Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Diseño e Implementación de una Arquitectura para una Interfaz de Mensajeria Instantánea en DDS

Javier Povedano Molina

Dirigido por: Juan Manuel López Soler
Juan José Ramos Muñoz

Departamento de Teoría de la Señal, Telemática y Comunicaciones
Universidad de Granada

Granada, 2 de mayo de 2008

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

LISTA DE TABLAS

ÍNDICE DE LISTADOS

Parte I

Manual Técnico

CAPÍTULO 1

Introducción

1.1. Definición del problema

Hoy en día cada vez es más usual el uso de aplicaciones distribuidas. No es raro encontrarse con una serie de aplicaciones que se ejecutan en distintas máquinas de forma cooperativa dando la sensación de ser todo un mismo sistema.

Dentro de estos sistemas de computación distribuida, se pueden encontrar los denominados servicios de *distribución de datos*. En este caso, lo que se encuentra disperso no son las aplicaciones, sino una serie de datos de interés. Un ejemplo claro de ello puede ser la monitorización de sensores. Cada sensor puede encontrarse en un lugar distinto, los cuales se comunican con la entidad monitorizadora por medio de alguna red de conexión.

En casos de datos distribuidos como los mencionados anteriormente, los requisitos de tiempo real, fiabilidad, falla ante errores suelen ser muy importantes. Una arquitectura que mantenga todas estas cualidades de una manera adecuada es muy compleja de construir. No sería razonable tener que diseñar para cada nueva aplicación que requiera de distribución de datos una arquitectura que la soporte. Para paliar este problema, algunas empresas han dedicado esfuerzos en diseñar una infraestructura de distribución de datos de una manera común y razonable. Este esfuerzo se ha materializado en lo que hoy se conoce como Data Distribution Service (DDS)[2].

DDS proporciona un *middleware* que abstrae al programador de aplicaciones que necesiten de datos distribuidos de las tareas necesarias para la recogida de los mismos, con máximas prestaciones. DDS deja de lado el clásico paradigma cliente/servidor , para usar un nuevo paradigma mucho más adecuado para distribución de datos: publicación/suscripción.

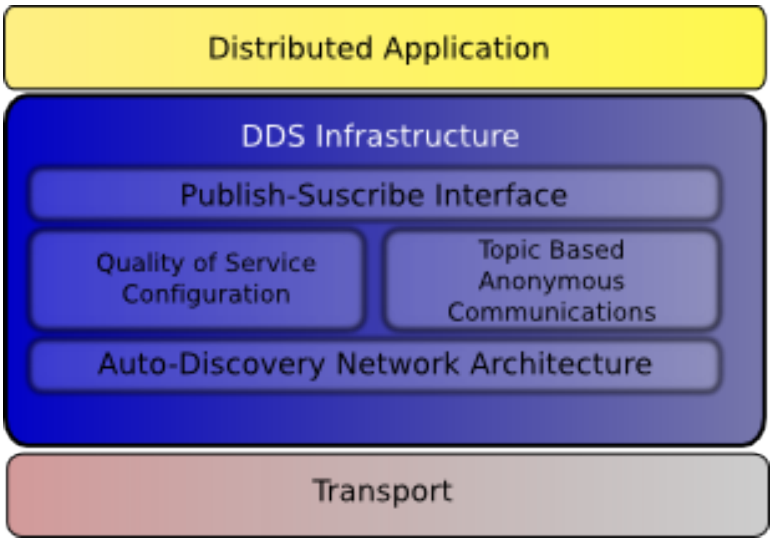


Figura 1.1: Infraestructura de DDS

DDS es por tanto un esfuerzo que normaliza el mecanismo de publicación/suscripción, de manera que puede ser añadido a cualquier aplicación de una manera sencilla. Las ventajas de usar DDS son:

1. Está basado en un paradigma conceptualmente sencillo como es publicación/suscripción

2. Es escalable dinámicamente

3. Está diseñado de modo que la sobrecarga del sistema sea mínima

4. Soporta comunicaciones de uno a uno, de uno a muchos o de muchos a muchos

5. Facilita al desarrollador enormemente la tarea de desarrollo de aplicaciones para distribución de datos

La utilización de un nuevo paradigma supone cambiar la manera de trabajar de las aplicaciones en muchos aspectos. El paradigma de publicación/-suscripción no se basa en la existencia de un servidor central, que es el que se encarga de satisfacer las solicitudes del cliente. En lugar de ello, lo que se hace es que cada entidad (es decir, un computador o sistema embebido que soporte DDS) que posea un dato susceptible de ser de interés para otra entidad, informa al *middleware* de la disponibilidad del dato y *publica* dicho dato (de ahí el término publicación). Por otro lado, existen entidades con interés hacia ciertos datos. En este caso, cuando una entidad está interesada en un dato, lo que se hace es que se informa al middleware de dicho interés,

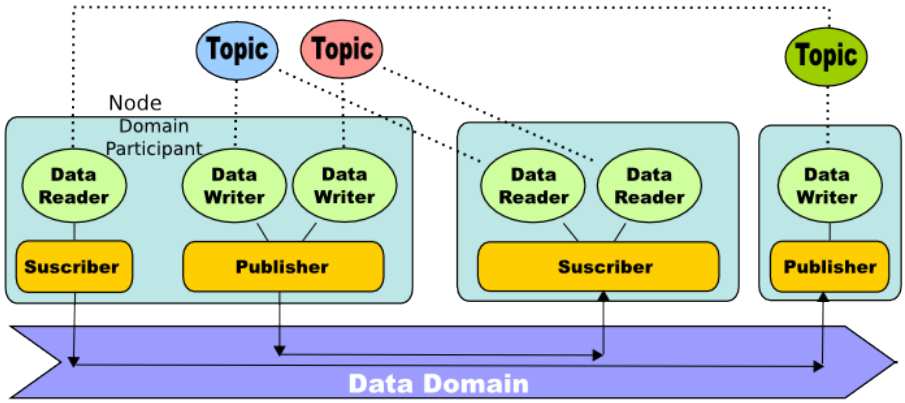


Figura 1.2: Entidades DDS participantes en un Dominio (Domain)

y éste se encargara de proporcionarlo. En este caso, se dice que la entidad se ha suscrito a un tipo de dato.

Además de estos conceptos de *publicación/suscripción*, para facilitar la explicación del trabajo realizado y de la arquitectura de DDS, se hace necesaria la explicación de nuevos conceptos:

Descubrimiento (Discovery): Es el proceso que se realiza para identificar las entidades que están publicando datos en DDS. Este proceso se inicia de forma casi instantánea por DDS, y sólo requiere que el usuario proporcione un punto de partida donde comenzar la búsqueda.

Tópico (Topic) : Se define como tópico cada uno de los distintos tipos de datos que son publicados en DDS. Cada tópico tiene sus propias características, y sus tipos de datos asociados.

Instancia (Instance): Un mismo tópico (tipo de datos) puede ser publicado por muchas entidades. Para que cada tópico publicado se diferencie de los demás se usa un mecanismo de claves únicas. Por ejemplo, si tenemos datos de posición de diversos radares (algo muy común en triangulación), para identificar de dónde proviene cada dato añadimos un campo de clave, para que cada dato de posición se utilice correctamente en los cálculos de determinación de la posición exacta de un objeto. El empleo de claves está gestionado por el middleware de DDS.

Muestra (Sample): Una vez definido el concepto de instancia es conveniente definir el concepto de muestra. Éste se refiere a cada uno de los valores que toma una instancia a lo largo del tiempo. En el ejemplo de los radares empleado en la definición de instancia, una muestra sería

cada uno de las sucesivas tomas de datos de cada radar que determinan la distancia del mismo en un instante determinado.

Dominio (Domain): Mecanismo de agrupamiento de entidades en DDS. Cada entidad se encuentra situada dentro de un dominio por medio de un participante de dominio. Todas las aplicaciones que dispongan de participantes en un dominio, pueden descubrir e interaccionar con otras aplicaciones remotas con participantes en el mismo dominio.Mediante este mecanismo se consigue aumentar la escalabilidad así como aislamiento de tráfico.

Una vez definidos estos conceptos, el siguiente paso es identificar las entidades presentes en DDS para ver como se relacionan. Esto se pone de manifiesto en la figura 1.1, donde aparecen una serie de entidades relacionadas que se identifican a continuación:

Suscriber: Es el punto de comunicacion entre los DataReader y la aplicación. Se encarga de gestionar las suscripciones (tópicos en los que se haya declarado interés).

Publisher: Es el encargado de publicar tópicos en un dominio, que serán recogidos por los suscriptores situados en otro participante de dominio.

DataWriter: Es el punto de acceso de una aplicación a DDS para publicar datos que serán publicados por un publisher

DataReader: Es el punto de acceso de una aplicación a DDS para recibir datos que fueron recogidos por un suscriber

Además de lo anteriormente comentado, es conveniente mencionar una caraterísticas ventajosa de DDS: internamente, todas las entidades presentes en DDS, disponen de una serie de políticas de calidad de servicio (QoS). Estas políticas son modificables de modo que pueden afinarse en función del entorno de explotación final.

Políticas de Calidad de Servicio (QoS) Las políticas de calidad de servicio (QoS) son uno de los atractivos de utilizar DDS. Mediante ellas se pueden gestionar aspectos muy variados del funcionamiento de DDS, desde el descubrimiento de entidades remotas distribuidas, hasta parámetros de tiempo de recogida de datos o de usuario. Las políticas de calidad de servicio disponibles son:

- Destination Order
- Entity Factory
- Latency Budget
- Deadline
- Group Data
- Lifespan
- Durability
- History
- Liveliness

- Ownership
- Ownership Strength
- Partition
- Presentation
- Reader Data Lifecycle
- Reliability
- Resource Limits
- Time-Based Filter
- Topic Data
- Transport Priority
- User Data
- Writer Data Lifecycle

De todas las políticas mostradas, algunas sólo están disponibles para ciertos tipos de entidades. Por otro lado sólo algunas de ellas pueden ser modificadas dinámicamente. De todas estas políticas de calidad de servicio, se destacan algunas de ellas, por ser de mayor interés por sus características:

Deadline Este parámetro se refiere al ratio mínimo de envío de datos por parte de un DataWriter. Es útil para aplicaciones que publican los datos periódicamente. Si se supera el tiempo de Deadline entre muestras, la aplicación es notificada, quedando a decisión del desarrollador las acciones a llevar a cabo en este supuesto.

Lifespan Se refiere al tiempo en el que un mensaje es válido. Si se supera el tiempo para este mensaje, es eliminado de DDS.

Transport Priority Se refiere a la prioridad que recibe un mensaje en la capa de transporte subyacente. Dicho valor es válido sólo si la capa de transporte subyacente soporta prioridad de mensajes.

Para una descripción más detallada de éstas y otras políticas de calidad de servicio, véase [2] ó [3].

Como hemos visto, la manera de desarrollar aplicaciones que usen DDS cambia un poco respecto de una aplicación cliente/servidor. Como muestra se puede observar las diferencias presentes entre las figuras 1.3 y 1.4. Para facilitar el acceso a estas aplicaciones a un usuario lego en la materia, sería conveniente acercarlo un paso más al usuario. La idea de este proyecto es que un usuario pueda acceder a un entorno en el que funciona DDS, pero usando una única herramienta que ya conozca, en vez de tener que instalar nuevas aplicaciones cada vez que quiera acceder a nuevos tópicos. Hoy en día, cualquier usuario es capaz de acceder a un servicio de mensajería y usarlo bastante fluidamente. Por tanto para el objetivo propuesto se pretende aprovechar la facilidad de uso que proporciona una interfaz de este tipo, para que el usuario pueda acceder fácilmente a un sistema existente operando sobre DDS sin necesidad de perderse en detalles técnicos ni tener que aprender nuevas aplicaciones cada vez que necesite acceder a nuevos servicios del mismo.

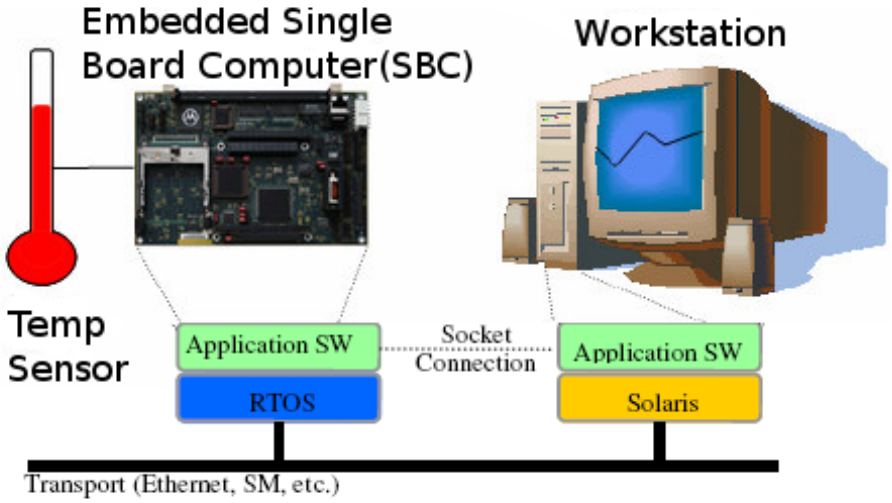


Figura 1.3: Sistema distribuido de datos sin DDS

La mensajería instantánea está basada en el intercambio de texto entre usuarios (aunque como se verá más adelante, en la actualidad es más complejo que un simple intercambio de texto), los cuales pueden tener varios estados visibles para el resto de usuarios. Con el sistema que a desarrollar, el usuario puede acceder a los datos distribuidos (tópicos) mediante un simple intercambio de textos en lenguaje *seminatural*. Con esto se consigue que la curva de aprendizaje sea mínima, y mediante un diseño adecuado se conseguirá además que sea fácilmente extensible sin esfuerzo extra para el usuario.

Es interesante definir una serie de conceptos básicos para facilitar la comprensión del trabajo realizado, tal y cómo se ha hecho para los conceptos relacionados con DDS.

Buddy Es cada uno de los usuarios que aparecen en la lista de contactos de un cliente de mensajería instantánea. En un sistema de mensajería instantánea típico podemos mantener conversaciones con uno o múltiples buddies simultáneamente.

token Se define un token como una palabra o conjunto de símbolos de texto indivisibles y que cuentan con un significado propio. Las sentencias de una gramática se forman mediante una secuencia ordenada de tokens.

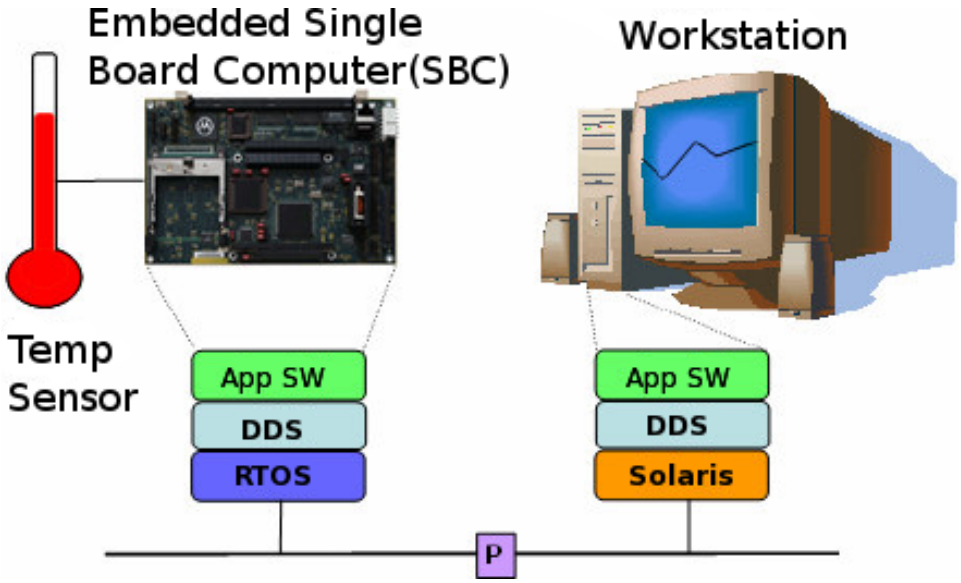


Figura 1.4: Arquitectura de un sistema de distribución de datos con DDS

1.2. Recursos

Para el desarrollo de este proyecto no se necesitan recursos especiales, así como tampoco para la explotación del mismo. Diferenciaremos también los recursos software, hardware y humanos para el desarrollo del mismo.

1.2.1. Humanos

- Prof *D. Juan Manuel López Soler* y *D. Juan J. Ramos Muñoz*, ambos profesores del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, como tutores del proyecto.

- D. Javier Povedano Molina, alumno de la Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones de la Universidad de Granada, que será el encargado de realizar dicho proyecto siguiendo las directrices definidas por los tutores del mismo.

1.2.2. Hardware

Para la realización del proyecto, como se ha mencionado anteriormente no es necesario el empleo de hardware especializado. El hardware empleado en el entorno de explotación tampoco necesita de características especiales. Detallaremos a continuación el utilizado en este caso.

- Ordenador portátil

- Impresora láser
- Conexión a Internet

1.2.3. Software

- Sistema Operativo GNU/Linux Ubuntu 7.01 Feisty Fawn
- Paquete de utilidades GNU
- Paquete procesador de textos LaTeX
- Editor de textos
- Compilador gcc v3.4
- Librería de distribución de datos RTI DDS 4.1e
- Generador de Documentación Doxygen
- Glade-2 para el diseño de Interfaces

1.3. Antecedentes y Estado del arte

Esta parte del documento se dedica a explicar el estado actual de los temas relacionados con el proyecto abordado. Este apartado, se centrará por tanto en hacer un breve recorrido por la historia de la mensajería instantánea y la distribución de datos.

1.3.1. Antecedentes en mensajería instantánea

Lo que hoy se conoce como mensajería instantánea, puede decirse que tiene su origen en los años 70. El primer sistema pionero en el uso de charla en por medio de ordenadores fue *P*LATO [4] . Este sistema, entre otros sentó los conceptos de mensajería instantánea, foros de discusión, juegos multijugador, etc... Entre las características más destacables de cara al futuro, de dicho sistema fue el empleo de listas de contactos y presencia de los mismos.

Casi en el mismo tiempo (en la década de los 70), también apareció el programa *t*alk primero para computadores *D*EC PDP-11, y posteriormente en versiones para sistemas *U*NIX. Este sistema en su versión para *U*NIX fue bastante popular en entornos académicos durante la década de los 80/90.

El Proyecto *A*thena del MIT [5], dio un paso importante al publicar el primer cliente de mensajería basado en un interfaz gráfico (GUI). Fue para el protocolo de mensajería *Z*ephir [6], y se publicó el año 1987. Durante esta época fueron apareciendo cada vez más protocolos de mensajería de texto.

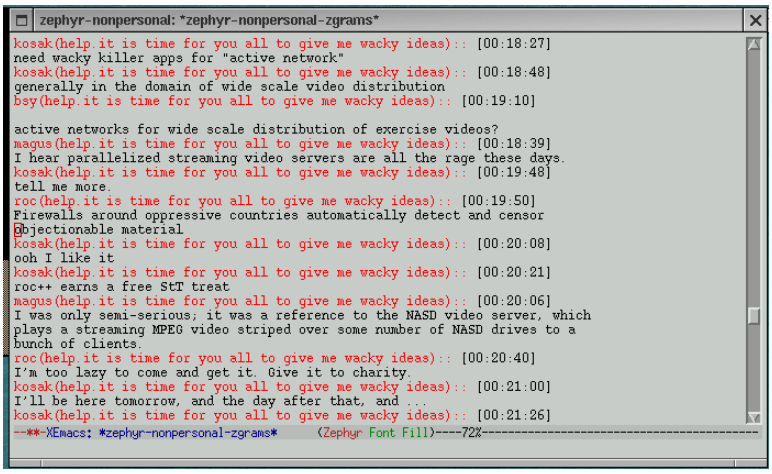


Figura 1.5: Cliente de mensajería instantánea para *Zephyr*

A partir de la década de los 90, estos sistemas de mensajería fueron popularizádose a la vez que lo hacía Internet. Durante esta década, se popularizaron enormemente redes de mensajería instantánea como IRC o ICQ. Si comparamos las figuras 1.5 (Zephyr) y **??**(mIrc), se observa la evolución de la interfaz a lo largo del tiempo.

