid Mission*,

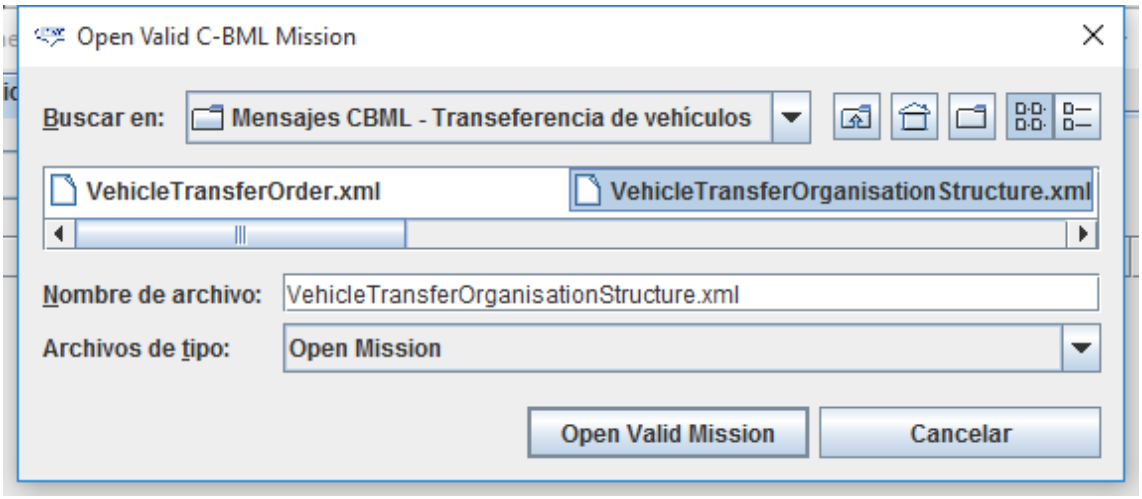

Imagen 48 - Selección de misión

La misión se ha creado correctamente en el servidor (es válido el formato XML) y se ha recibido de vuelta al estar subscrito al tema donde se publicó.

3) En el servidor,

Imagen 49 - Misión almacenada en el servidor

1.4.2 Pedir el XML de una misión

El cliente envía la petición al servidor y éste le responde con el XML dentro del mensaje de asentimiento.

Así pues,

1) Desde otro cliente, se realiza la subscripción al tema *Ejemplo* para tener la opción de descargar la misión que el otro cliente acaba de subir. Tras esto, en la pestaña *Missions*, se ve la misión que ha subido el otro cliente.

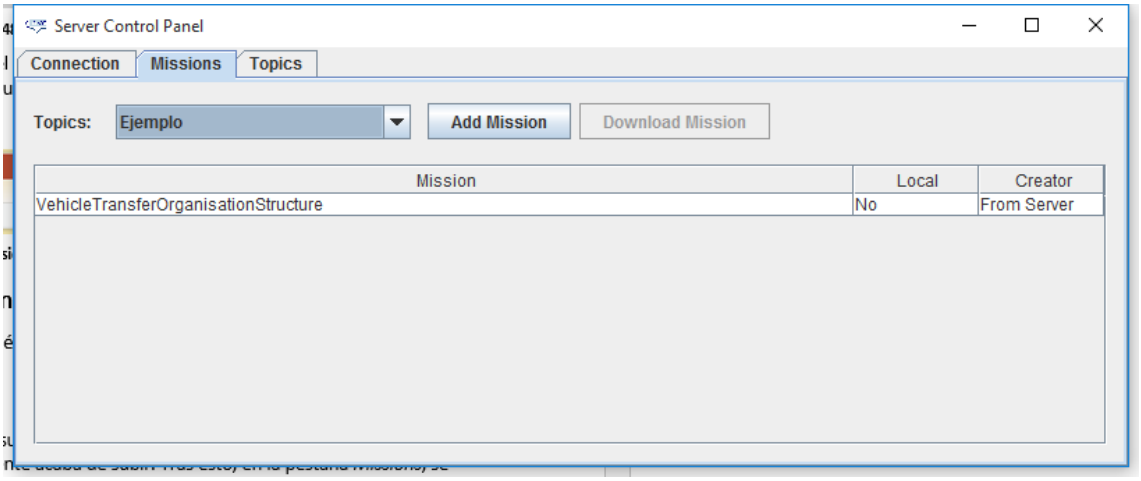

Imagen 50 - Misiones en el tema "Ejemplo"