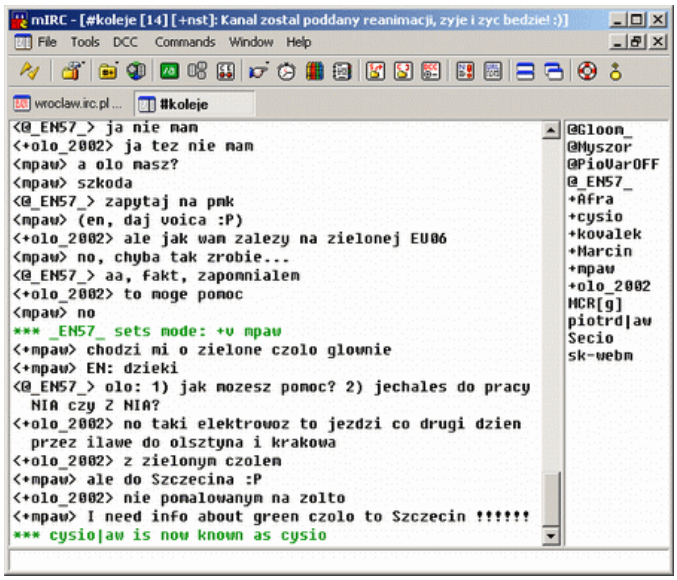


Figura 1.6: Cliente de mensajería instantánea *mirc* para *IRC*

A finales de los años 90, los servicios de mensajería existentes fueron dejando paso a una nueva evolución de los mismos. Los contenidos multimedia y comunicaciones más complejas fueron abriéndose paso. Clientes (redes)

de mensajería como *MSN Messenger* y *Yahoo! Messenger* fueron ganando adeptos. Sus características eran cada vez más componentes multimedia, control de presencia, compartición de ficheros,etc.. Estos sistemas son los que acuñan el significado de mensajería instantánea a día de hoy.

Es evidente la evolución que existe desde los primeros sistemas de mensajería de texto (que se limitaban a intercambiar texto entre clientes conectados), hasta los sistemas actuales, cuyo abanico de funcionalidades es mucho más amplio: intercambio de ficheros, emoticonos personalizados, VoIP, videoconferencia, mensajes *o*ffline, cifrado de datos. ..

Algunos sistemas de mensajería existentes, aprovecharon su carácter de abiertos, para ser extendidos y automatizados para un uso propio. El caso más popular es el caso de IRC, cuyo carácter de estándar abierto, hizo que mucha gente fabricara sus propios *b*ots para realizar tareas automatizadas. Estos bots, simulaban un contacto en un cliente de mensajería, programado para realizar determinadas tareas tales como tomar estadísticas, proteger canales de chat, etc... Este concepto de contacto automatizado será muy útil en el desarrollo de este proyecto.

Según algunos estudios, hoy en día alrededor de 250 millones de usuarios de todo el mundo usan mensajería instantánea de una forma habitual. Se ha convertido en una herramienta de uso bastante generalizado, por lo que no es raro que haya una competitividad alta por dominar ese mercado.

Tras estudiar la evolución histórica de los servicios de mensajería instantánea, en el apartado 1.3.3 se estudia la estructura interna y características destacables de los mismos.

1.3.2. Antecedentes en Distribución de datos

Las sistemas para la distribución de datos geográficamente dispoersos, pueden llegar a ser algo muy complejo, por lo que distintas entidades han creado middleware dedicado a la tarea de facilitar el desarrollo de aplicaciones sobre este tipo de servicios.

Middleware, se refiere al software que media para conectar dos componentes software en un sistema distribuido. La definición exacta según ObjectWeb es:

> "The software layer that lies between the operating system and the applications on each side of a distributed computing system. "
>
> –Krakowiak, Sacha. What's middleware?. ObjectWeb.org. Retrieved on 2005-05-06.

Otra definición, según *Carnegie Mellon SE Institute*

> "Middleware is connectivity software that consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network. "
>
> – Carnegie Mellon SE Institute
> http://www. sei.cmu.edu/str/descriptions/middleware.html

Aunque la utilización del término *m*iddleware se ha popularizado recientemente, tiene bastante antiguedad. La primera referencia al término de la que se tiene conocimiento por parte de los autores, se tiene en [1], un documento que data de 1968 y habla de una entidad de software intermedia. Al paso de los años, el término fue incrementando su uso en los 80, hasta surgir a partir de los años 90 el auténtico boom.

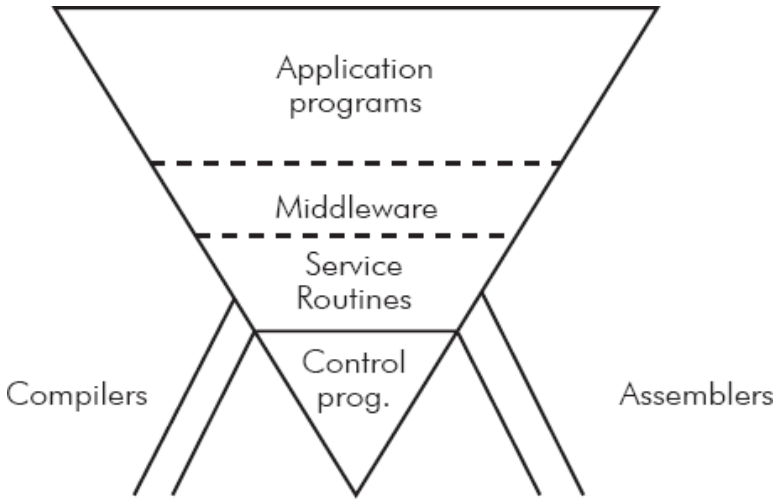


Figura 1.7: Primera referencia al término *middleware*, tal y como aparece en [1]

La empresa RTI comenzó desarrollar DDS, y en Junio de 2003, fue adoptado como estándar de la OMG. Su finalización ocurre en Junio de 2004. La idea por la que se adoptó como estándar era la necesidad de aumentar CORBA . DDS+THALES (desarrolladora de SPLICE(data-cetric architecture) se unieron para crear la especificación que fue aprobada por la OMG .

Tambien exixten otros servicios de distribución de datos, como por ejemplo JMS (Java Message Service), que aparece junto con Java2, y es un middleware Java que soporta tanto el paradigma de productor/consumidor para el paso de mensajes, como el de publicación/suscripción. Se dice que es un Message Oriented Middleware, debido a las características de Java de orientación a objetos.

1.3.3. Estado del arte en Mensajería Instantánea

En la actualidad existen multitud de servicios de mensajería instantánea, cada uno con una serie de caractersticas propias. Progresivamente, cada vez se estudian más protocolos de mensajería distribuida, y cada vez haciendo más énfasis en la seguridad y privacidad. Hay multitud de protocolos, pero el único que anda más cerca de ser estándar (debido a su condición de código abierto) es Jabber. No obstante, existe multitud de protocolos que aunque son más completos que Jabber en capacidades, su condición de código cerrado los alejan de ser estándar.

Normalmente se identifican los clientes de mensajería instantánea con los protocolos: es algo que debemos evitar. En lo que sigue cuando se hable de mensajería instantánea se referirá a los protocolos, no a los clientes. De este modo evitarán las confusiones que pudieran ocurrir.

Sistemas actualmente en producción En la siguiente sección se identifican los sistemas de mensajería instantánea actualmente en producción.

Jabber : Protocolo de mensajería instantánea basado en XMPP (subconjunto de XML) [7][8] que en la actualidad esté tomando una gran relevancia. Es un protocolo de mensajería abierto, hecho que lo hace interesante respecto de la mayoría de sus competidores. El hecho de que sea un protocolo abierto, hace que estén continua expansión, haciendo que cada día aparezcan nuevas capacidades o se mejoren las existentes. Tambien se emplea en Google Talk, el servicio de mensajería instantánea de Google. Las características destacables del mismo son:

Es un protocolo descentralizado, no depende de un servidor central, de hecho cualquier persona con conocimientos medios puede instalar un servidor de Jabber.

Es un protocolo extensible, debido a que es abierto, cualquiera puede ampliar sus funcionalidades para ofrecer nuevos servicios dentro del mismo. Muchos servidores amplian sus capacidades ofreciendo nuevos servicios (que dicho sea de paso deben de estar soportados por los clientes empleados)

Existen pasarelas que permiten la interconexión con otros protocolos. No es de extrañar que podamos comunicarnos con contactos con MSN messenger.

MSNP : Posiblemente es el protocolo de mensajería instantánea más empleado. Se ha popularizado por el programa MSN Messenger (actualmente Messenger Live 8) Fue desarrollado por Microsoft, y en la actualidad soporta una gran cantidad de funcionalidades, tales como transmisión de voz, vídeo, imágenes, mensajes offline,... La versión más reciente del protocolo es la MSNP10, que es un protocolo cerrado.

GG : Protocolo de mensajería instantánea muy popular en Polonia. Va por delante de messenger y de jabber en cuanto a cuota de mercado en dicho país. En el resto de países la presencia es casi nula. El cliente que se emplea para utilizarlo es el llamado Gadu-Gadu. Es un protocolo propietario como la mayoría de protocolos que estudiamos. Como características destacables se encuentran: mensajes offline, data-dispatch, soporte de VoIP y en las ltimas versiones se soportan conexiones SSH para aumentar la seguridad de la conexión. Es un protocolo centralizado y al igual que OSCAR (ver más adelante) [9] usa números PIN para identificar a los usuarios.

OSCAR : Protocolo de mensajería instantánea y información de presencia empleado por redes de mensajería instantánea como son ICQ y AIM. Es un sistema propietario y que en la actualidad ha sido entendido (en parte) por ingenierï¿œ inversa.

YMSG : Protocolo de mensajería instantánea empleado actualmente por la red de Yahoo! Messenger. Es un protocolo propiedad de Yahoo!, y que por tanto tiene las mismas desventajas de los protocolos cerrados (dependencia de una entidad, centralización,...) Por lo demás es un protocolo bastante completo en cuanto a servicios que ofrece y que se emplea mucho en la actualidad. Un hecho destacable es que recientemente permite interactuar con contactos de otros protocolos como MSNP, por medio de la existencia de pasarelas.

IRC : Cada vez más en desuso. Sistema de chats basado en salas. En los inicios de Internet, este sistema de chats alcanzó una gran relevancia, debido a que fue el primer sistema de chats para el gran pblico (hasta entonces los chats eran casi monopolio de los grandes sistemas Unix). Alcanzó un gran éxito en los años 90 junto con ICQ.

Aunque aún se emplea este servicio de mensajería, ha perdido bastante terreno, en parte a las limitaciones del mismo. Es un protocolo abierto [10], lo que hizo que existieran bastante clientes y fuera portable a multitud de arquitecturas con facilidad. Los servicios que permitían eran bastante limitados, ya que el protocolo solo consideraba el texto plano como forma de comunicación, aunque también eran posibles el envío de archivos. Debido a la relevancia que tuvo en su momento, y a que en la actualidad aún se emplea hemos considerado oportuno mencionarlo.

ICQ : Podríamos decir que este sistema es uno de los precursores de los actuales servicios de mensajería instantánea. La arquitectura del mismo es bastante primitiva, lo que hace que aunque en parte fuera descentralizado, fueran bastante comunes los splits, debido a la falta de una infraestructura que proporcionara redundancia al mismo.

Skype : Un protocolo en la actualidad muy empleado. Orientado a obtener comunicación por medio de VoIP, aunque tambien permite mensajes de texto. Como inconveniente, hay que destacar que es un protocolo propietario y centralizado, ya que depende de un servidor central. Lo atractivo del mismo es que se puede considerar que fue el primer protocolo de mensajería instantánea orientado directamente y principalmente a la comunicación mediante VoIP.

Sistemas Experimentales en mensajería instantánea

Son sistemas que no se encuentran disponibles al gran público, ya sea por que son sistemas de mensajería instantánea privados, o bien por que hayan sido creados con motivos de investigación

En muchos casos son propuestas de sistemas de mensajería instantánea que han sido creados con para la investigación en el seno de Universidades, o por el mero hecho de estudiar alternativas posibles a los problemas inherentes a muchos servicios de mensajería instantánea.

DAPIM [11] propuesto por la Universidad de Kent (USA), es una propuesta de un protocolo de mensajería instantánea que se basa en tres principios básicos: distribución, anonimicidad, y privacidad. Los resultados obtenidos fueron satisfactorios.

Onobee [12]: de mano de la empresa PPServer, se ofrece este producto que es una solución empresarial, donde se aplica el concepto de descentralización para ofrecer mensajería instantánea, así como teleconferencia, compartición de archivos, pizarras colaborativas, compartimiento de programas, etc. ..

Hush messenger [13] este servicio de mensajería, está basado en un protocolo propio de la empresa Hushmail, y se encuentra orientado casi principalmente a la privacidad. Se combina la mensajería instantánea con cifrado Pretty Good Privacy (PGP), para ofrecer al usuario una protección extra en sus conversaciones.

Lakaim [14] :experimento desarrollado en Java, para construir un servicio de mensajería distribuida. Para ello se ha desarrollado sobre la librería jChord, que implementa el protocolo de descubrimiento distribuido Chord [15].

1.4. Fases de Desarrollo

Como todo proceso de desarrollo de software, se realizará en una serie de fases bien diferenciadas. Debido a la propia naturaleza del proyecto, las fases consideradas necesarias son las siguientes.

1. **Revisión** del estado del arte: en esta fase se realizará un estudio de DDS, sus implementaciones, así como de los sistemas de mensajería instantánea, para ello se utilizarán todos los recursos bibliográficos y electrónicos disponibles.

2. **Desarrollo** de un lenguaje de interacción entre el usuario y las entidades DDS. Durante esta fase del proyecto se caracterizarán las funcionalidades que cabe esperar de la interfaz y su materialización en la especificación de un lenguaje y su gramáica, de forma tal que permita dialogar al usuario con las distintas entidades DDS (remotas) de la red.

3. **Implementación**: una vez especificado el lenguaje durante esta fase se desarrollará una aplicación del tipo de mensajería instantánea que evidencie las potencialidades del lenguaje y gramáticas desarrolladas.

4. **Evaluación**: durante esta fase se evaluarán las prestaciones de las herramientas desarrolladas en entornos reales, valorando especialmente su flexibilidad.

5. **Documentación**: coincidente en el tiempo con las etapas anteriores, se procederá a la generación de la documentación de los resultados pertinente, sin descartar la publicación de los mismos.

1.5. Restricciones

A la hora de desarrollar el proyecto, se hacen necesario establecer una serie de restricciones que limitan el desarrollo del mismo, y por tanto influyen a la hora de la toma de decisiones. En este caso se han tenido en cuenta las siguientes consideraciones.

1.5.1. Factores Dato

Los factores dato, son aquellos que vienen dados de forma ajena al desarrollador de la aplicación y que deben de cumplirse. En este caso se identifican los siguientes:

- La aplicación debe de ser *portable*, es decir debe de ser fácilmente adaptada a diversas arquitecturas y sistemas operativos.

- La aplicación debe de ser de *código abierto*, o al menos extensible mediante una API de extensión bien definida

- La aplicación debe de poder soportar de forma sencilla nuevos tópicos

1.5.2. Factores Estratégicos

Los factores estratégicos, son aquellas decisiones iniciales tenidas en cuenta por parte del desarrollador por diversas razones y que no vienen impuestas externamente, como ocurre en el caso de los factores dato.

Elección de entorno de desarrollo Se ha elegido un entorno libre libre como GNU/Linux, ya que se considera que el software libre proporciona herramientas más adecuadas para el desarrollo de proyectos de este tipo, incurriendo así en un ahorro en software. Además, disponer del código de muchos programas, hacen que sea más fácil el desarrollo.

Elección de la estrategia de desarrollo Crear un proyecto desde cero, es algo que sólo es adecuado en determinadas circunstancias, especialmente cuando el coste de adaptación del proyecto en esfuerzo es igualable a comenzar el proyecto desde cero. En este caso, debido a la gran cantidad de aplicaciones de mensajería instantánea existentes, consideramos que era más lógico estudiar si alguna aplicación de mensajería de código abierto podía ser adaptada fácilmente a los requisitos. La elección de la arquitectura, debe respetar todos los requisitos a cumplir indicados en la sección 1.5.1. Estas restricciones limitaban el rango de búsqueda, quedando las siguientes opciones:

Miranda IM [16] Cliente de mensajería instantánea de código abierto y multiprotocolo. Actualmente, sólo es soportado por plataformas Windows. No es muy popular en la actualidad debido a la escasa repercusión que tiene el software libre en Windows. Además de ser de código abierto, es extensible mediante plugins.

Trillian [17] Cliente de mensajería multiprotocolo. Es de código propietario, pero dispone de una API que permite extender su funcionalidad. De uso más extendido que Miranda IM. Como inconveniente, hemos de decir que en la actualidad sólo es soportado por Windows y que la versión de desarrollador no es gratuita como ocurren en otros clientes de mensajería.

Gaim [18] Cliente de mensajería muy popular en entornos GNU/Linux. Es multiprotocolo y de código abierto. Su uso es muy extendido, de hecho muchas distribuciones de GNU/Linux lo incluyen por defecto. Además de esto ha sido desarrollado con la portabilidad en mente, por lo que es soportado por Windows.

A tenor de la comparativa anterior, se ha tomado la decisión de elegir el cliente Gaim, por ser el que más se adapta a los requisitos. Las razones de dicha decisión son las siguientes:

- Código *abierto* lo que facilita la labor de adecuarlo a nuestras necesidades
- Disponible en varias plataformas, lo que aumenta la *portabilidad* a múltiples sistemas
- Dispone de una API pública bien documentada
- Dispone de un *sistema de plugins*, que facilita la ampliación de funcionalidades de una forma casi transparente.
- Dispone de versiones sin instalación, lo que lo hace fácilmente transportable en dispositivos de almacenamiento como dispositivos *extraíbles*
- Soporta gran cantidad de protocolos de mensajería instantánea, algo que coincide con la filosofía integradora del presente proyecto
- No necesitara disponer de un *servidor dedicado* para que el sistema funcione, lo que posibilita su instalación en *intranets* sin conexión hacia el exterior
- Separación del código de interfaz gráfica (GUI) y mensajería instantánea (IM)

Figura 1.8: Logo de *Gaim*

En sucesivos capítulos, se verán más en detalle muchas de estas características de Gaim, y las ventajas que ellas suponen. Así por ejemplo la separación entre GUI e IM, hacen que el diseño de la aplicación sea mucho más sencillo, evitando así tareas costosas en tiempo como diseño y prueba de interfaces. Gracias a esto los esferzos se centrarán en diseñar y probar únicamente la funcionalidad del proyecto y no otros requisitos no tan relacionados tan directamente.

1.6. Objetivos

Para finalizar el capítulo de introducción, en este último apartado se centrará en detallar los objetivos que persigue este proyecto.

Integración entre Mensajería Instantánea/Sistema de Distribución de Datos (DDS) Como se ha mencionado anteriormente, tanto los sistemas de mensajería instantánea como los sistemas de Distribución de Datos en tiempo real, está cobrando cada vez más importancia. El objetivo de este

proyecto es utilizar ambos en un único sistema para poder tener un control del mismo sin tener que disponer de múltiples programas de control. Se podrá así gestionar fácilmente múltiples entidades de un sistema de control empotrado, desde un cliente de mensajería. Se considera, por tanto, éste el objetivo principal de este proyecto.

Acercamiento de los sistemas de distribución de datos al usuario no experto Como se ha mencionado anteriormente, debido a la dificultad de los sistemas de distribución de datos, hace que en la actualidad se su uso e interacción esté restringido a un público muy determinado debido a su complejidad. Como objetivo secundario, se propone por tanto hacer más accesible a los usuarios el uso de servicios de distribución de datos, usando para ello algo ampliamente conocido por los usuarios.: la mensajería instantánea.

CAPÍTULO 2

Especificación de Requisitos

Este capítulo tiene como objetivo identificar el conjunto de requerimientos que el sistema a desarrollar debe de cumplir, diferenciándose en dos tipos:

- **Requisitos funcionales**, que son los que se encuentran orientados a la **funcionalidad** que debe desempeñar la aplicación de cara al usuario final.
- **Requisitos no funcionales**, orientados al **funcionamiento** final de la aplicación. Son requisitos que no tienen que ver con la funcionalidad de la aplicación, sino como se verá más adelante, están más orientados a características que debe cumplir el funcionamiento de la aplicación.

El sistema desarrollado, como se ha mencionado en secciones anteriores, consiste en una extensión a un cliente de mensajería para monitorizar y gestionar un entorno en el cual se emplea el Servicio de Distribución de Datos (DDS), recientemente aceptado como estándar por la *OMG* . Es por esto, que sólo se tendrá en cuenta al usuario del cliente de mensajería para determinar dichos requisitos.

2.1. Requisitos Funcionales

Consulta de tópicos Una de las funciones básicas del sistema (podríamos decir que casi la razón de ser) es la de monitorizar los tópicos que se publiquen en un entorno DDS. Este acceso de consulta a los tópicos se realizará mediante una gramática lo más parecida al lenguaje natural posible, y con la que se realizará la comunicación con los *buddies* del cliente de mensajería instantánea. La respuesta proporcionada a las preguntas/peticiones

formuladas por el usuario se realizarán en un lenguaje próximo al natural, y devolviendo los datos en un formato legible por un humano.

Consulta del entorno Un entorno DDS puede llegar a ser bastante complejo. Pueden publicarse multitud de tópicos en el mismo, en dominios distintos, etc... El usuario tiene que tener alguna forma de poder acceder a dicha información para saber qué es lo que está monitorizando/consultando. Debe de tener acceso a los dominios en los que participa, los tópicos que reconoce el sistema, etc...

Extensión de la aplicación La idea es que nuestra aplicación sea fácilmente extensible, para poder albergar nuevos tópicos dentro de la infraestructura diseñada. Esta extensión se debe realizarse de la manera más transparente posible para el usuario. Esta es la tarea más complicada, debido a la cantidad de tópicos distintos que pueden existir en un entorno DDS.