Esta misión es posible ver que NO está en local, y que el origen sería desde el servidor, es decir, que no ha sido creada por él mismo. Ahora, si se pulsa sobre la misión, se activa la opción *Download Mission.* Pulsando sobre ella es posible seleccionar la ruta y el nombre donde se quiere guardar la misión. Tras esto, se pulsa sobre *Save XML Mission* y la misión quedará alojada en *Local.*

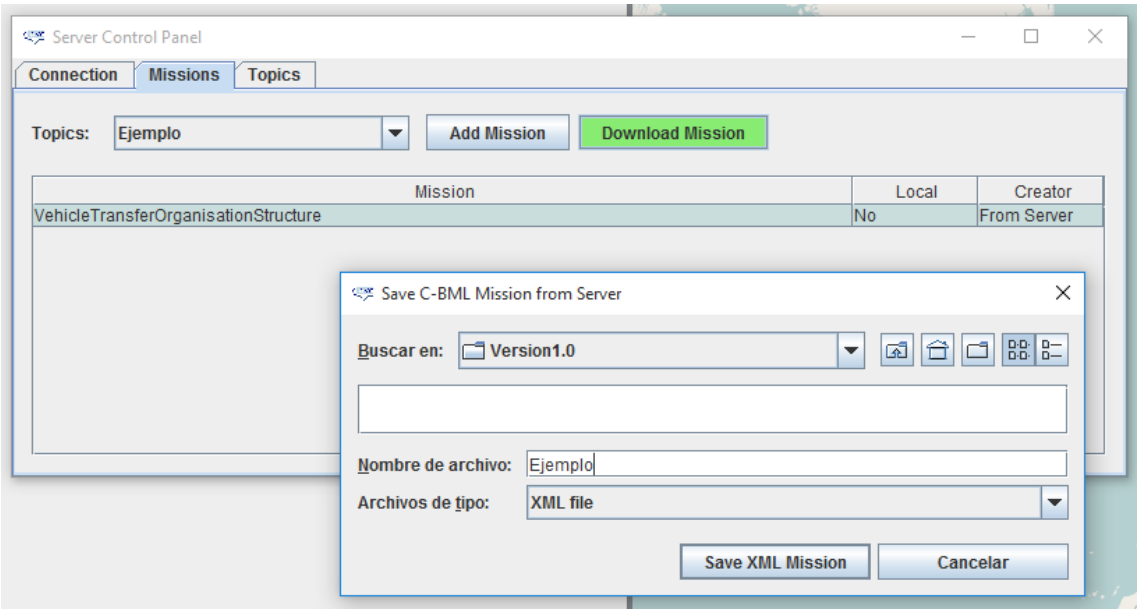

Imagen 51 - Descarga de misiones

Si se inspecciona el XML de ésta misión, es posible ver que se ha descargado correctamente.

1.5 Log del Servidor

El cliente envía la solicitud para tener el *log* del servidor y éste le responde con el *log* dentro del mensaje en texto plano.

Así pues,

1) En el cliente, en la pestaña *Connection*, pulsar sobre *Get Server Log*.

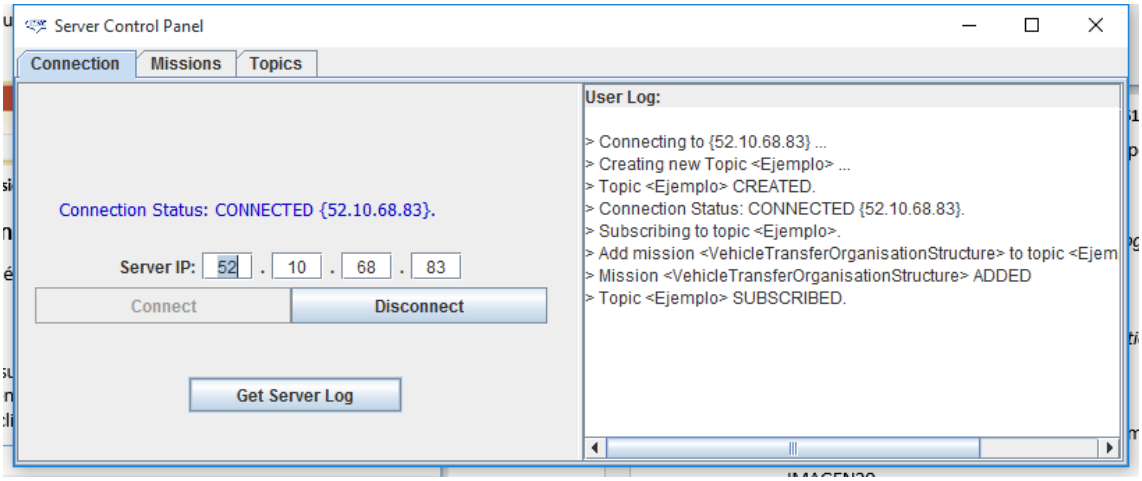

Imagen 52 - Descargar el log del servidor

2) Se escoge el nombre y ruta para almacenar el archivo del log y se pulsa en *Save Server Log*

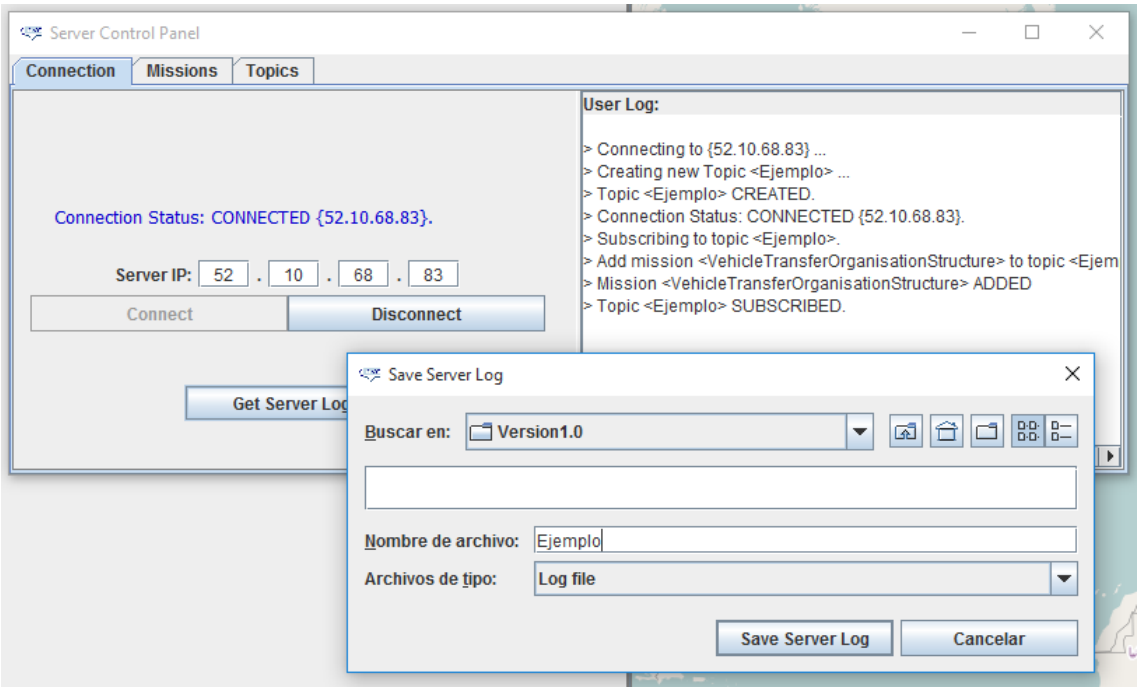

Imagen 53 - Log del servidor

Una vez hecho esto, se comprueba que se haya descargado correctamente.