El usuario debe de poder añadir nuevos tópicos dentro del sistema de mensajería instantánea sin tener que volver a construir todo el sistema. Para ello, se hará uso de una arquitectura basada en plugins, de modo que instalar un nuevo tópico en el sistema, sólo suponga copiar un único archivo en una determinada ruta, siendo así reconocido por la aplicación.

Modificación de las políticas de suscripción/publicación Una de las ventajas de un entorno DDS, es que es debe de ser muy flexible en cuestión de *calidades de servicio* (QoS) . La API de DDS proporciona métodos para modificar las políticas de calidad de servicio. El diseño a realizar debe de proporcionar al usuario un fácil acceso a dichas políticas, y posibilitar la modificación de las mismas de una manera fácil para el usuario. Para ello se deberá habilitar una parte de la gramática para la gestión de políticas de calidad de servicio.

Debido a las características particulares de cada política de calidad de servicio, no todas son susceptibles de ser modificadas desde el sistema por diversos motivos. Tras estudiar las características de todas las políticas y de la arquitectura de DDS hay que decidir las políticas. Para ello hemos tenido en cuenta dos características:

1. Las políticas modificables desde la interfaz diseñada sistema deben de ser modificables dinámicamente.

2. Al ser el diseño un sistema extensible por medio de plugins orientados a tópico, sólo las políticas soportadas por los tópicos deben de ser soportadas.

Teniendo en cuenta estas premisas, las políticas QoS candidatas para ser modificadas (ver página 16), son:

- Lifespan
- Deadline
- Transport Priority

Dichas políticas deben de permitirse. Sin embargo, esto no significa que el resto de las políticas no puedan ser modificadas, sino que deben habilitarse mecanismos que permitan el acceso al resto de políticas que puedan ser de interés al usuario mediante acceso la configuración del plugin con una interfaz.

Gestión de los Buddies La comunicación con el sistema DDS, se realizará mediante *buddies*. Estos deben de permitir una configuración/organización configurable al gusto del usuario, para clarificar la interacción con los mismos. Debe de ser posible agrupar los tópicos en un *buddy*, para así disminuir el número de *buddies* necesarios en caso de disponer muchos tipos de tópicos distintos suscritos.

2.2. Requisitos no Funcionales

Los requisitos no funcionales, son aquellos que definen *cómo* funciona un sistema, en lugar de centrarse en *qué* hace un sistema.

El sistema está diseñado para facilitar a un usuario lego en la materia, la interacción con un sistema de distribución de datos, usando para ello el paradigma de la mensajería instantánea. Es por esto que los requisitos no funcionales de nuestra aplicación, estarán concebidos para satisfacer al usuario final principalmente.

Facilidad de uso El sistema no debe de diferir en su uso de cualquier cliente de mensajería normal para no despistar al usuario. El usuario no tiene por qué conocer los entresijos de un servicio de distribución de datos, para así utilizar con éxito la aplicación. Basta con que sepa utilizar un programa de mensajería instantánea.

Transparencia Las operaciones de nivel de distribución de datos, deben de realizarse de manera transparente al usuario. El descubrimiento de entidades, la actualización de instancias de tópicos, etc... serán realizadas sin intervención del usuario que usa la aplicación.

Fácil instalación La instalación del programa para ser integrado dentro de una red de distribución de datos mediante DDS, debe de hacerse con el menor impacto posible. Dicho de otra manera, los cambios sobre el servicio de distribución de datos en explotación deben de ser mínimos. Los cambios

no deben implicar más allá de instalar un cliente de mensajería instantánea y alguna extensión para el mismo. Las entidades del servicio de distribución de datos no deben de ser modificadas, así mismo debe evitarse también la instalación de servicios adicionales.

Portabilidad La portabilidad es un factor importante en el diseño del proyecto. Al igual que la plataforma elegida para el desarrollo del mismo se encuentra disponible en múltiples plataformas, el cliente de mensajería a diseñar también debe igualmente ser portable a múltiples plataformas, o al menos ser portable lo más directamente posible. Hoy en día gran cantidad de programas son portables, por lo que es un requisito fácilmente asumible.

CAPÍTULO 3

Análisis

En este capítulo, se realiza un análisis de la aplicación a diseñar. Este es el siguiente paso una vez identificados los requisitos y objetivos a alcanzar. Veremos por ejemplo, a un nivel mayor de detalle, cómo debe orientarse el proyecto, describir la funcionalidad del mismo a un nivel de menor abstracción, etc... Dicha información, que ha sido recavada durante esta fase, será utilizada en las siguientes fases de desarrollo del proyecto.

3.1. Gramática de interacción

Como ya ha mencionado en capítulos anteriores, el proyecto que se está abordando consiste, en realizar un interfaz de interacción entre entidades DDS y humanos por medio de mensajería instantánea. El método de interacción elegido es texto escrito como si se estuviera dialogando con un contacto de mensajería instantánea "virtual". Es por esto que será necesario especificar el lenguaje reconocido el sistema.

La gramática debe de proporcionar mecanismos que hagan que el acceso a un sistema DDS por parte del usuario sea lo más sencillo posible. Además de ser una gramática sencilla, debe de ser intuitiva, para que la curva de aprendizaje sea mínima. Deben de soportarse las siguientes operaciones:

- **Consulta de instancias de tópicos**: Es una de las operaciones más importantes, ya que es una de las razones de ser de la aplicación. Nótese que se debe poder acceder a las instancias de un mismo tipo de tópico por medio de una clave única
- **Consulta del estado del sistema**: El enfoque adoptado debe permitir el acceso a información genérica de DDS, como por ejemplo saber en

qué dominios se dispone de *DomainParticipants* activos, que tópicos somos capaces de reconocer, etc. ..

- **Consulta de políticas de calidad de servicio (QoS)** Una de las características esenciales de un sistema DDS es la existencia de una serie de políticas de *calidad de servicio*. Es por esto que no se deben olvidar a la hora de diseñar la gramática. Deben incluirse operaciones de consulta de *calidad de servicio*. Se considerarán las políticas de calidad a nivel de *tópico*.
- **Modificación de las políticas de calidad de servicio (QoS)**: Ya que el sistema es capaz de acceder a las *calidades de servicio* y cambiar algunas de estas en tiempo real, deben incorporarse mecanismos que permitan cambiarlas por medio de la gramática.
- **Operaciones genéricas de interfaz**: En esta categoría se incluyen a las operaciones genéricas como por ejemplo un soporte de ayuda a la gramática, operaciones de ayuda, operaciones de consulta de versión actual, etc. ..

Definición de la gramática Una vez que ya quedaron definidas las operaciones que debe soportar la gramática, se debe definir esta de modo que las soporte todas. Como se ha mencionado anteriormente, la definición de la misma se realizará mediante notación *BNF* para hacer más claro su entendimiento y hacerla fácilmente entendible. En la figura **??** se muestra la gramática definida.

```
<sentence> ::= <get-sentence> | <list-sentence>
       | <set-sentence> | <help-sentence> | VERSION
<get-sentence> ::= GET <topic_name> WITH KEY <key_value>
        | GET <topic_name> QoS
<set-sentence> ::= SET <topic_name> <qos-type> TO <integer>
<qos-type> ::=
<list-sentence> ::= LIST <listable-category>
<listable-category> ::= DOMAIN | TOPICS
<help-sentence> ::= HELP | HELP <help-topic>
<help-topic> ::= GET | SET | QOS | LIST | TOPICS
<integer> ::= [0-9]*
<topic_name> ::= [a-z]*
```

Figura 3.1: Gramática BNF definida para nuestra aplicación

El siguiente paso es identificar cada tipo de sentencia con cada una de las operaciones que debe soportar nuestra aplicación.

`<sentence>` Es el símbolo inicial de la gramática. Cada linea de texto escrita en una ventana de conversación se considerará como una sentencia.

`<get-sentence>` Como se observa en la definición de gramática, este tipo de sentencias se emplea tanto como para consultar el valor de instancias como para consultar el valor de políticas de *calidad de servicio*. En el caso de consulta de instancias, se necesitará el valor de la clave única de la instancia.

`<set-sentence>` Esta sentencia es la encargada de gestionar los valores de las *políticas de calidad de servicio*. Para cambiar el valor de la política de calidad de servicio, necesitamos saber qué política se desea cambiar (`<qos-type>`), y el valor a asignarle (`INTEGER`). Solo se han incluido las políticas modificables dinámicamente por sencillez.

`<help-sentence>` Operaciones genéricas de interfaz que permiten al usuario tener acceso a ayuda tanto de la sintaxis de la gramática como a conceptos generales de DDS. Gracias a esta sentencia se evita que el usuario se sienta perdido en algún momento.

`VERSION` Igual que la sentencia `<help-sentence>`, pero para consultar la versión de DDS.

3.2. Diagrama de clases de análisis

En la figura **??**, se muestra una primera aproximación a lo que es el diagrama de clases de análisis.

Entidades observadas en el diagrama

GaimBuddy Cada uno de los contactos que tenemos en el programa de mensajería instantánea. En este caso, cada Buddy se encuentra asociado a una serie de Tópicos (Topics) y sus respectivas Instancias.Es una clase proporcionada por Gaim.

GaimAccount Cada una de las identidades utilizadas en un programa de mensajería es lo que llamamos un *GaimAccount*. Cada usuario puede tener varias *GaimAccounts*, y cada una de ellas se encuentra ligada a un protocolo de mensajería instantánea. Es una clase proporcionada por Gaim.

Topic Cada uno de los distintos tipos de datos presentes en DDS es lo que se llama tópico. En este caso llamamos Topic a una representación interna de un DDS Topic.

TopicInstance Cada tópico tiene un conjunto de instancias asociadas que se diferencian entre sí por el valor de una clave.

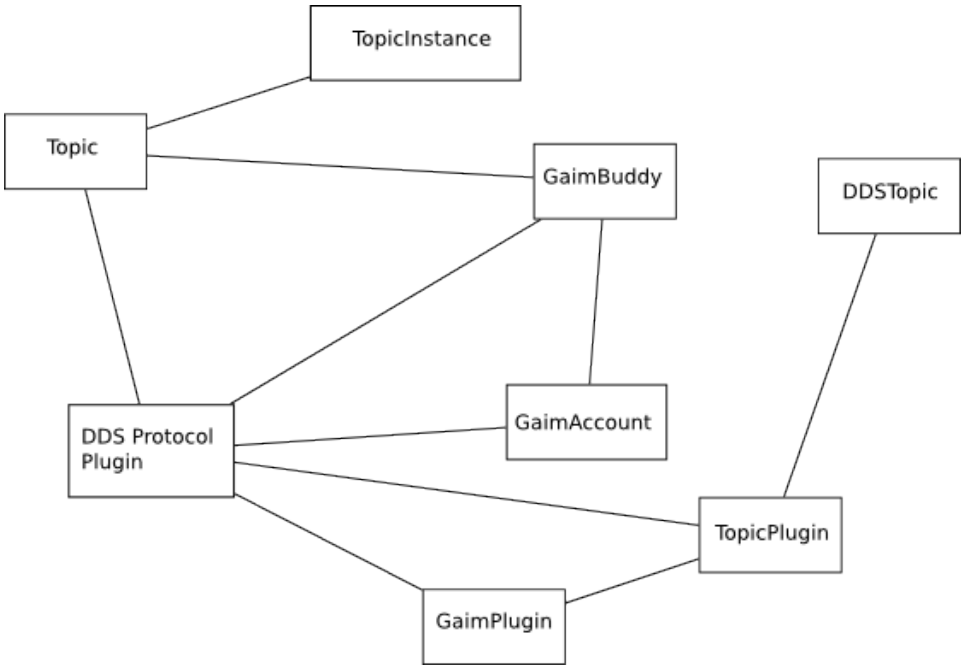


Figura 3.2: Diagrama de clases de análisis

GaimPlugin Es un tipo genérico de plugin a partir del cual derivan los demás. Cada plugin tiene un identificador que lo identifica únicamente.Es una clase proporcionada por Gaim.

DDS Protocol Plugin Es un tipo de plugin encargado de gestionar el protocolo de mensajería instantánea de consulta de DDS.

Topic Plugin Plugin encargado de registrar, gestionar y desregistrar un tipo de DDS Topic en el sistema

DDS Topic Topic implementado por DDS. Cada tipo de tópico es manejado por un plugin de tópico.

En vista de las definiciones anteriores, y del diagrama **??** se extraen una serie de conclusiones que explican de una manera más clara la estructura de la aplicación.

- El Usuario debe de crear una cuenta asociada al plugin de protocolo DDS
- Por cada nuevo tópico a reconocer, se debe cargar un Topic Plugin apropiado en el sistema
- Las clases Topic y Topic plugins son representaciones internas de la clase DDS Topic

- Cada Buddy se encuentra asociado a una serie de Topics.
- Las consultas acerca de una instancia de tópico se realizan por medio de una conversación con el Buddy asociado a dicho tópico.

3.3. Casos de uso

En la figura 3.1 se muestra el diagrama de casos de uso, en el que se identifican los diferentes escenarios que se pueden encontrar a un nivel alto de abstracción. Además de los escenarios se muestran otras relaciones como por ejemplo los actores que forman parte en cada uno de los escenarios.

Cada caso de uso tiene un escenario principal, que es el desarrollo normal de un proceso cuando no hay ningún tipo de error. Si aparecen errores en el proceso cada caso de uso presentará una serie de excepciones. A tenor de lo que se muestra en la figura 3.1, se identifican los siguientes casos de uso:

Inicia sistema El usuario quiere acceder al servicio de mensajería instantánea para entidades DDS y lo hace por primera vez. Este caso de uso está dedicado a inicializar todas las estructuras y entidades presentes.

Carga tópico En este caso se desea que que el programa reconozca un nuevo tópico. El usuario solicita el registro de dicho tópico en el sistema para poder ser accedido.

Descarga tópico Este caso de uso sirve para lo contrario que el anterior, y se verifica cuando no se desea hacer uso de un tipo de tópico en el sistema.

Actualiza instancia Este caso de uso se desencadena cuando se actualiza una instancia de tópico en DDS y hay que proceder a la actualización de las estructuras internas.

Cambia QoS Es un caso de uso abstracto que se refiere a la acción de cambiar la política de calidad de servicio de un tópico determinado.

Cambia QoS Lifespan Caso de uso concreto de *cambia QoS* en el caso de querer cambiar la política *lifespan* de un tópico.

Cambia QoS durability Caso de uso concreto de *cambia QoS* en el caso de querer cambiar la política *durability* de un tópico.

Cambia QoS transport priority Caso de uso concreto de *cambia qos* en el caso de querer cambiar la política *transport piority* de un tópico.

Notificar Evento En este caso de uso se avisa al usuario de un evento ocurrido en DDS mediante un mensaje que le llega mediante una notificación por medio de un buddy de mensajería instantánea.

Consulta Instancia Caso de uso que es iniciado cuando el usuario desea conocer datos acerca de una instacia de un tópico.

Consulta Dominio Caso de uso que es iniciado cuando un usuario solicita información acerca de un dominio de DDS.

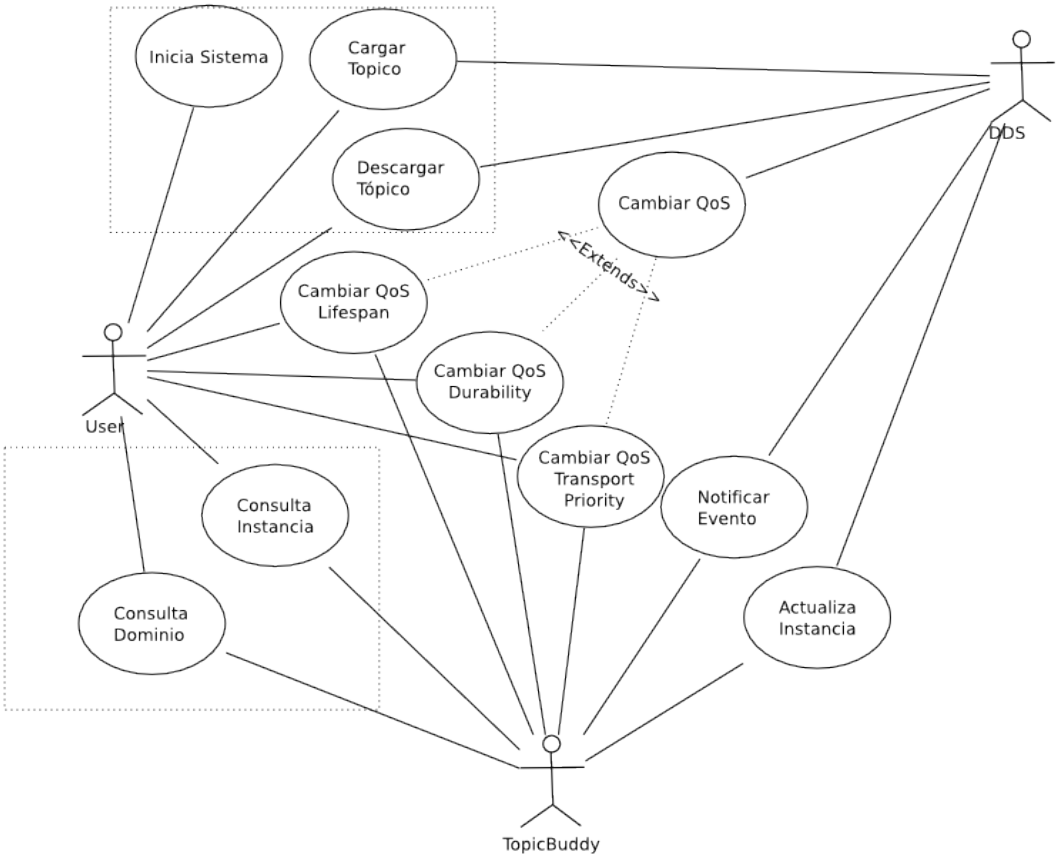


Figura 3.3: Diagrama de Casos de Uso

3.3.1. Identificación de Actores

Buddy Este actor se identifica con la definición que se dió del mismo en la página 18. En este caso, un *buddy* es usuario *virtual* ya que no se corresponden con un usuario *real*. Cada *buddy* puede encontrarse en varios estados (como ocurre en casi cualquier cliente de mensajería) y puede estar asociado a uno o varios tópicos.

DDS Middleware El rol de este actor se identifica con el software encargado de gestionar la funcionalidad relacionada con un servicio de distribución de datos. Conceptualmente, decimos que cada *TopicBuddy* mantendrá contacto con el actor *DDS Middleware* para realizar las gestiones pertinentes.

Usuario Se identifica con el usuario de la aplicación, que en este caso es la persona que quiere interaccionar con el servicio de distribución de datos. El usuario interactúa con los *buddies DDS* para obtener información de las *entidades DDS*.

rtidds-prpl Es un plugin de tipo *protocolo de mensajería* . Se encarga de construir un servicio de mensajería para acceder a un servicio DDS, para ello define las operaciones necesarias. Sus funciones además consisten en controlar los buddies y reconocer/generar los mensajes que se intercambian con el usuario.

topic-plugin Es un plugin encargado de gestionar un determinado tópico en el sistema. Este actor gestiona las políticas de dicho topic, así como ser el encargado de recoger de DDS las instancias de dicho tópico.

CAPÍTULO 4

Diseño

Tras la fase de análisis, abordada en el anterior capítulo, en la que se identifica *qué* se quiere hacer, el siguiente paso es *cómo* se debe hacer. Durante este capítulo entraremos en los detalles más relevantes del diseño de nuestra aplicación. Para estructurar y facilitar el diseño, el problema se enfoca considerando.

- La Arquitectura del sistema
- Las Operaciones del sistema
- El Despliegue de la operación

4.1. Arquitectura del sistema

Una de las partes más importantes del diseño de la aplicación, es el diseño de la arquitectura de la misma. En este caso, la aplicación a desarrollar ya existe, y por lo tanto ya existe una arquitectura de la misma. El propósito a perseguir es *extender* una aplicación para dotarla de funcionalidad por medio de *plugins*.

La aplicación de mensajería *Gaim*, dispone de una arquitectura particular que le proporciona entre otras características portabilidad y extensibilidad. Dicha aplicación se encuentra dividida en dos capas bien diferenciadas que son:

gaim-core (libgaim) Es la parte que gestiona la aplicación de mensajería instantánea. Podría decirse que es el núcleo de la misma, y en ella se llevan a cabo tareas como por ejemplo: la gestión de protocolos, gestión de plugins, conexiones a cuentas de mensajería, gestión de IPC...

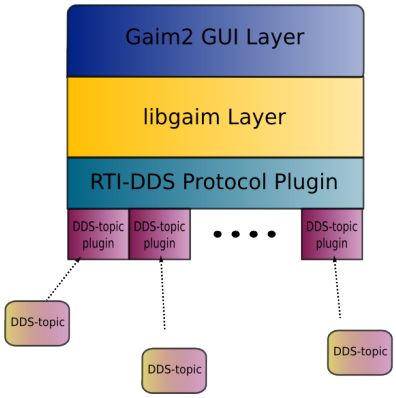


Figura 4.1: Propuesta de diseño de la arquitectura

gaim-gtk (gui layer) Es la capa que corresponde a la interfaz de usuario con la capa *gaim-core*. En ella se gestionan las ventanas de conversación, de configuración, lista de contactos, entre otras.

Gracias a esta división en la arquitectura se han podido desarrollar aplicaciones de mensajería en diversas arquitecturas con un esfuerzo mínimo (respecto a lo que supondría desarrollar una aplicación de mensajería completa). Como ejemplo baste mencionar a *Adium*, cliente de mensajería instantánea para plataformas *MacOs X* que usa el núcleo de *Gaim*, o el ejemplo de *gaim-text*, que posibilita conectarse a un servicio de mensajería instantánea en consolas de texto, útil por ejemplo en escenarios donde no es posible el uso de interfaces gráficas.

Arquitectura de plugins Además de esta arquitectura de grano grueso, hay una estructura subyacente que además es interesante detallar, que es la arquitectura de plugins de la aplicación. Debido a la naturaleza del presente proyecto, esta es la arquitectura de más interés. Situada dentro de la capa *gaim-core*, en ella se definen varios tipos de plugins según su funcionalidad, en particular se definen:

Protocol Plugin Se llaman así por proporcionar al cliente de mensajería la habilidad de soportar nuevos protocolos de mensajería instantánea. Este tipo de plugins no deben incluir código de interfaz de usuario, para mantener la diferenciación entre UI y IM que establece gaim en su estructura interna. Gaim soporta gran cantidad de protocolos por defecto, como por ejemplo: Msn, yahoo!, Jabber, Gadu-Gadu, Irc,...

UI Plugin Se encargan de extender la interfaz gráfica de usuario. Por ejemplo, si se desea crear nuevas ventanas de configuración, se debe usar este tipo de plugin.

Core Plugin Añaden funcionalidad genérica a la aplicación, añadiendo funcionalidad genérica a la misma. Por ejemplo: se pueden añadir acciones a un protocolo mediante un plugin de este tipo, generar mensajes de auto-respuesta, cifrado de mensajes,...

Además de existir diversos tipos de plugins, la arquitectura de plugins presenta una serie de características destacables y que pueden ser aprovechadas en el de diseño de la aplicación:

- Gaim emplea un sistema de dependencias entre plugins, que no permite la carga de un plugin si las dependencias no fueron satisfechas. Esto proporciona robustez y la posibilidad de crear plugin que cooperan conjuntamente.

- Se puede modificar el comportamiento de la aplicación mediante plugins empleando un sistema de señales emitidas por el programa. Creando *callbacks* para dichas señales modificamos el flujo de la aplicación. Así por ejemplo podemos realizar una acción cada vez que se escribamos un mensaje en una conversación de mensajería instantánea. (Para mayor información del sistema de señales, véase la API de Gaim [18]). Una señal además recibe una serie de parámetros que se encuentran asociados a la misma, de modo que se puedan realizar acciones relacionadas con la misma. Conociendo por los capítulos anteriores el comportamiento de la aplicación se identifican las siguientes señales que deben ser capturardas por la aplicación:

  - **sent-message** Esta señal es emitida cada vez que es enviado un mensaje en una ventana de conversación con un buddy, y entre otros parámetros tiene el destinatario, y el mensaje. El interés de esta señal es capturar la conversación del usuario con el buddy para realizar las acciones pertinentes.
  - **conversation-created** Esta señal se emite cada vez que se crea una nueva conversación. Esta señal es capturada para mostrar al usuario de que se encuentra conversando con un buddy automatizado (bot). De este modo el usuario es consciente de que es una respuesta automatizada, y por tanto debe atenerse a ciertas reglas sintácticas.

- Los plugins en Gaim permiten comunicarse entre sí. Gaim define un mecanismo interno de comunicación entre plugins mediante una arquitectura IPC(Inter Processes Communication). Un plugin puede solicitar un método de otro, para utilizar la información devuelta para su uso interno. Gracias a esta característica, junto con las otras anteriores, los plugins pueden ser 100 % cooperantes.

En vista de todos los datos anteriores, el enforque adoptado consiste en desarrollar la aplicación como un conjunto de plugins que trabajan cooperativamente para proporcionar la funcionalidad deseada. Dividiremos por tanto el subsistema en dos capas: Una capa dedicada a gestionar las tareas de DDS, y otra dedicada a la interacción con el usuario. Con este diseño, se consigue que para ampliar nuestra aplicación con nuevos tópicos, bastará con añadir un plugin que diera soporte para el nuevo tópico. En la figura 4.1 se puede observar la arquitectura propuesta. En la sección 4.4 se proporcionarán más detalles de la arquitectura diseñada.

4.2. Contratos del sistema

Los contratos del sistema son una forma de especificación de las operaciones, que indican una serie de requisitos que deben cumplir, además de cierta información útil en fases posteriores del desarrollo de un sistema. Dichos contratos son extraidos del *diagrama de casos de uso* (ver fig. 3.1) y formalizan el funcionamiento de los mismos.

Según el diagrama de casos de uso, los contratos a especificar son:

- Iniciar Sistema
- Cargar tópico
- Descargar Tópico
- Consultar Dominio
- Consultar Instancia
- Cambiar QoS (y sus variantes Lifespan, Durability,Transport Priority)
- Notificar Evento

Veamos por tanto la descripción de los contratos.

Contrato Iniciar Sistema

Nombre del caso de uso	Iniciar Sistema
Objetivo del caso de uso	Iniciar todas las estructuras internas mantenidas en el sistema
Actores que participan	Usuario, DDS Middleware
Precondiones	-
Postcondiciones	Las estructuras pertinentes del sistema quedan inicializadas
Flujo del caso de uso	1. Se inicia el analizador sintáctico, registrando las palabras reservadas de la gramática soportada 2. Se inicia la base de datos interna de dominios/topicos/instancias, que se usará para las futuras consultas 3. Se registran los mecanismos de comunicación entre componentes
Excepciones	-

Contrato Cargar Tópico

Nombre del caso de uso	Cargar Tópico
Objetivo del caso de uso	Cargar un caso de uso de modo que sea reconocido por el sistema en las sucesivas consultas a instancias de dicho tópico.
Actores que participan	Usuario, DDS Middleware
Precondiones	-
Postcondiciones	El caso de uso queda registrado en el sistema a efectos de consultas futuras.
Flujo del caso de uso	1. - El actor usuario solicita el registro de un tópico en el sistema 2. - El actor DDS Middleware crea un participante en el dominio del tópico con las políticas QoS por defecto 3. - Se asocia el tópico a un determinado buddy de mensajería tal que si no existe se creará 4. - Se añade un token que se asociará al tópico registrado 5. - Si todo ha ido bien, se cambia el estado del buddy asignado a online y acaba el caso de uso. En caso contrario, se informará del error al usuario y finalizá.
Excepciones	▪ No se pueden crear *participantes del dominio* (DomainParticipant) ▪ El topico ya se encuentra registrado

Contrato Descargar Tópico

Nombre del caso de uso:	Descargar Tópico
Objetivo del caso de uso:	Eliminar toda referencia al tópico que vamos a desregistrar, de modo que ya no sea reconocido por el sistema
Actores que participan:	Usuario, DDS Middleware
Precondiones:	El tópico se encuentra registrado en el sistema.
Postcondiciones:	El tópico ya no se encontrará registrado al finalizar el caso de uso.
Casos de uso relacionados:	Registrar Tópico
Flujo del caso de uso:	1. El actor *usuario* solicita el desregistro del tópico, ya que no está interesado en él. 2. Se informa al actor *DDS Middleware* sobre el deseo de desregistrar el tópico. Éste da de baja al *DomainParticipant* que había creado previamente. 3. Se desregistran los topicos, se eliminan las instancias y las asociaciones tópico/buddy. 4. Se elimina el token correspondiente al tópico que hemos desregistrado en la gramática
Excepciones	▪ El tópico no se encuentra registrado

Contrato Consultar QoS

Nombre del caso de uso:	Consultar QoS
Objetivo del caso de uso:	Consultar las políticas de calidad de servicio de un determinado tópico
Actores que participan:	Usuario, DDS Middleware
Precondiones:	El tópico debe de ser válido y reconocido por el buddy.
Postcondiciones:	Se genera un mensaje informando de la política QoS actual
Flujo del caso de uso:	1. El actor usuario solicita la política QoS de un determinado tópico a un Buddy. *a*) Si el buddy reconoce al topic, continuar *b*) En caso contrario, finalizar el caso de uso, e informar al *usuario* del error ocurrido 2. El buddy solicita al actor *DDS Middleware* las políticas del tópico y se las devuelve al Buddy 3. El buddy se las presenta al usuario en un formato legible por él.
Excepciones:	▪ La petición contiene errores sintácticos ▪ El buddy consultado y el tópico que queremos consultar no se encuetran asociados ▪ El tópico no se encuentra registrado en el sistema

Contrato Consultar Instancia

Nombre del caso de uso:	Consultar Instancia
Objetivo del caso de uso:	Devolver el valor de una instancia de tópico al usuario de una forma legible.
Actores que participan:	Usuario, Buddy
Precondiones:	El tópico debe de existir, y debe de ser reconocido por el buddy. La clave de la instancia debe de existir y pertenecer a una instancia
Postcondiciones:	-
Flujo del caso de uso:	1. El actor *usuario* solicita conocer el valor de una instancia de tópico, para ello manifiesta su intención a un buddy. 2. La solicitud del *buddy* es procesada y se consulta en un registro interno. *a*) Si el *buddy* reconoce el tópico, y además la instancia identificada por la clave existe, se procesa para devolvérsela de forma legible al usuario y se continuará con el caso de uso *b*) Si alguna de las condiciones anteriores no se cumple, se informará al *usuario* del error y se terminará el caso de uso. *c*) El valor de la instancia es presentado al *usuario*.
Excepciones:	▪ El tópico no se encuentra registrado en el sistema ▪ La petición contiene errores sintácticos ▪ No existe la instancia del tópico o no tenemos información acerca de ella ▪ Tópico y Buddy no se encuentran asociados

Contrato Consultar Dominio

Nombre del caso de uso:	Consultar Dominio
Objetivo del caso de uso:	El usuario obtiene información acerca de los dominios de DDS en los que participa.
Actores que participan:	Usuario, Buddy
Precondiones:	El dominio debe de existir
Postcondiciones:	-
Flujo del caso de uso:	1. El actor *usuario* solicita información acerca de un dominio determinado a un buddy 2. El *buddy* procesa la información y muestra la información disponible en formato legible *a*) Si no se dispone de información acerca del dominio, el buddy no posee participantes en ese dominio, o no existe. Mostraremos un error al usuario y acabaremos el caso de uso. *b*) Si todo ha ido bien, el buddy muestra la información y finalizar el caso de uso
Excepciones:	1. Errores sintácticos en la petición del usuario.

Contrato Actualizar Instancia

Nombre del caso de uso:	Actualizar Instancia
Objetivo del caso de uso:	Actualizar el valor de una instancia de acuerdo a los valores de DDS.
Actores:	DDS Middleware
Precondiciones:	El tópico debe encontrarse registrado
Postcondiciones:	La instancia de tópico debe quedar almacenada para consulta con los valores adecuados.
Escenario principal	1. El actor DDS Middleware detecta una actualización de una instancia de tópico remota 2. El actor DDS Middleware solicita la actualización del valor de la instancia al sistema 3. El Sistema busca el tópico que ha sido actualizado 4. El Sistema busca la instacia del tópico actualizado 5. El Sistema actualiza los valores de la instancia de tópico
Excepciones	▪ El tópico no existe ▪ Si la instancia no existía se debe de crear una nueva estructura interna para ella

Contrato Notificar evento

Nombre del caso de uso:	Notificar evento
Objetivo del caso de uso:	Notificar al usuario final por medio de un mensaje de un *buddy* de los eventos relevantes que ocurran en DDS, como por ejemplo fallos en QoS
Actores:	DDS Middleware, Topic-Buddy,Usuario
Precondiciones:	-
Postcondiciones:	El usuario debe recibir la información en un formato legible
Escenario principal	1. Ocurre un evento en DDS que es detectado por el DDS Middleware 2. Se contruye un mensaje que informe del evento 3. El mensaje se envía a un topic buddy asociado al tópico donde ocurre el mensaje 4. El topic buddy muestra el mensaje del evento al usuario
Excepciones	▪ Si no se indica el topic buddy, se usará un topic buddy por defecto

Contrato Cambiar QoS

Nombre del caso de uso:	Cambiar QoS
Objetivo del caso de uso:	Cambiar las políticas de calidad de servicio de un determinado tópico.
Actores que participan:	Usuario, Buddy, DDS Middleware
Precondiones:	El tópico debe de encontrarse y disponer de un participante en un dominio. Además la política debe de ser modificable dinámicamente.
Postcondiciones:	Las politicas QoS del tópico quedan actualizadas.

Flujo del caso de uso:	1. El actor *usuario* solicita el cambio de una política QoS de un tópico a un *buddy*.
	   a) Si el *buddy* reconoce el tópico, solicita al *DDS Middleware* el cambio de política QoS.
	   b) En caso contrario, el *buddy* muestra al *usuario* un error y acaba el caso de uso.
	2. El *DDS MIddleware*, cambia los parámetros de las politicas solicitadas. El resultado de la operación (ya sea exitosa o fallida) se muestra al usuario y el caso de uso acaba.

Excepciones:	▪ El tópico no se encuentra registrado ▪ El buddy y el tópico no se encuentran asociados ▪ Error/es sintáctico/s en la petición ▪ Valor incorrecto de QoS ▪ Error en la obtención/asignación de la política QoS del tópico
Observaciones:	Este es un caso de uso abstracto, que se materializará en casos de uso más concretos

Contrato Cambiar QoS Lifespan

Nombre del caso de uso:	Cambiar QoS Lifespan
Objetivo del caso de uso:	Cambiar las políticas de calidad de servicio de un determinado tópico.
Actores que participan:	Usuario, Buddy, DDS Middleware
Precondiones:	El tópico debe encontrarse registrado y disponer de un participante en un dominio.
Postcondiciones:	La politica QoS Lifespan del tópico queda actualizada.
Flujo del caso de uso:	1. El actor *usuario* solicita el cambio de una política QoS Lifespan de un tópico a un *buddy*. *a*) Si el *buddy* reconoce el tópico solicita al *DDS Middleware* el cambio de política QoS. *b*) En caso contrario, el *buddy* muestra al *usuario* un error y acaba el caso de uso. 2. El *DDS MIddleware* obtiene la política actual y cambia los parámetros de la politica lifespan solicitada. El resultado de la operación (ya sea exitosa o fallida) se muestra al usuario y el caso de uso acaba.
Excepciones:	▪ El tópico no se encuentra registrado ▪ El buddy y el tópico no se encuentran asociados ▪ Error/es sintáctico/s en la petición ▪ Valor incorrecto de QoS Lifespan (fuera de rango o no entero) ▪ Error en la obtención/asignación de la política QoS del tópico

Contrato Cambiar QoS Deadline

Nombre del caso de uso:	Cambiar QoS Deadline
Objetivo del caso de uso:	Cambiar la política de calidad de servicio de un determinado tópico.
Actores que participan:	Usuario, Buddy, DDS Middleware
Precondiones:	El tópico debe encontrarse registrado y disponer de un participante en un dominio.
Postcondiciones:	La politica QoS Deadline del tópico queda actualizada.
Flujo del caso de uso:	1. El actor *usuario* solicita el cambio de una política QoS Deadline de un tópico a un *buddy*. *a*) Si el *buddy* reconoce el tópico solicita al *DDS Middleware* el cambio de política QoS. *b*) En caso contrario, el *buddy* muestra al *usuario* un error y acaba el caso de uso. 2. El *DDS MIddleware* obtiene la política actual y cambia los parámetros de la politica durability solicitada. El resultado de la operación (ya sea exitosa o fallida) se muestra al usuario y el caso de uso acaba.
Excepciones:	▪ El tópico no se encuentra registrado ▪ El buddy y el tópico no se encuentran asociados ▪ Error/es sintáctico/s en la petición ▪ Valor incorrecto de QoS Deadline (fuera de rango o no entero) ▪ Error en la obtención/asignación de la política QoS del tópico

Contrato Cambiar QoS Transport Priority

Nombre del caso de uso:	Cambiar QoS Transport Priority
Objetivo del caso de uso:	Cambiar la política de calidad de servicio de un determinado tópico.
Actores que participan:	Usuario, Buddy, DDS Middleware
Precondiones:	El tópico se debe encontrar registrado y además disponer de un participante en un dominio.
Postcondiciones:	La politica QoS Transport Priority del tópico queda actualizada.

Flujo del caso de uso:	1. El actor *usuario* solicita el cambio de una política QoS Transport Priority de un tópico a un *buddy*.
	   a) Si el *buddy* reconoce el tópico solicita al *DDS Middleware* el cambio de política QoS.
	   b) En caso contrario, el *buddy* muestra al *usuario* un error y acaba el caso de uso.
	2. El *DDS MIddleware* obtiene la política actual y cambia los parámetros de la politica transport priority solicitada. El resultado de la operación (ya sea exitosa o fallida) se muestra al usuario y el caso de uso acaba.

Excepciones:	▪ El tópico no se encuentra registrado ▪ El buddy y el tópico no se encuentran asociados ▪ Error/es sintáctico/s en la petición ▪ Valor incorrecto de QoS Transport Priority (fuera de rango o no entero) ▪ Error en la obtención/asignación de la política QoS del tópico

4.3. Operaciones del sistema

Para explicar el funcionamiento y operaciones del sistema, se opta por una descr ipción formal basada en diagramas de secuencia. En esta sección se proponen y explican las operaciones más relevantes

4.3.1. Registrar tópico

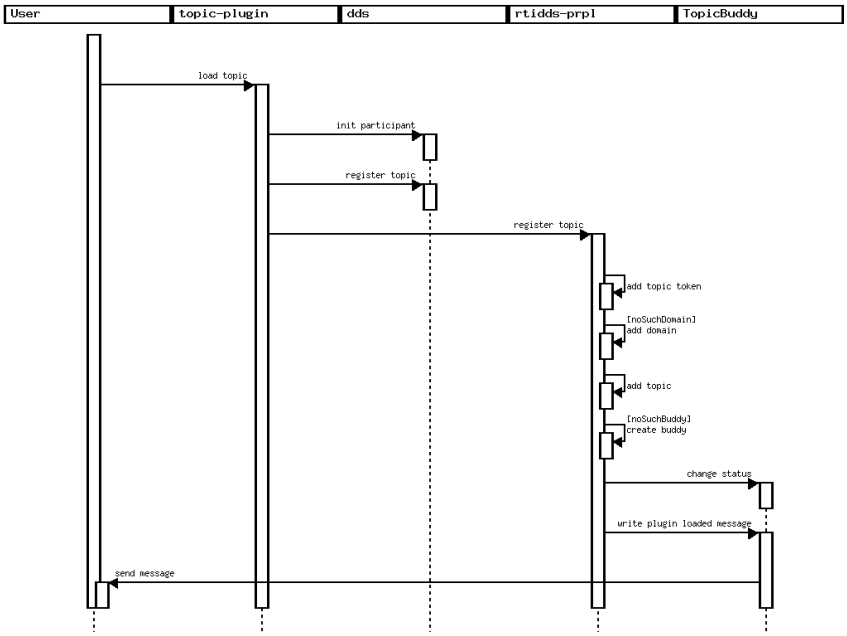


Figura 4.2: Diagrama de secuencia de *Registra Tópico*

Dentro de la operación *Registrar Tópico* se identifican las siguientes operaciones mostradas en la figura 4.2.

load topic El usuario requiere la carga de un nuevo *topic-plugin* para dar soporte a un nuevo tipo de tópico.

init participant Crea un *DomainParticipant* y un *DataReader* en DDS, para darnos de alta en el mismo.

register topic Asocia un tópico al *DataReader* creado.

register topic Indica al *rtidds-prpl* que registre un nuevo tipo de tópico.

add topic token Se añade un nuevo token al analizador sintáctico para que sea reconocido como un nombre de tópico.

add domain Crea un nuevo dominio en nuestra base de datos interna, concretamente el dominio que contiene a al tópico.

add topic Almacena la información concreta del buddy en nuestra base de datos interna, y la almacenamos. Se almacenan datos acerca de estructura del topic, entre otros.

create buddy Crea el buddy asociado a el nuevo tópico en caso de que no exista.

change status Cambia el estado del buddy asociado al tópico. Cambia el estado del buddy a en línea.

write plugin loaded message Genera un mensaje para informar al usuario que el tópico ha sido correctamente instalado

send message Manda al usuario el mensaje de información del plugin cargado

4.3.2. Desregistrar tópico

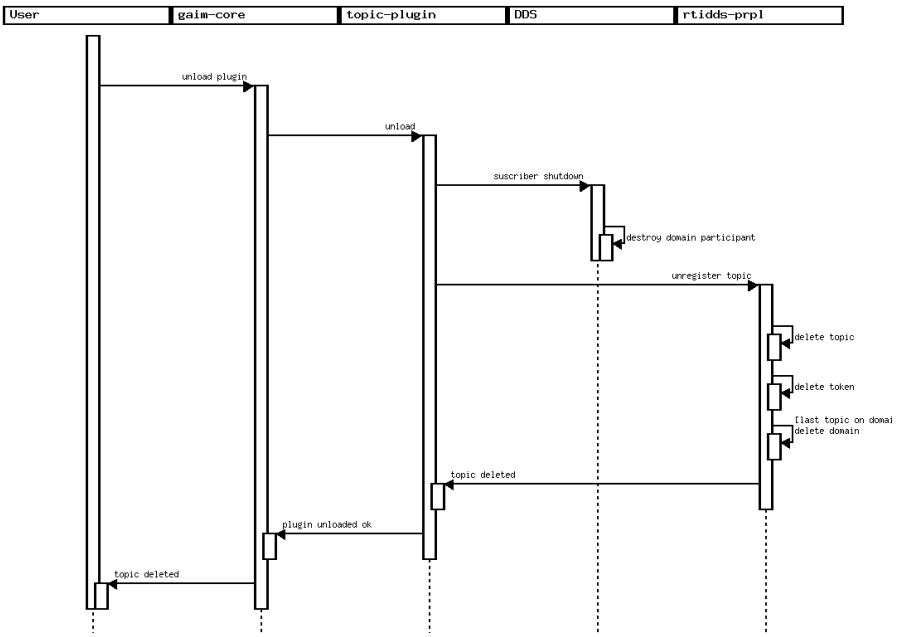


Figura 4.3: Diagrama de secuencia de *Desregistra Tópico*

Estas son las operaciones identificadas. Ver figura 4.3 para más detalle.

Unload topic

unload

suscriber shutdown

destroy domain participant

unregister topic

delete topic

delete token

delete domain

topic deleted

plugin unloaded

4.3.3. Cambia QoS

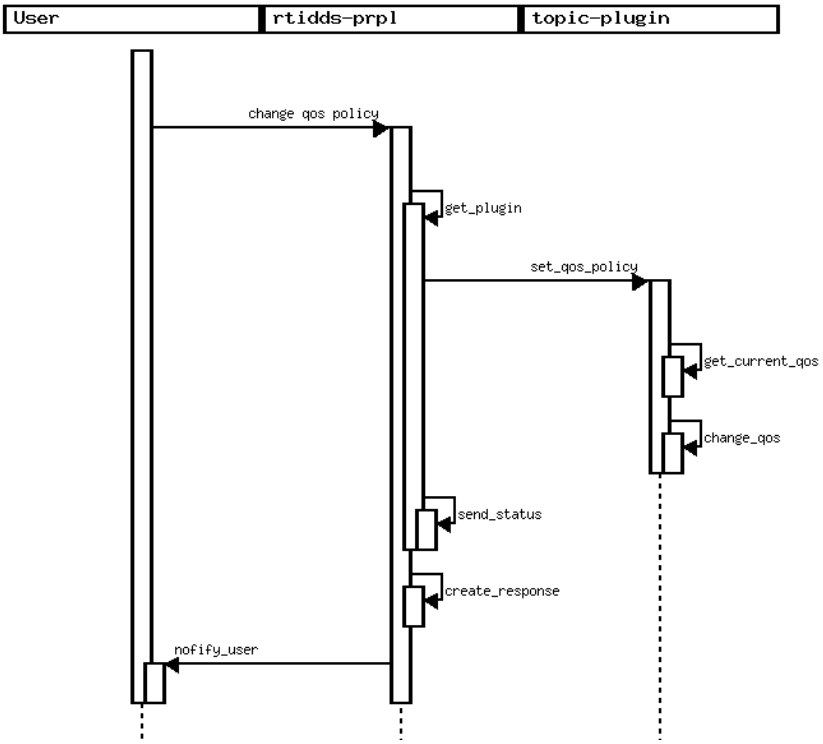


Figura 4.4: Diagrama de secuencia de *Cambia QoS*

La operación *Cambia QoS* representada en la figura 4.4 es una operación abstracta que se materializa en casos más concretos en función de la política que se desee modificar. Más concretamente se especializa para las siguientes políticas QoS:

- Deadline

- Lifespan
- Transport Priority

Con esto se presente resaltar, que esta operación no se encuentra como tal en la aplicación, sino que más bien es una *plantilla* para varias operaciones muy similares a ella. En la operación *Cambia QoS* se han identificado las siguientes operaciones:

cambia qos policy El usuario solicita cambiar algún parámetro de la políticas QoS

get plugin Obtenemos el plugin encargado de gestionar el tópico al que queremos cambiar la política. Para ello realizamos una consulta en nuestra base de datos interna.

set qos Realizamos una llamada IPC al plugin encargado de manejear dicho tópico, y solicitamos el cambio de una política

get qos Obtenemos la política QoS actual del tópico

change qos Cambiamos el *deadline* del tópico.

create response Generamos una respuesta para el usuario

notify user Notificamos al usuario del resultado de la operación

Estas operaciones son comunes para las operaciones *Cambia QoS Deadline, Cambia QoS Lifespan y Cambia QoS Transport Priority.*

4.3.4. Cambia QoS Deadline

En la figura 4.5 podemos ver el diagrama de secuencia de la operación *change QoS Deadline*, encargada de cambiar la política *Deadline* de las calidades de servicio de un tópico determinado. Este es un caso concreto de *Cambia QoS*, en el que se cambia la política *Deadline*. Las operaciones específicas en este caso son:

cambia qos deadline El usuario solicita cambiar la política de QoS *deadline* a cierto valor entero

set qos deadline Se realiza una llamada IPC al plugin encargado de manejear dicho tópico, y solicitamos el cambio de *deadline* a cierto valor entero que refleja un umbral del ratio de recepción de muestras de un determinado tópico.

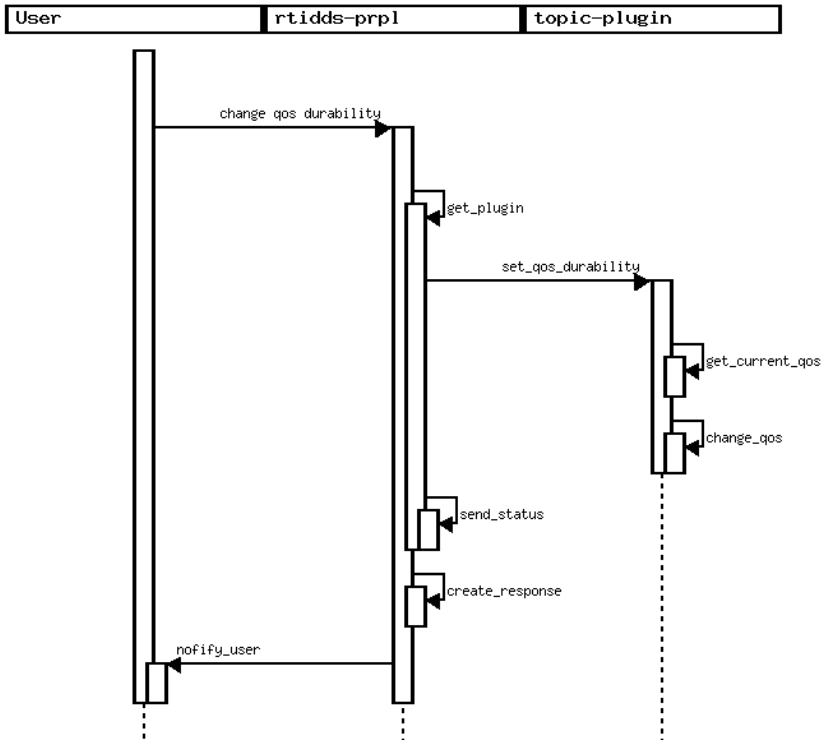


Figura 4.5: Diagrama de secuencia para *change qos deadline*

4.3.5. Cambia QoS Lifespan

En la figura 4.6 podemos ver el diagrama de secuencia de la operación *change QoS deadline*, encargada de cambiar la política *Lifespan* de las calidades de servicio de un tópico determinado. Como el caso de *Cambia QoS Deadline*, este es un caso concreto de *Cambia QoS*, y se identifican las siguientes operaciones nuevas:

cambia qos lifespan El usuario solicita cambiar la política de QoS *lifespan* a cierto valor entero.

set qos lifespan Se realiza una llamada IPC al plugin encargado de manejar dicho tópico, y solicitamos el cambio de *lifespan* a un valor entero que refleja la validez de los mensajes en el sistema.

4.3.6. Cambia QoS Transport Priority

En la figura 4.7 podemos ver el diagrama de secuencia de la operación *change QoS deadline*, encargada de cambiar la política *Transport Priority* de las calidades de servicio de un tópico determinado. Además de las operaciones descritas anteriormente, se han identificado las siguientes:

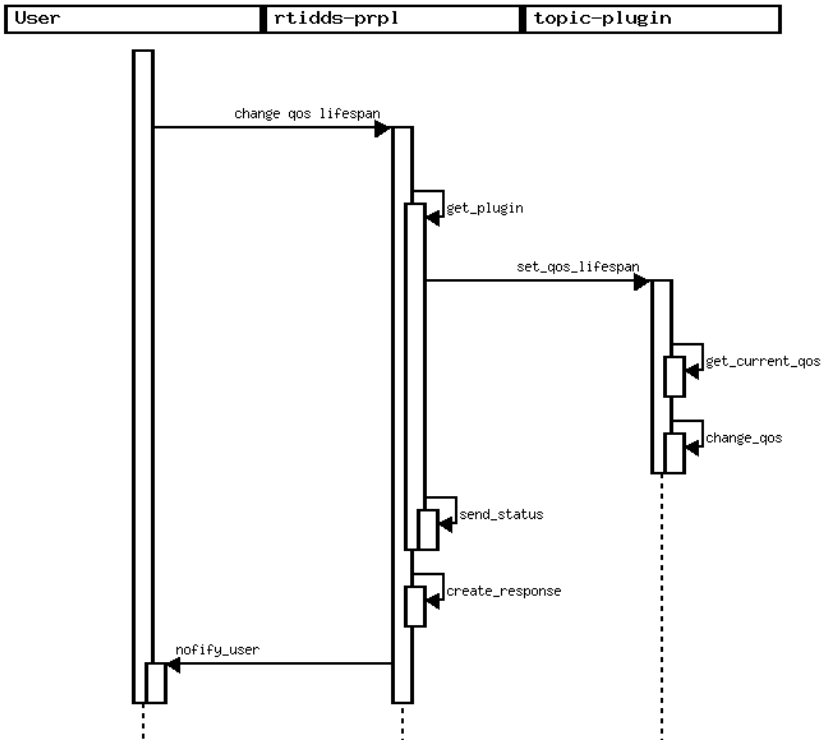


Figura 4.6: Diagrama de secuencia para *change qos lifespan*

cambia qos transport El usuario solicita cambiar la política de QoS *transport priority* a cierto valor entero.

set qos transport Se realiza una llamada IPC al plugin encargado de manejear dicho tópico, y solicitamos el cambio de *transport priority*. El valor de dicha política es un valor entero que se refiere al valor de prioridad de los tópicos en la capa de transporte sobre la que se apoya DDS. Solo tiene validez en capas de transporte que soporten mensajes con prioridades.

4.3.7. Consulta Instancia

Estas son las operaciones identificadas para el caso de consulta instancia. Ver figura 4.8 para más detalle.

ask about domains El usuario pregunta acerca de un dominio/s con un mensaje a un buddy.

parse message El mensaje es analizado sintácticamente para comprobar la correctitud además de aquello por lo que pregunta.

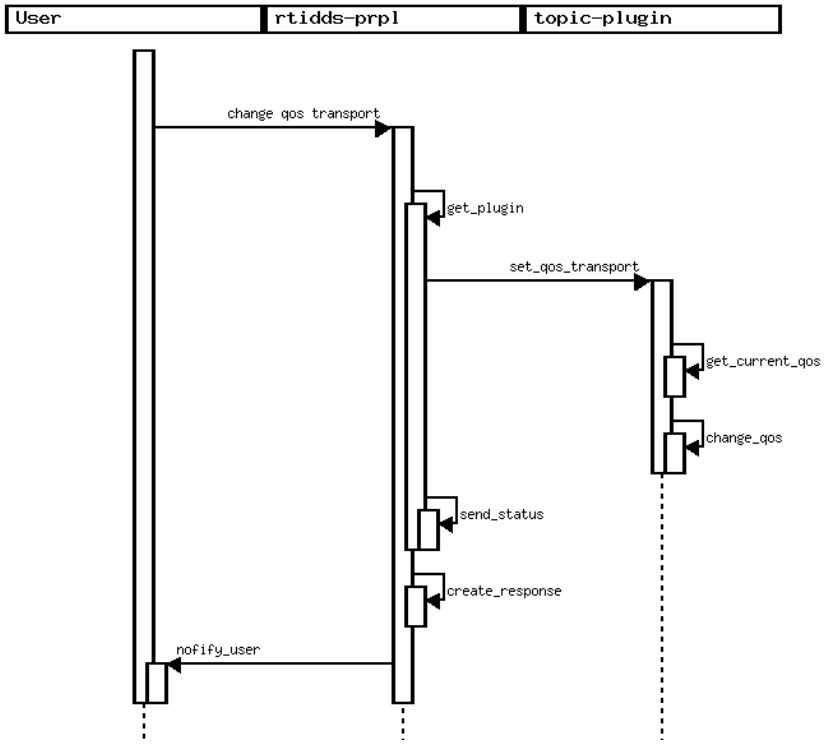


Figura 4.7: Diagrama de secuencia de *Cambia QoS Transport Priority*

query instance by key Hacemos una consulta en nuestra base de datos interna acerca de una instancia, que se encuentra identificada con una determinada clave respecto de los demás.

generate response Generamos una respuesta con el contenido de nuestra base de datos.

write response La respuesta se escribe en una conversación con el buddy.

send response Se manda la respuesta finalmente para que el usuario pueda leerla.

4.3.8. Consulta QoS

Estas son las operaciones identificadas para el caso de *Consulta QoS*. Ver figura 4.9 para más detalle.

query topic qos El usuario escribe un mensaje al buddy solicitándole información de un determinado tópico.

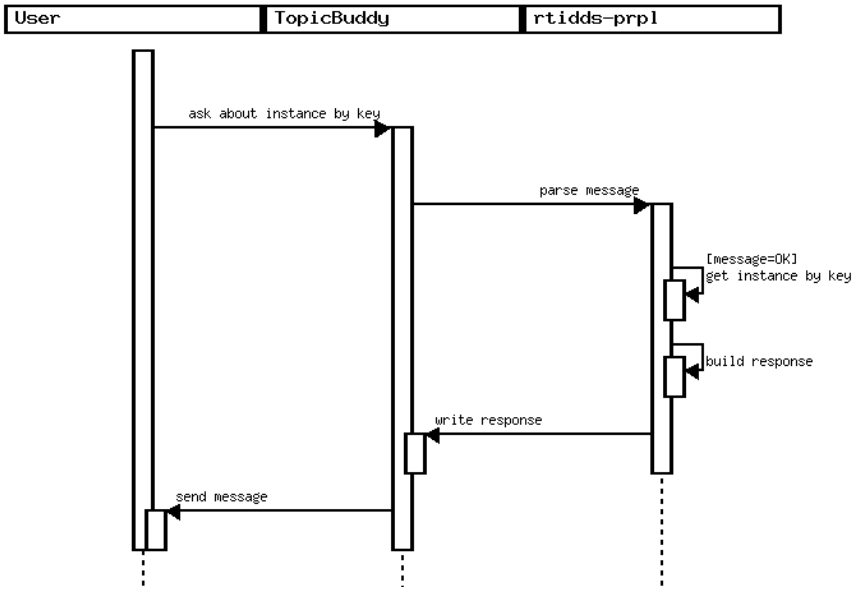


Figura 4.8: Diagrama de secuencia de *Consulta Instancia*

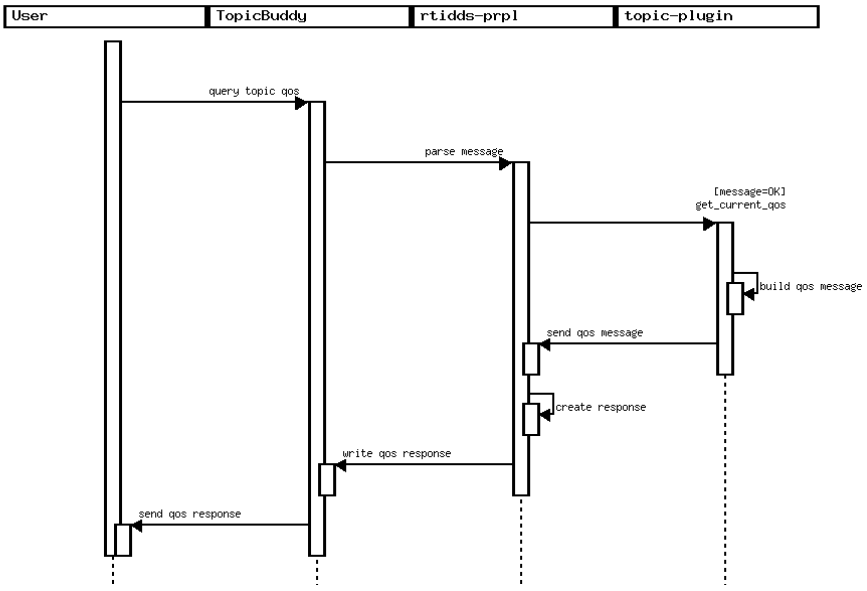


Figura 4.9: Diagrama de secuencia de *Consulta QoS*

parse message El mensaje es analizado sintácticamente para comprobar la correctitud de aquello por lo que pregunta así como para extraer la información necesaria para la consulta, como el nombre del tópico

get current qos Obtener las políticas QoS de un determinado tópico

build qos message Construir un mensaje XML que incluya toda la información acerca de las políticas de calidad de servicio de un tópico.

send qos message Mandar el mensaje XML contruido al *rtidds-prpl*

generate response Construir una respuesta de texto a partir del mensaje XML.

write response La respuesta se escribe en una conversación con el buddy.

send response Se manda la respuesta finalmente para que el usuario pueda leerla.

4.3.9. Consulta Dominio

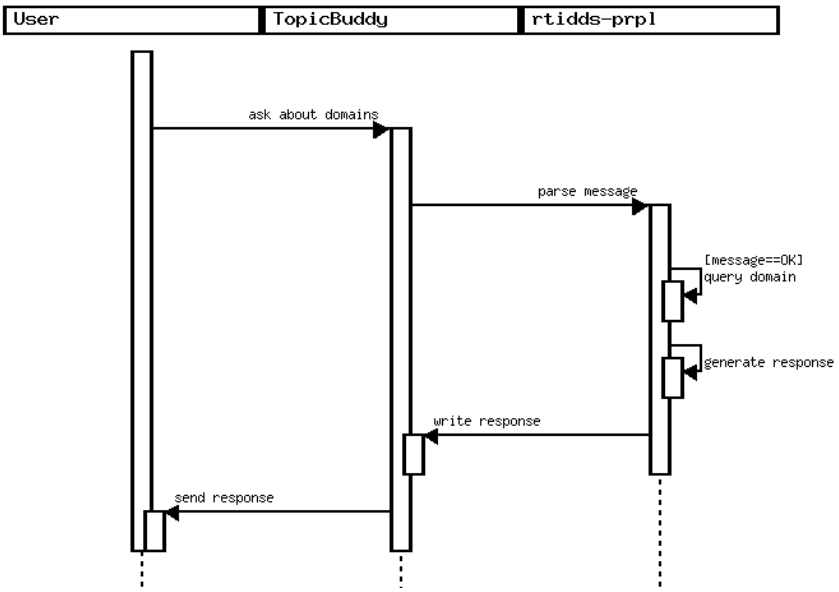


Figura 4.10: Diagrama de secuencia de *Consulta Dominio*

Estas son las operaciones identificadas para *Consulta Dominio*. Ver figura 4.10 para más detalle.

ask about domains El usuario pregunta acerca de un dominio/s con un mensaje a un buddy.

parse message El mensaje es analizado sintácticamente para comprobar la correctitud además de aquello por lo que pregunta.

query domain Hacemos una consulta en nuestra base de datos interna acerca de dominios.

generate response Generamos una respuesta con el contenido de nuestra base de datos.

write response La respuesta se escribe en una conversación con el buddy.

send response Se manda la respuesta finalmente para que el usuario pueda leerla.

4.3.10. Actualiza Instancia

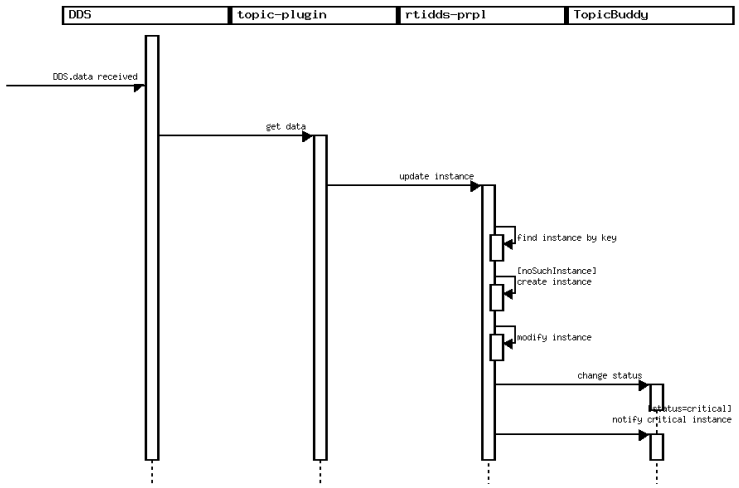


Figura 4.11: Diagrama de secuencia de *Actualiza Instancia*

Estas son las operaciones identificadas para la operación *Actualizar Instancia*. Ver figura 4.11 para más detalle.

data received Operación asíncrona desencadenada cuando llegan nuevos datos al DDS.

get data Mediante esta operación, se extraen los datos de DataReader de DDS y los almacenamos en una variable

update instance Se notifica al *rtidds-prpl* sobre la llegada de nuevos datos.

find instance by key Operación que se encarga de localizar una determinada instancia almacenada localmente para modificarla

create instance Si la instancia que ha llegado, no existía en la base de datos local, la creamos con los datos que han llegado

modify instance Modifica los datos de la instancia almacenada localmente

change status Modifica el estado del buddy asociado al topic que ha llegado

send message En el caso que la instancia recibida suponga un estado de alarma, se notifica al usuario con un mensaje asíncrono.

4.3.11. Notificar Evento

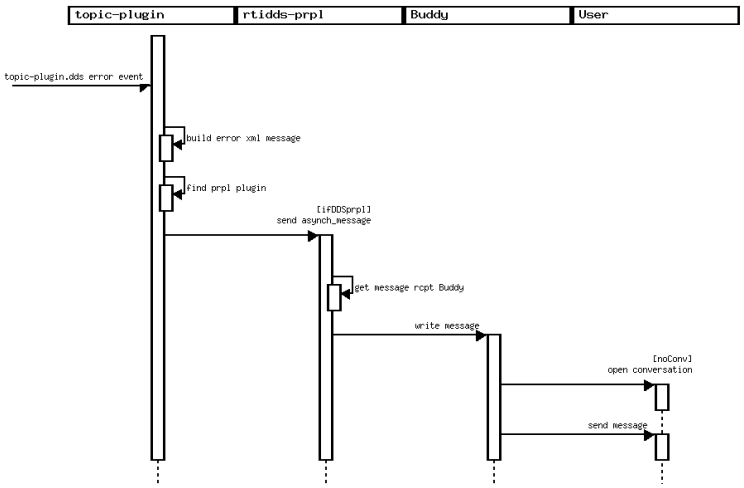


Figura 4.12: Diagrama de secuencia de *Notificar Evento*

Estas son las operaciones identificadas para el caso de *Notificar Evento*. Ver figura 4.12 para más detalles acerca de la secuencia de uso y las entidades involucradas.

DDS error Event Es el nombre genérico para una operación que se desencadena cuando ocurre algún evento en DDS. Normalmente son *callbacks*, llamados cuando ocurre algún error en las políticas QoS. Cuando uno de estos eventos ocurre se desencadena el proceso que hemos denominado *asynch message*

build error XML message Es una operación que construye un mensaje de error en función de los datos de los que dispone. El mensaje creado es un mensaje XML que se formateará para generar una respuesta *human-readable*

find prpl plugin Esta operación es la encargada de encontrar el plugin de protocolo de DDS, para comunicarse con el. Para ello se busca entre todos los plugin de protcoolos según el ID del protocolo

Send Asynch message Esta operación es una operación remota que define el *rtidds-prl*, que se encarga de mandar mensajes de manera asíncrona sin que el usuario haya generado una petición previa relacionada. Se usa para generar mensajes de error y notificarselos al usuario

get message rcpt buddy Los mensajes asícronos pueden ser dirigidos al buddy *DDS* (que sirve para gestionar el sistema), o para un buddy determinado. Mediante esta operación obtenemos el destinatario del mensaje

write message Mediante esta operación solicitamos escribir un mensaje a un determinado buddy con un formato ya legible para humanos

open conversation Solicita la creación de una conversación en el caso que no exista una previamente

send message Escribimos el mensaje en la conversación previamente abierta, de modo que el usuario pueda acceder a ella

4.4. Diagrama de despliegue

Uno de los factores que son importantes en un buen diseño es el diseño de una arquitectura fácilmente implantable. Este proyecto hace hincapíe en este aspecto, ya que uno de sus objetivos principales es el de facilitar al usuario el uso de entornos distribuidos.

En la figura **??**, se muestra el diseño de la arquitectura que propuesta en el proyecto. Se ha adoptado la nomenclatura UML para que sea fácilmente entendible.

Nomenclatura empleada Se observa en diagrama de la figura **??** que estamos ante una arquitectura distribuida. Cada caja tridimiensional, representa un nodo de procesamiento (ésto no significa que dos nodos no puedan estar en la misma máquina física). Dentro de cada nodo se observan una serie de cajitas con un icono. Estas cajitas representan piezas software que se comunican entre sí. Por otro lado aparecen líneas de asociación etiquetadas. La etiqueta especifica el mecanismo de interacción entre ellas.

Descripción de la arquitectura Como se puede ver, existen dos tipos de entidades diferenciados en la arquitectura propuesta:

Nodo de publicación Este nodo se refiere a una entidad DDS *típica*. Esta entidad se limita a publicar tópicos que serán leídos por un nodo cliente y presentados al usuario mediante la interfaz de mensajería instantánea. Estos nodos no requieren la ejecución de código especial,

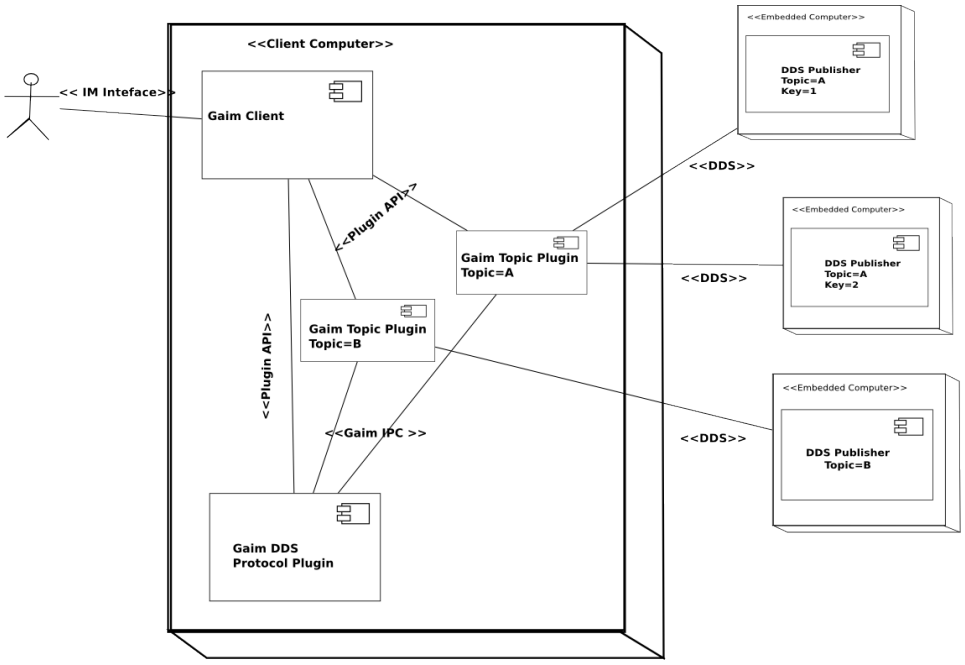


Figura 4.13: Diagrama de despliegue

por lo que no es necesario describirlos, ya que la aplicación podrá ejecutarse en entornos DDS ya establecidos sin modificación alguna de los nodos de publicación.

Nodo cliente Es el propósito de el proyecto. Constituye el punto de interconexión entre un Usuario (físico) y el sistema DDS. El proyecto consiste por tanto en el diseño e implementación de un nodo cliente por tanto.

4.4.1. Arquitectura de un nodo de publicación

Un nodo de publicación únicamente necesitará las funciones para acceder al DDS. Para ello hace uso de la API de acceso al DDS. Necesitará por tanto disponer de un participante en el dominio (DomainParticipant), y un publicador (DataWriter) que serán los encargados de publicar instancias de un determinado tópico. Con esto en mente, se observa que el proyecto no deberá hacer modificaciones en los nodos de publicación con lo que el impacto de implantación del prouetco en un escenario típico ya existente es casi nulo.

4.4.2. Arquitectura de un nodo cliente de mensajería instantánea

Como se mencionó en secciones anteriores, el proyecto desarrollado aprovecha un programa de mensajería instantánea para ser extendido y añadirle la funcionalidad requerida. Es por esto que se debe aprovechar la arquitectura que proporcione el programa para adaptarlo a las necesidades. Gaim (la aplicación a extender) proporciona la posibilidad de extensión del programa mediante el empleo de *plugins*, por lo que la arquitectura del programa aprovechará dicha ventaja.

El nodo cliente se organiza por tanto en dos niveles (ver figura 4.1): un primer nivel que se encarga de la interacción con el usuario (rtidds-prpl) y una capa que se encarga de interactuar con el servicio de distribución de datos (topic-plugins). Hay que recordar, que según este diseño, debe de existir un *topic-plugin* por cada tipo de tópico soportado. Estos dos niveles se comunican entre sí mediante una API bien definida.

El *rtidds-prpl* plugin

En el sistema existe un único *rtidds-prpl* que será el encargado de realizar una serie de funciones en el sistema, que son:

- Interactuar con el usuario, reconociendo las peticiones del usuario y efectuando las acciones que correspondan a dicha acción.
- Encaminar las peticiones acerca de un tópico al plugin encargado de gestionar dicho tópico. p.e. los cambios y consultas de política *QoS*, que son específicas para cada tópico.
- Presentar los resultados e información al usuario en un formato legible para un ser humano, a ser posible lo más cercano al lenguaje natural.
- Gestionar una base de datos interna que represente el estado actual de DDS.

Para llevar a cabo estas funcionalidades, se ha realizado un estudio de como interaccionarían los *rtidds-prpl* y *topic-plugin*, y se ha considerado que un *rtidds-prpl* debe proporcionar la siguiente interfaz:

- Registrar nuevo tópico
- Actualizar instancia
- Desregistrar tópico
- Mensaje asíncrono

Se observa de acuerdo con la arquitectura propuesta, que este tipo de plugins no necesita de un acceso directo al DDS, por lo que se libera de la creación de parcicipantes de dominio (DomainParticipants) y subscriptores (DataReader). Las funciones de acceso al DDS, son relegadas por tanto a los *topic-plugins*, siendo el *rtidds-prpl* un mero intermediario entre los *topic-plugins* y la librería de mensajería instantánea. Como ventaja, nótese que esta opción de diseño mejora la reutilizabilidad del desarrollo.

Los *topic-plugins*

En contraposición a lo que ocurre con el *rtidds-prl*, típicamente existen múltiples *topic-plugin* cargados en el sistema. Más concretamente, existe uno por cada tópico a reconocer por la aplicación. Las funcionalidades que deben cumplir éstos son:

- Registrar/desregistrar un tópico en el sistema
- Monitorizar el estado de todas las instancias de dicho tópico en DDS y notificárselo al *rtidds-prl*, que es el encargado de gestionar una base de datos interna
- Gestionar las políticas QoS para dicho tópico

El cometido de un *topic-plugin* es el de acceder al DDS para obtener nuevos valores de instancias, gestionar las políticas de calidad y servicio y notificar a los *rtidds-prpl*, por lo tanto éstos deben de acceder directamente al DDS. Para acceder al DDS cada *topic-plugin* creará un *DomainParticipant* y un *DataReader*. Estos serán los encargados de recoger la información acerca de publicaciones de instancias de tópicos.

Además de utilizar los procedimientos remotos proporcionados por el *rtidds-prpl* para comunicarse con el, se proporcionan procedimientos IPC propios. En el caso de los *topic-plugins*, son mayormente dedicados a gestionar las políticas QoS de un tópico determinado. Los procedimientos en cuestión son:

- Cambiar QoS Lifespan
- Cambiar QoS Transport Priority
- Cambiar QoS Deadline
- Obtener QoS

Una vez identificadas todas las funcionalidades e interfaces que proporcionan cada uno de los tipos de plugin del sistema, enla figura 4.13 se observa un ejemplo de interacción entre *rtidds-prpl* y un *topic-plugin* determinado. En esta figura, se pueden ver a los dos tipos de plugin a lo largo

de tiempo y como intercambian mensajes entre sí. La interacción se inicia cuando se carga un *topic-plugin* y se realiza una petición de registro en el sistema. Los siguientes intercambios que se realizan son de todo tipo, como pueden ser actualizaciones (*update_topic_instance*), mensajes de notificación (*asynch_message*), o bien de gestión de QoS (get_qos). La interacción entre ambos se finaliza cuando el usuario solicita descargar un *topic-plugin* y se solicita un desregistro de un determinado tópico.

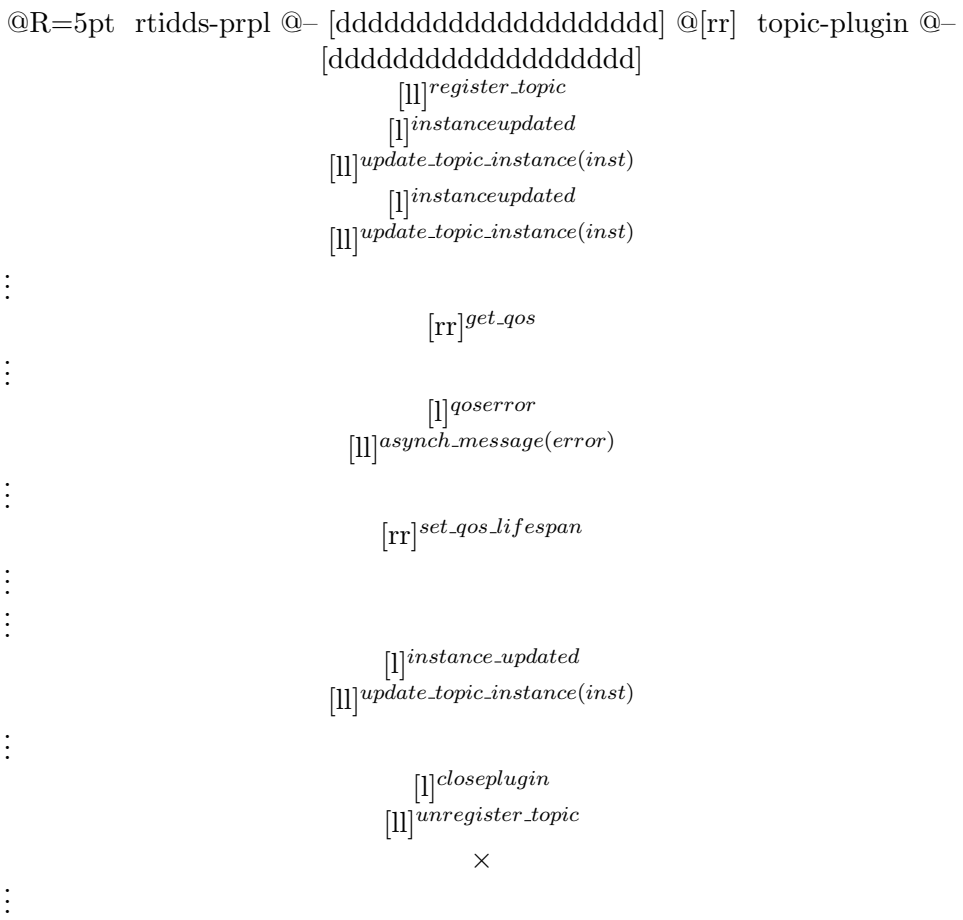


Figura 4.14: Secuencia de ejemplo de cómo interactúan un *rtidds-plugin* y un *topic-plugin*

4.5. Diseño de las operaciones IPC

Para concluir con el diseño, resta por abordar las operaciones IPC. A continuación se describen las operaciones identificando las tareas a realizar en cada una de ellas.

4.5.1. Registrar tópico

Descripción Esta operación es la encargada de registrar un tópico en el sistema para que sea reconocido por la aplicación de mensajería instantánea y así poder hacer consultas sobre el mismo. Será llamada cada vez que carguemos un *topic-plugin*, y se deberán cumplir todos los pasos correctamente para que el *topic-plugin* finalice su carga.

Tareas a realizar

1. Dar de alta un identificador de tópico para que sea reconocido por el parseador. Así cada vez que escribamos el nombre de dicho tópico, se identificará como tal.

2. Almacenar información acerca de un tópico para facilitar las consultas y conocer acerca de su estructura

3. Modificar el estado de el buddy asociado a un tópico a *online*

Parámetros de entrada

- Nombre del tópico: Nombre con el que se identifica al tópico y con el que será reconocido en el analizador sintáctico.
- Plugin propietario: Identificador del plugin que se encarga de manejar el tópico que estamos registrando
- Buddy: Nombre del buddy asocido al tópico, al que podremos realizar las consultas acerca del tópico.
- Descripción del tópico: Una breve descripción de la estructura interna del tópico (tópicos).
- Dominio: Dominio en el que se suscribe el *topic-plugin* para suscribirse a dicho tópico.

Valor devuelto

- Se devolverá un valor que indicará el éxito/fracaso del registro.

4.5.2. Desregistrar tópico

Descripción Esta función es la antagónica a *registrar tópico* (ver sección 4.5.1). Es llamada cuando se descarga un topic plugin, y se encarga de devolver la aplicación al estado anterior a la carga de dicho *topic-plugin* respecto a un tópico.

Tareas a realizar

1. Dar de baja el identificador de dicho tópico para que no sea reconocido por el parseador como un nombre de tópico
2. Eliminar toda la información de la base de datos interna que dispongamos de dicho tópico, sus instancias y dominios si procede.
3. Pasar el estado del buddy asociado a dicho tópico a *offline* si no tuviera más tópicos asociados.

Valores de entrada

- Nombre del tópico: Nombre del tópico que queremos desregistrar

Valor devuelto

- Se devolverá un valor que indicará el éxito/fracaso del registro.

4.5.3. Mensaje asíncrono

Descripción Esta operación es la encargada de mostrar mensajes de información al usuario para informarle de eventos. Estas informaciones se producen sin petición previa por parte del usuario, por eso se denominan asíncronos. Se definen dentro de *rtidds-prpl*, y es llamada por un *topic-plugin* cuando ocurre algún evento, como por ejemplo un fallo en las políticas QoS.

Tareas a realizar

- Tomar el mensaje y obtener el destinatario (buddy)
- Formatear el mensaje para mejorar la legibilidad
- Crear una conversación con el destinatario si no existía
- Enviar el mensaje al buddy destinatario para que le aparezca al usuario

Valores de entrada

- Mensaje: mensaje que informa del evento ocurrido.
- Destinatario: buddy que recibirá la notificación para mostrársela al usuario.

Valor devuelto Ninguno

4.5.4. Actualizar instancia

Descripción Esta función se encarga de actualizar los valores de una determinada instancia de tópico cuando se reciben nuevos valores en la base de datos XML interna de la aplicación. Esta función es invocada remotamente cada vez que un *topic-plugin* es notificado de la llegada de nuevas muestras de instancias de tópico en DDS.

Tareas a realizar

1. Analizar el mensaje XML que contiene la información de la instancia
2. Actualizar la instancia del tópico en la base de datos interna si existe. Si no existe previamente, entonces ha de crearse una referencia en la base de datos.
3. Si la instancia así lo indica, debe de cambiarse el estado del buddy.

Valores de entrada

- Instancia: mensaje XML con la información de la instancia
- Key: clave que identifica a la instancia
- status: estado al que debe cambiar el buddy asociado

Valor devuelto Ninguno

4.5.5. Cambiar QoS Transport Priority

Descripción Función que declara cada *topic-plugin* y que es encargada de cambiar el valor de prioridad de transporte de un tópico. Esta función remota es invocada cuando el usuario escribe un mensaje a un buddy solicitando el cambio de la prioridad de transporte de un tópico determinado. Posteriormente se llama a esta función dentro del *topic-plugin* correspondiente.

Tareas a realizar

- Comprobar el valor correcto de *Transport Priority*
- Cambiar la prioridad de los mensajes de la capa de transporte para un tópico determinado.
- Notificar al usuario del resultado de la operación

Valores de entrada

- prioridad: Valor entero que representa la nueva prioridad de transporte para el tópico correspondiente.

Valor devuelto Ninguno

Advertencias Esta operación solo tendrá sentido y validez en entornos cuyas capas de transporte soporten la prioridad.

4.5.6. Cambiar QoS Lifespan

Descripción Función que declara cada *topic-plugin* y que es encargada de cambiar el valor de la política QoS *lifespan* de un tópico. Esta función remota es invocada cuando el usuario escribe un mensaje a un buddy solicitando el cambio de la política QoS *lifespan* de un tópico determinado. Posteriormente se llama a esta función dentro del *topic-plugin* correspondiente.

Tareas a realizar

- Comprobar el valor correcto de *Lifespan*
- Cambiar el tiempo de vida (Lifespan) de los mensajes de DDS
- Notificar al usuario del resultado de la operación

Valores de entrada

- lifespan: Valor entero que representa la nueva *lifespan* en segundos de dicha política QoS para el tópico correspondiente

Valor devuelto Ninguno

4.5.7. Cambiar QoS Deadline

Descripción Función que declara cada *topic-plugin* y que es encargada de cambiar el valor de la política QoS *deadline* de un tópico. Esta función remota es invocada cuando el usuario escribe un mensaje a un buddy solicitando el cambio de la política QoS *deadline* de un tópico determinado. Posteriormente se llama a esta función dentro del *topic-plugin* correspondiente.

Tareas a realizar

- Comprobar el valor correcto de *Deadline*
- Cambiar el ratio de recepción mensajes válido (Deadline) del tópico correspondiente
- Notificar al usuario del resultado de la operación

Valores de entrada

- Durabilidad: Valor entero que representa la nueva durabilidad en segundos de dicha política QoS para el tópico correspondiente.

Valor devuelto Ninguno

4.5.8. Obtener QoS

Descripción Operación que es invocada cuando el usuario solicita información acerca de las políticas QoS de un tópico determinado. Es una función registrada por los *topic-plugins*

Tareas a realizar

1. Consultar la política QoS del tópico
2. Construir un mensaje XML para devolvérselo al *rtidds-prpl*
3. Devolverle dicho mensaje

Valores de entrada Ninguno

Valor devuelto

- QoS: Mensaje que contiene toda la información acerca de las políticas QoS. Este mensaje podrá ser formateado posteriormente para mayor legibilidad.

CAPÍTULO 5

Implementación

El proyecto desarrollado no sólo se llimita al diseño de la arquitectura, sino que también se implementa.

En este capítulo se describen algunos detalles de la implementación que pueden ser interesantes remarcar. En particular se hace hincapié en algunos aspectos de la misma, como son:

- Reconocimiento de gramática semi-natural.
- Mecanismos de comunicación entre *rtidds-prpl* y *topic-plugin*
- Generación de mensajes en lenguaje semi-natural para el usuario
- Generación de nuevos *topic-plugins* para reconocer nuevos tópicos

5.1. Reconocimiento de gramática semi-natural

Implementar una gramática es un proceso que puede resultar costoso si se desea que sea fácilmente extensible/modificable. Una mala implementación del analizador sintáctico que reconozca la gramática hace que el código se convierta en inmanejable e ininteligible.

Para evitar estos inconvenientes se ha optado por emplear herramientas automatizadas de generación de analizadores sintácticos . Mediante el empleo de una gramática BNF modificada con código fuente incrustado estas herramientas facilitan el proceso de creación de analizadores sintácticos además posteriores procesos de modificación/extensión de la gramática.Además la gramática definida en el capítulo 3, queda completamente con una gramática BNF. Es por esto que la gramática se puede implementar con de un analizador sintáctico de los llamados *LookAhead Left-Right* (LALR).

Muchas son las herramientas de este tipo, cada una con sus ventajas e inconvenientes. Para ser fieles a la filosofía de utilización de aplicaciones de código abierto elegida como requisito, se elije alguna de las disponibles en cualquier distribución GNU/Linux. En este caso se ha optado por *bison*, [19] la implementación GNU de un generador de parseadores muy popular: *YACC*. Tanto *bison* como *YACC* generan parseadores *LALR*.

El proceso de implementación, por tanto consiste en traducir la gramática definida en la sección 3.1 a *bison* y definir las acciones pertinentes para cada tipo de sentencia.

5.2. Comunicación entre *rtidds-prpl* y *topic-plugins*

En la sección 4.5 se mencionaron los mecanismos IPC que se utilizan para intercambiar información entre los plugins. En esta sección se adentrará un poco más en detalles de implementación, así como un mayor de detalle en la información intercambiada.

Para la comunicación entre plugins, se aprovecharán las herramientas de IPC que proporciona Gaim. Dicha comunicación se realiza mediante el intercambio de mensajes *XML*[20] entre plugins, que son procesados y desencadenan una serie de acciones en el plugin que las recibe.

Para poder utilizar las llamadas IPC de *Gaim*, primero hay que registrarlas en el sistema y definir su tipo y parámetros cuando el plugin es cargado. Una vez registradas pueden ser empleadas desde cualquier plugin que conozca su existencia. El proceso de llamada se realiza por medio de una función específica de *Gaim*. Para más información de estas funciones consultar la documentación de *Gaim*

El plugin *rtidds-prpl* hace públicos los siguientes métodos, para que sean invocados por los *topic-plugin*.

register_topic Se encarga de dar de baja a un tópico en el sistema. Se almacenan datos relacionados con ellos, como por ejemplo, la estructura del tópico, o el *topic-plugin* asociado. Es invocada cada vez que se carga un *topic-plugin*

unregister_topic Se encarga de dar de baja a un tópico en el sistema. El tópico no será reconocido más por el sistema. Es invocada cada vez que se descarga un *topic-plugin*

update_topic_instance Se encarga de actualizar el valor de una instancia almacenada determinada cada vez que se actualiza la instancia en el *DDS*. Cada vez que un dato es recogido por un *topic-plugin*, es invocada.

asynchronous_message Sirve para notificar al usuario de una incidencia

en el *DDS*, tales como la expiración de unos datos, o algún error ocurrido en un *topic-plugin*.

Por otro lado los *topic-plugin* hacen públicos los siguientes métodos para que sean invocados por el *rtidds-prpl*.

get_qos Devuelve un mensaje *XML* con los parámetros de QoS del tópico gestionado por el *topic-plugin*. Es invocado por el *rtidds-prpl* a petición del usuario por medio de la gramática

set_qos_deadline Cambia el parámetro de QoS *deadline* de un determinado tópico. Es invocado por el *rtidds-prpl* a petición del usuario.

set_qos_lifespan Cambia el parámetro de QoS *lifespan* de un determinado tópico. Es invocado por el *rtidds-prpl* a peticón del usuario.

set_qos_transport_priority Cambia el parámetro de QoS *transport priority* de un determinado tópico. Es invocado por el *rtidds-prpl* a petición del usuario.

A continuación se describen los parámetros que dispone cada una de ellas y los valores que devuelven.

5.2.1. register_topic

Como se definió en el capítulo 4, esta operación es llamada por cada *topic-plugin* cuando es cargado, para así registrar el tópico en el sistema correctamente.

```
gboolean rtidds_register_topic(gchar *xml_topic_data);
```

Parámetros de entrada El parámetro de entrada es un mensaje XML, que contiene información acerca de:

- Nombre del tópico
- Buddy asociado
- Cuenta asociada
- Plugin asociado
- Dominio asociado
- Estructura interna del tópico

En el listado **??** se observa con detalle el formato de mensaje XML intercambiado para registrar el tópico. Como se observa en el DTD, toda la información necesitada se encuentra en el mensaje XML.

```
<!ELEMENT topic (instance+,struct)>
<!ATTLIST topic
        name NMTOKEN #REQUIRED
        plugin NMTOKEN #REQUIRED
        buddy NMTOKEN #REQUIRED
        account NMTOKEN #REQUIRED>
<!ELEMENT struct (member+)>
<!ATTLIST
        name NMTOKEN #REQUIRED>
<!ELEMENT member (#PCDATA)>
<!ATTLIST
        name NMTOKEN #REQUIRED
        type (double|string|integer|char) #REQUIRED>
```

Listado 5.1: Estructura DTD del mensaje de registro de tópico

5.2.2. desregister_topic

Esta función deshace todo lo que lleva a cabo la función register topic.

```
void rtidds_unregister_topic(gchar *topic_name);
```

Parámetros de entrada Esta función solo requiere el nombre del tópico para el que se desea eliminar toda la información almacenada.

5.2.3. update_topic_instance

```
gboolean rtidds_update_instance(gchar *topic_name, gchar *key,gchar *value, gchar
*status);
```

Parámetros de entrada Para esta función son necesarios varios parámetros:

topic_name Nombre del tópico al que pertenece la instancia

key Clave de la instancia

value Mensaje XML que incluye los valores de la instancia. Su estructura se muestra en el listado **??**

status Nombre simbólico del estado de la instancia

```
<!ELEMENT topic (instance+,struct)>
<!ATTLIST topic
        name NMTOKEN #REQUIRED
        plugin NMTOKEN #REQUIRED
        buddy NMTOKEN #REQUIRED
        account NMTOKEN #REQUIRED>
<!ELEMENT struct (member+)>
<!ATTLIST
        name NMTOKEN #REQUIRED>
<!ELEMENT instance (member+)>
<!ATTLIST instance
        key ID #REQUIRED)
<!ELEMENT member (#PCDATA)>
<!ATTLIST
        name NMTOKEN #REQUIRED
        type (double|string|integer|char) #REQUIRED>
```

Listado 5.2: Estructura DTD del mensaje de actualización de instancia

5.2.4. asynchronous_message

```
gboolean rtidds_async_message(gchar *xml_message);
```

Parámetros de entrada Como se observa en la definición de la función, ésta solo recibe un parámetro que es un mensaje XML. En el listado **??** se muestra con más detalle la estructura del mensaje intercambiado.

```
<!ELEMENT msg (#PCDATA)>
<!ATTLIST msg
        rcpt #NMTOKEN #IMPLICIT>
```

Listado 5.3: Estructura DTD del mensaje asíncrono

5.2.5. get_qos

```
gboolean get_qos(gchar **xml_qos);
```

Parámetro de entrada Esta función recibe un único parámetro de entrada que es un puntero a una cadena, que será reservada dinámicamente por la función y que corresponde a un mensaje XML que codifique las actuales políticas QoS. En el listado **??** se propone una descripción de la estructura de dicho mensaje.

```
<!ELEMENT qos (durability,transport,lifespan)>
<!ELEMENT durability (#PCDATA)>
<!ELEMENT lifespan (#PCDATA)>
<!ELEMENT transport (#PCDATA)>
```

Listado 5.4: Estructura DTD del mensaje de consulta de QoS

Como se puede observar no se necesitan parámetros como el nombre del tópico del que queremos obtener la QoS, ya que esta función se define dentro de cada *topic-plugin*. Debido a esto es el plugin *rtidds-prpl* es el encargado a la función *get_qos* del tópico que se desea modificar.Esto ocurre igualmente en el resto de funciones IPC relacionadas con la QoS (*set_qos_deadline*).

5.2.6. set_qos_deadline

```
gboolean set_qos_deadline(glong val);
```

Parámetros de entrada Esta función recibe un único parámetro de entrada que es un valor entero, que indica el número de segundos de la política QoS *Deadline* y que se refiere al ratio de recepción de mensajes.

5.2.7. set_qos_lifespan

```
gboolean set_qos_lifespan(glong val);
```

Parámetros de entrada Esta función recibe un único parámetro de entrada que es un valor entero, que indica el número de segundos de la política QoS *Lifespan*.

5.2.8. set_qos_transport_priority

```
gboolean set_qos_transport_priority(glong val);
```

Parámetros de entrada Esta función recibe un único parámetro de entrada que es un valor entero, que indica el número de segundos de la política QoS *Transport priority*.

5.3. Mecanismo de consulta

Toda la información recibida por medio de los métodos remotos del *rtidds-prpl* es almacenada internamente como un árbol *XML*. La decisión de almacenar la información acerca del estado del *DDS* en *XML* se debe a varias razones:

- Es una tecnología estándar, que cada vez tiene mayor presencia
- Permite estructurar la información de una forma intuitiva mediante marcas
- Dispone de mecanismos de transformación a formatos *legibles*, como HTML

- Dispone de mecanismos de consulta bastante intuitiva

```
<!ELEMENT DDS domain*>
<!ELEMENT domain (id,topic*)>
<!ELEMENT topic (instance+)>
<!ATTLIST topic
        name NMTOKEN  #REQUIRED
        plugin NMTOKEN #REQUIRED
        buddy NMTOKEN #REQUIRED
        account NMTOKEN #REQUIRED>
<!ELEMENT instance (member+)>
<!ATTLIST instance
        key ID #REQUIRED)
<!ELEMENT member (#PCDATA)>
<!ATTLIST
        name NMTOKEN #REQUIRED
        type (double|string|integer|char) #REQUIRED>
```

Listado 5.5: Estructura DTD del árbol XML

Para definir la estructura interna del arbol XML, utilizaremos DTD , el mecanismo que utilizan los documentos XML para definir la estructura de un documento válido.

5.3.1. Mensajes XML

En esta sección se describe cómo se generan las respuestas *XML* que se utilizarán para generar respuestas en texto (las cuales se amplian en la próxima sección). Como se ha mencionado previamente las consultas del documento se realizan usando consultas por medio de *XPath*[21].

El modo de proceder es el siguiente:

1. El usuario realiza una consulta al buddy acerca de alguna entidad *DDS* (tópico, dominio, instancia,. ..)

2. Se analiza sintácticamente la consulta y se construye la expresión XPath coincidente con dicha consulta.

3. Se aplica dicha consulta y se construye una subárbol *XML* basado en las coincidencias en el mismo.

4. Se transforma el árbol *XML* resultante en texto legible y estructurado.

```
<!ELEMENT response (domain*,msg)>
<!ELEMENT msg (#PCDATA)>
<!ATTLIST msg
        rcpt NMTOKEN #IMPLIED>
<!ELEMENT domain (id,topic*)>
<!ELEMENT topic (instance+)>
<!ATTLIST topic
        name NMTOKEN  #REQUIRED
        plugin NMTOKEN #REQUIRED
        buddy NMTOKEN #REQUIRED
```

```
        account NMTOKEN #REQUIRED>
<!ELEMENT instance (member+)>
<!ATTLIST instance
        key ID #REQUIRED)
<!ELEMENT member (#PCDATA)>
<!ATTLIST
        name NMTOKEN #REQUIRED
        type (double|string|integer|char) #REQUIRED>
```

Listado 5.6: Estructura DTD de las respuestas

En el listado 5.2 se muestra la estructura de un mensaje de respuesta. Se puede observar que no difiere mucho de la estructura del árbol de representación interna (5.1)

```
<response>
<msg rcpt=buddy/>No results!</msg>
</response>
```

Listado 5.7: Mensaje de respuesta (no hay ocurrencias)

En los listados 5.2 y **??** se muestran los mensajes de respuesta obtenidos para una consulta sin ocurrencias y una consulta sobre dominios respectivamente.

```
<response>
<domain id="2">
<topic name="sample" buddy="sample buddy" account="test@account"
                                                    plugin="sample-topic"/>
</domain>
<domain id="1">
<topic name="sample 2" buddy="sample buddy" account="test@account"
                                                    plugin="sample2-topic"/>
</domain>
</response>
```

Listado 5.8: Mensaje de respuesta (consulta de dominios)

Para el registro de tópicos, tambien se intercambian fragmentos *XML* entre el *rtidds-prpl* y los *rtidds-topic*. En este caso los fragmentos incluyen toda la información básica acerca de cada tópico, así como de la estructura interna de las instancias (para validación semántica)

La consulta sobre el almacenamiento local del *DDS* se realiza mediante búsqueda de expresiones *XPath*. *XPath* es una metodología de acceso a nodos de un árbol *XML* por medio de cadenas que identifican a un nodo. Aprovechar esta capacidad que brinda *XML* es vital para hacer nuestra aplicación fácilmente extensible y crear nuevas consultas al estado del *DDS*. Cada vez que realizamos una consulta sobre el árbol *XML*, nuestra aplicación devuelve un subárbol *XML* con la información solicitada. Dicha información, como veremos en la próxima sección, es procesada y presentada al usuario mediante un mensaje de mensajería instantánea.

El proceso de consulta, se desencadena cuando el usuario introduce una sentencia apropiada en una conversación de mensajería instantánea. Esta sentencia es analizada y se realiza una búsqueda en el árbol *XML* de acuerdo a la misma, devolviéndose un nuevo árbol *XML* con los resultados.

5.4. Generación de mensajes

Como se ha descrito en la sección anterior, la información del *DDS* se almacena localmente mediante un árbol *XML*. Se pueden aprovechar las capacidades de transformación de árboles *XML* en otras formas mediante el empleo de *XSLT*. [22] *XSLT* es una tecnología que permite transformar texto marcado con *XML* en otra forma de representación alternativa de acuerdo a unas reglas proporcionadas.

La elección de *XSLT* además de la sencillez y la potencia que ello supone, se debe a que permite al usuario adaptar el mensaje de presentación de datos a sus necesidades. Esta adaptación se realiza únicamente cambiando un archivo de texto, con lo que el usuario no tiene que preocuparse de otras tareas más engorrosas y complicadas como recompilar la aplicación cada vez que necesite cambiar la forma de presentación.

5.4.1. Estructura del documento *XSLT*

Un documento de transformación *XSLT* es en sí mismo un documento *XML* con una estructura determinada (es un subconjunto del XML). En este caso lo que tiene que definir el XSLT es como se presentan y formatean al usuario:

- los **dominios** (*domain*),
- **tópicos** (*topic*),
- **instancias de tópicos** (*instance*),
- **mensajes al usuario** (*msg*)

A continuación se describen las directivas XSLT empleadas para los casos de interés.

Formateo de dominios

```
20 <!-- Domain Template !-->
21 <xsl:when test="domain"> <xsl:for-each select="domain">
22 <xsl:sort select="./@id"/>
23 <!-- Insert your IMHTML data here -->
24 <xsl:text>Domain </xsl:text><b><xsl:value-of select="./@id"/></b>:<br/>
25 Available topics: <xsl:for-each select="topic"> <xsl:value-of select="./@name"/>
26
27 (plugin: <xsl:value-of select="./@plugin"/>)
28 <xsl:if test="position()!=last()">, </xsl:if></xsl:for-each><br/>
29 <!-- End of IMHTML Data -->
30 </xsl:for-each>
31 </xsl:when>
```

Listado 5.9: Directivas XSLT para formateo de dominios

Formateo de tópicos

```
35 <!-- Topic Template !-->
36 <xsl:when test="topic">
37 <xsl:for-each select="topic">
38 <!-- Insert your IMHTML data here -->
39 Topico <b><xsl:value-of select="./@name"/></b> :          <br/>
40    Plugin : <xsl:value-of select="./@plugin"/><br/>
41    Buddy  : <xsl:value-of select="./@buddy"/><br/>
42    Struct : {<xsl:for-each select="./struct/member">
43                 <xsl:value-of select="./@name"/>
44                 (<xsl:value-of select="./@type"/>)
45                 <xsl:if test="position()!=last()">,</xsl:if></xsl:for-each>}<br/>
46 <!-- End of IMHTML Data -->
47 </xsl:for-each>
48 </xsl:when>
```

Listado 5.10: Directivas XSLT para formateo de tópicos

Formateo de instancias

```
51 <!-- Instance Template -->
52 <xsl:when test="instance">
53 <xsl:for-each select="instance">
54 <b>Instance[<xsl:value-of select="./@key"/>]:</b><br/>
55 <xsl:for-each select="member">
56 <xsl:value-of select="./@name"/>:<xsl:value-of select="."/><br/>
57 </xsl:for-each>
58 </xsl:for-each>
59 </xsl:when>
```

Listado 5.11: Directivas XSLT para formateo de instancias de tópicos

Formateo de mensajes

```
13 <xsl:when test="msg">
14 <!-- Insert your IMHTML data here -->
15 <font color="#FF0000"><b><xsl:value-of select="."/></b><br/></font>
16 <!-- End of IMHTML Data -->
17 </xsl:when>
```

Listado 5.12: Directivas XSLT para el formateo de mensajes

No se entrará en los detalles de cómo modificar un documento *XSLT*, por ser algo que escapa al alcance del presente documento, pero se puede observar que es bastante intuitivo.

5.5. Creación de nuevos *topic-plugin*

Como se ha explicado previamente, el mecanismo a emplear para agregar nuevos tópicos a la aplicación consiste en instalar un nuevo plugin que soporte dicho tópico. Esta flexibilidad para soportar nuevos tópicos de una manera fácil, se vería ensombrecida si crear dicho plugin supusiera demasiado esfuerzo extra. Para ello se necesita una herramienta que genere dichos plugins de una manera lo más automática posible.

El procedimiento para construir dicha herramienta debe de componerse de los siguientes pasos:

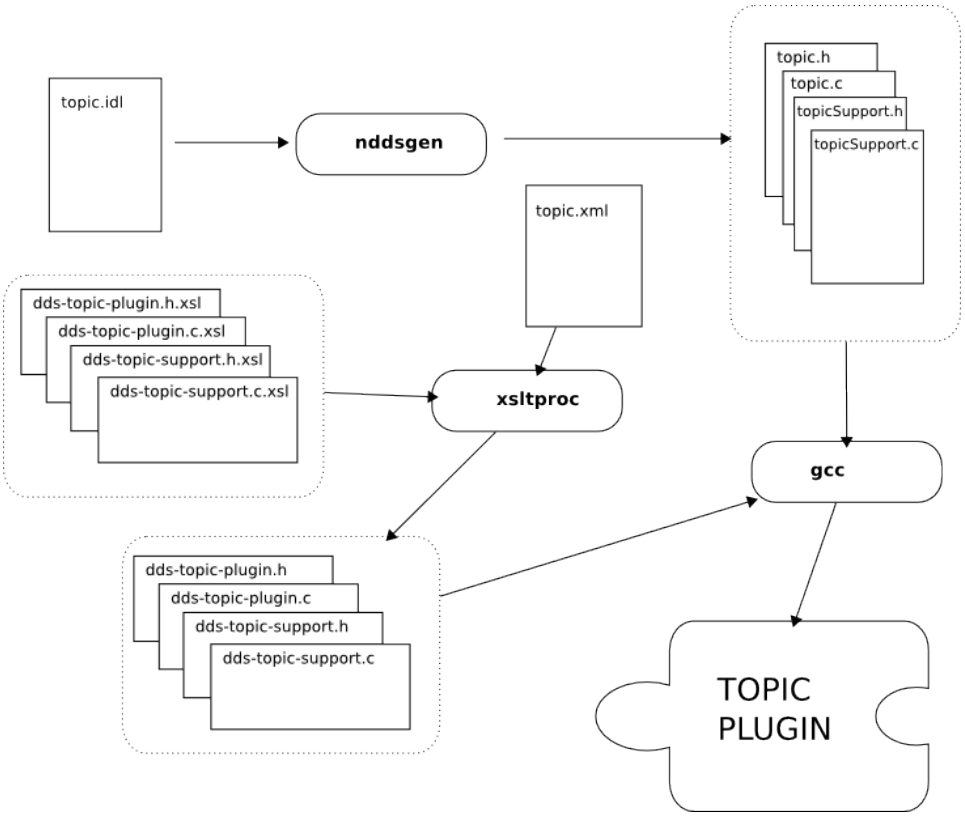


Figura 5.1: Proceso de generación de un *topic-plugin*

1. Crear un *topic-plugin* totalmente funcional y comprobar su perfecta integración con la aplicación.

2. Identificar las piezas de código reutilizables y las trozos de código que deben adaptarse en función del tópico que se deseen soportar.

3. Diseñar un mecanismo que adapte automáticamente el código según el tópico a soportar.

En resumen, lo que se busca es una herramienta de generación de código en función de una descripción de datos, tal y como lo hacen *rpcgen y nddsgen* (esta última es la herramienta proporcionada por RTI para generar código específico para un tópico). Por cuestiones de homogeneidad y lógica, la fuente de descripción de datos que utilizaremos para generar el código es un archivo IDL, tal y como lo hace *nddsgen*. En la figura 5.1 podemos observar el proceso de creación de un *topic-plugin* de una forma gráfica.

Para las cuestiones de adaptar código en función de las descripciones de tópicos mediante ficheros idl, se considera que la mejor opción es basar la generación en transformaciones *XSLT*. Para ello, hay que partir de una

descripción *XML* del tópico y su estructura. La misma herramienta *nddsgen*, proporciona una descripción XML del código, por lo se puede aprovechar dicho código XML para utilizarlo en la herramienta de generación de *topic-plugin*.

```
1
2 struct sample_topic
3 {
4         short id; //@key
5         double double_value;
6         long int_value;
7         string<255> string_value;
8 };
```

Listado 5.13: Ejemplo del documento *idl*

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <specification idlFileName="sample_topic.idl">
3  <struct name="sample_topic">
4  <member visibility="" pointer="no" enum="no" type="short" name="id"/>
5  <directive kind="key"/>
6  <member visibility="" pointer="no" enum="no" type="double" name="double_value"/>
7  <member visibility="" pointer="no" enum="no" type="long" name="int_value"/>
8  <member visibility="" pointer="no" enum="no" type="string" name="string_value"
9                                                          maxLengthString="255"/>
10 </struct>
11 </specification>
```

Listado 5.14: Documento XML generado a partir del listado 5.7

En el listado 5.7, se muestra un ejemplo de archivo IDL donde se define la estructura de un tópico. A partir de este archivo la herramienta *nddsgen*, genera código listo para ser compilado. Dicho código posee las funcionalidades necesarias para enviar/recibir tópicos de DDS, crear los DataReader/DataWriters, etc... Además la misma herramienta proporciona una descripción XML de la estructura del tópico. Para el archivo IDL de ejemplo, se puede observar una muestra del XML generado en el listado 5.8.

A la vista de todo lo anterior el procedimiento de creación del topic quedaría plasmado en un *script* como el mostrado en el listado **??**. Como se observa se siguen varios pasos:

1. En primer lugar se genera el código de acceso al DDS así como una descripción XML del archivo . idl origen. Para ello se emplea la herramienta *nddsgen*.

2. El siguiente paso es generar el código correspondiente al *topic-plugin*. En este caso empleamos la información XML generada por *nddsgen*, y se transforman las plantillas en código compilable de acuerdo a dicha información XML.

3. A continuación, se generan los archivos auxiliares, como son los *Makefiles* y otros archivos intermedios. Para este propósito se emplean las llamadas autotools (*automake, autoheader,autoconf,. ..*).

4. Se procede a la preparación del código para su compilación, para ello se ejecuta el script de configuración generado (*configure*).

Si todos estos pasos se realizaron correctamente, se obtendrá un código de *topic-plugin* totalmente compilable e instalable.

```
#!/bin/bash
set -au

export NDDSHOME=/opt/rtidds/ndds.4.1e

RTIDDSGEN=$NDDSHOME/scripts/rtiddsgen

echo Gaim DDS Plugin Generator
echo ------------------------------
echo

if [ -z "$1" ]
then
        echo You need to pass an idl file as argument
        exit
fi

# The Files
idl_file=$1
xml_file=${idl_file%%.*}.xml

echo Generating DDS archives with nddsgen...
$RTIDDSGEN -language C -xml -example i86Linux2.6gcc3.4.3 -replace $idl_file
rm makefile*
echo Creating Makefile.am...
xsltproc $NDDSHOME/resources/xml/Makefile.am.xsl $xml_file > Makefile.am
echo Creating configure.ac...
xsltproc $NDDSHOME/resources/xml/configure.ac.xsl $xml_file > configure.ac
echo Creating dds-topic-plugin.h...
xsltproc $NDDSHOME/resources/xml/dds-topic-plugin.h.xsl $xml_file > dds-topic-plugin.h
echo Creating dds-topic-plugin.c...
xsltproc $NDDSHOME/resources/xml/dds-topic-plugin.c.xsl $xml_file > dds-topic-plugin.c
echo Creating dds-topic-support.c...
xsltproc $NDDSHOME/resources/xml/dds-topic-support.c.xsl $xml_file > dds-topic-support.c

aclocal
autoheader
libtoolize
automake -ac
automake
autoconf
./configure

echo Done!
```

Listado 5.15: Código del script de generación de *topic-plugins*

CAPÍTULO 6

Pruebas

En este capítulo se realizará una descripción y realización de las pruebas llevadas a cabo en el sistema para comprobar su correcto funcionamiento y detectar el mayor número de fallos posibles. Debido a la naturaleza del proyecto, consistente en una serie de plugins que se incorporan a una aplicación, se realizarán una serie de pruebas que se centren en distintos aspectos:

- *Pruebas de instalación:* serán pruebas donde se compruebe la correcta instalación del sistema. Nótese que el proyecto consta de una serie de plugins, por lo que con este test se comprobará que se instalan en el lugar adecuado donde la aplicación pueda cargarlos.

- *Pruebas de carga:* Se comprobará que la aplicación carga los plugins que se han creado correctamente.

- *Pruebas de generación de plugins:* En esta sección se probará la correcta generación de *topic-plugins* a partir de ficheros *idl* que especifican la estructura del tópico.

- *Pruebas de funcionamiento:* Se diseñarán un par de escenarios donde se verificará el correcto funcionamiento de la aplicación y de su funcionalidad

Cada una de estas pruebas tiene sus peculiaridades en la forma de verificadas: No es lo mismo comprobar el funcionamiento de la aplicación que comprobar la instalación de cada uno de los plugins.

6.1. Pruebas de Instalación

La aplicación ha sido desarrollada utilizando herramientas disponibles en GNU/Linux como son *autoconf*, *automake* y *libtool*. Esto supone una ventaja a la hora de instalar la aplicación, ya que estas utilidades proporcionan facilidades para la compilación, así como la instalación de programas.

Gracias a esto, podemos instalar nuestra aplicación desde código fuente o desde paquetes de una forma bastante sencilla y normalizada dentro del mundo de linux.

6.1.1. Instalación desde código fuente

Para instalar nuestra aplicación desde el código fuente necesitaremos únicamente hacer los siguientes pasos (suponiendo que tengamos la aplicación comprimida en gaim-rtidds-prpl-0.1.tar.gz) :

```
$ tar -zxvf gaim-rtidds-prpl-0.1.tar.gz
$ cd gaim-rtidds-prpl-0.1/
$ . /configure
$ make
$ sudo make install # como root
```

Figura 6.1: Proceso de instalación desde código fuente

Una vez seguidos los pasos indicados en el listado 6.1.1, nuestra librería debe quedar instalada en el directorio de plugins de *Gaim* (por defecto `/usr/lib/gaim/`). En la figura 6.1.1 podemos observar que todo ha ido bien. Para ello comprobamos que se ha instalado en el directorio de plugins de *Gaim*

```
$ ls /usr/lib/gaim/librti*
/usr/lib/gaim/librtidds-prpl. la  /usr/lib/gaim/librtidds-prpl.so
```

Figura 6.2: Comprobación de la instalación correcta

6.1.2. Instalación desde paquete binario

Para aumentar la facilidad de instalación de nuestra aplicación, tambien se han creado paquetes binarios para instalar el paquete del programa directamente sin compilación previa. A continuación se muestra la instalación de dicho paquete.

```
 # sudo dpkg -i gaim-dds-prpl_1.0-1.deb
(Leyendo la base de datos . ..
276380 ficheros y directorios instalados actualmente. )
Preparando para reemplazar gaim-dds-prpl 1.0-1 (usando gaim-dds-prpl_1.0-1.deb)
```

```
Desempaquetando el reemplazo de gaim-dds-prpl . ..
Configurando gaim-dds-prpl (1.0-1) ...
 #
```

Al ser una operación de administración se necesitan privilegios de superusuario, por lo que se debe utilizar el comando `sudo` para la ejecución como superusuario. Para verificar la instalación se comprueba que se han instalado los ficheros necesarios para la aplicación correctamente:

```
 /usr/lib/gaim/# ls  *rtidds*
librtidds-prpl. la  librtidds-prpl.lai  librtidds-prpl.so  rtidds-style.xsl
```

Figura 6.3: Comprobación de la instalación del *rtidds-prpl* y la hoja de estilos

En la figura 6.3 se comprueba la instalación de los plugins, que son los ficheros `librtidds-prpl. *`, y de la hoja de estilos que corresponde al fichero `rtidds-style.xsl`. Esto se ha comprobado en el directorio de *plugins* de Gaim (`/usr/lib/gaim/`).

Una vez comprobado que se ha instalado correctamente los archivos del plugin y de la hoja de estilos, se procede a comprobar que se ha instalado el script de generación de *topic-plugins*. Para ello accedemos al directorio de scripts de nuestra distribución RTIDDS. Para ello comprobamos la existencia del archivo `rtiddsgaimgen` en el directorio de scripts de la distribución RTIDDS (`/opt/rti/ndds/scripts`).

```
/opt/rtidds/ndds/scripts # ls rtiddsgaimgen
rtiddsgaimgen
/opt/rtidds/ndds/scripts #
```

Figura 6.4: Comprobación de la instalación del script de generación de *topic-plugin*

El siguiente paso es comprobar que se han copiado las plantillas correspondientes a los ficheros fuente pra generar *topic-plugin*, que se encuentran en el directorio `resources/xml` de la distribución RTIDDS.

```
/opt/rtidds/ndds/resources/xml# ls
configure. ac.xsl        dds-topic-plugin.h.xsl   Makefile.am.xsl
dds-topic-plugin. c.xsl  dds-topic-support.c.xsl
```

Figura 6.5: Comprobación de la instalación de las plantillas de códigos fuente

En este punto se puede asegurar que los ficheros han sido correctamente instalados en el sistema. El siguiente paso es probar el correcto funcionamiento de los mismos tras la instalación, hecho que se comprueba en las siguientes secciones.

6.2. Pruebas de Carga

6.2.1. Carga del *rtidds-plugin*

El mecanismo de carga de los *protocol-plugin* se inicializa al arrancar la aplicación. Si todo ha ido correctamente, el plugin habrá sido instalado correctamente, se podrán crear cuentas de acceso al DDS y cargar *topic-plugins*.

```
$ gaim --debug | grep dds
plugins: probing /usr/lib/gaim/librtidds-prpl. so
prpl-rtidds: Init topics subsystem prpl-rtidds:  . ..OK
prpl-rtidds: Searching stylesheet in '/usr/lib/gaim/rtidds-style. xsl'
prpl-rtidds: Stylesheet successfully loaded
```

La siguiente prueba de carga del *rtidds-prpl* consiste en intentar la carga del mismo carga cuando falte algún componente que sea necesario para el funcionamiento del mismo, por lo que no debe completarse la carga del plugin. En este caso se intentará eliminar la hoja de estilo que transforma los mensajes. Esta hoja de estilo debe encontrarse almacenada en el mismo directorio que el plugin instalado, y es cargada al inicio de la carga del plugin.

En el listado 6.2.1 se comprueba el resultado que obtenemos al cargar *gaim* si no existe la hoja de estilos. Se observa que en este caso el plug-in no se carga y que por tanto es correcto, ya que no se podrían generar los mensajes correctamente, lo que haría funcionar incorrectamente nuestra aplicación. En la figura 6.8 se puede ver como no se ha cargado el plugin correctamente, no reconociendo la cuenta de DDS que ha sido creado y por tanto, no permitiendo el uso de la aplicación de acceso a DDS.

6.2.2. Carga del *topic-plugin*

La carga de un *topic-plugin* se realiza cada vez que se desea que la aplicación soporte un nuevo tipo de tópico. Dicha carga se realiza desde la

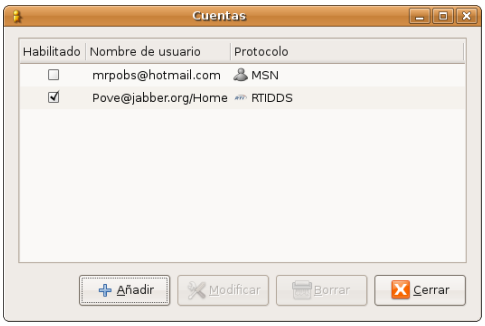


Figura 6.6: Ventana de cuentas

```
 $ gaim --debug | grep dds
I/O warning : failed to load external entity "/usr/lib/gaim/rtidds-style. xsl"
plugins: probing /usr/lib/gaim/librtidds-prpl. so
prpl-rtidds: Init topics subsystem prpl-rtidds:  . ..OK
prpl-rtidds: Searching stylesheet in '/usr/lib/gaim/rtidds-style. xsl'
prpl-rtidds: Stylesheet loading failed
```

Figura 6.7: Información de depuración cuando falta la hoja de estilos

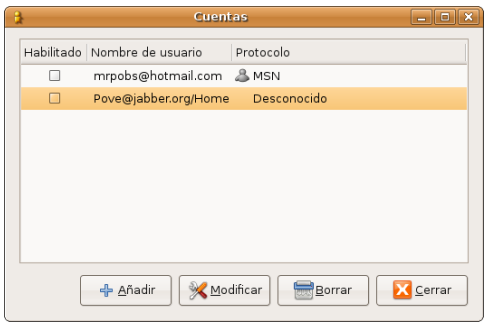


Figura 6.8: Carga de *rtidds-prpl* fallida

opción del menu `Herramientas->complementos`. En la figura 6.9, se observa dicha ventana. Al seleccionar el plugin, el sistema inicia la carga del mismo, y si esta ocurre correctamente se informa al usuario abriendo una ventana de información con el error ocurrido.

Una vez que se ha comprobado que el plugin ha sido cargado, ahora hay que comprobar que se ha registrado correctamente en el sistema. Para ello se realiza una prueba de listado de tópicos, en la que en primer lugar se solicita un listado de los tópicos antes de la carga del plugin, y en segundo lugar hacemos el mismo listado posterior a la carga del plugin. El resultado de esta operación se puede observar en la figura 6.10, comprobándose el correcto funcionamiento de la aplicación en este escenario.

6.3. Pruebas de generación de plugins

En esta sección se procederá a comprobar el proceso de generación de plugins a partir de un fichero de definición *.idl*. Tras este proceso, y partiendo de un fichero *.idl* válido, debe de generarse todo el código ya compilable, así como otros ficheros como los *Makefiles*. Para esta prueba tomaremos el fichero de prueba **??**, que dispone de la gran mayoría de los tipos de datos soportados por DDS.

```
1  struct test_topic
2  {
3          short id; //@key
4          double dbl_data;
```

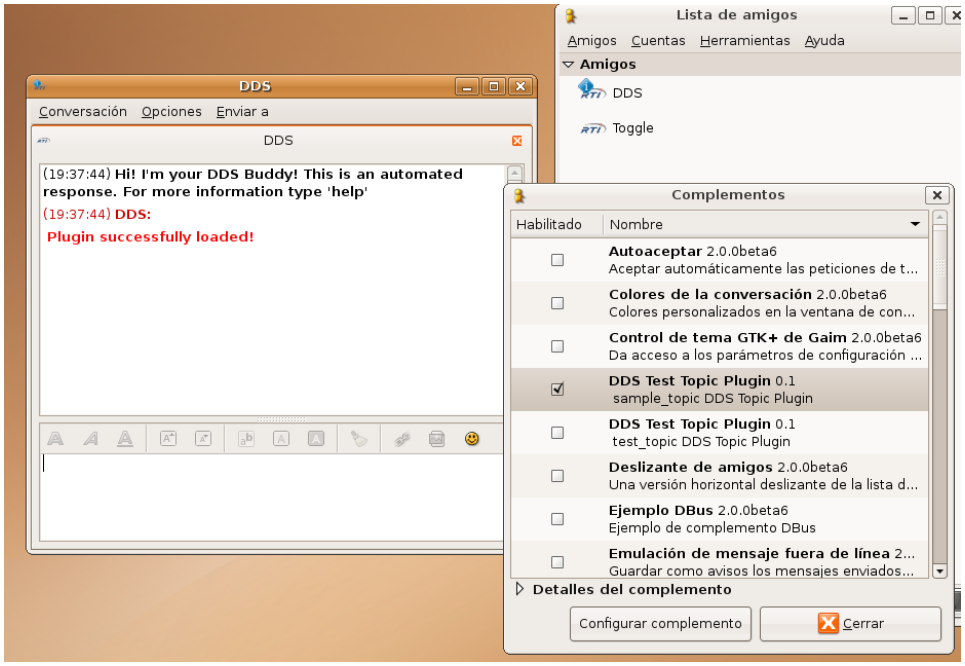


Figura 6.9: Carga de un *topic-plugin*

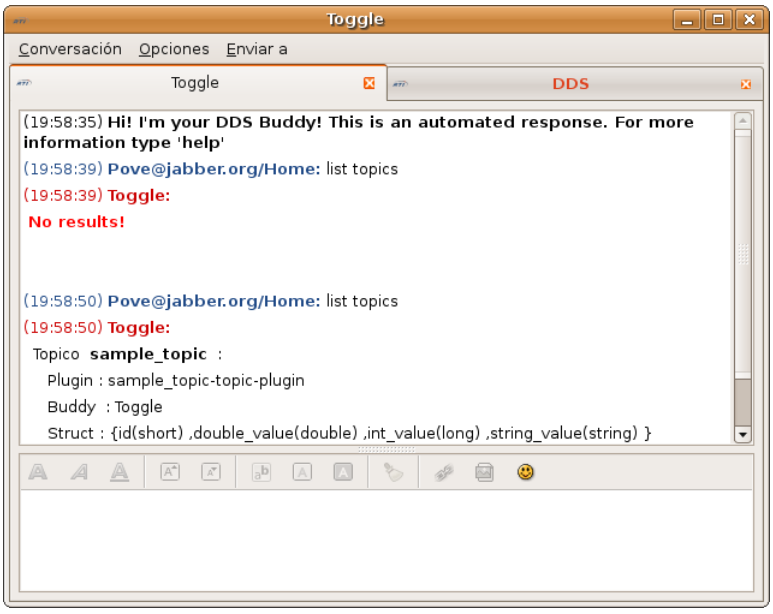


Figura 6.10: Comprobación de la carga de *topic-plugin*

```
5           short short_data;
6           string<255> str_data;
7   };
```

El proceso de generación de *topic-plugins* supone la ejecución de un script que partiendo de un fichero *.idl* generará el código fuente necesario. En la figura 6.3 se observa una traza de ejecución para el fichero `test_topic.idl` descrito anteriormente.

```
$ . /generate_plugin.sh  test_topic.idl
Gaim DDS Plugin Generator
--------------------------

Generating DDS archives with nddsgen. ..
Creating Makefile. am...
Creating dds-topic-plugin. h...
Creating dds-topic-plugin. c...
Creating dds-topic-support. c...
<. .....>
Done!
```

Figura 6.11: Generación del código del topic plugin para `test_topic.idl`

Tras la ejecución del script, en primer lugar se procede a comprobar la correcta generación de ficheros. En la figura **??** se observan los ficheros generados a partir del fichero de ejemplo `test_topic.idl`

```
# ls -a
.                config.sub           libtool       test_topic.idl
. .              configure            ltmain.sh     test_topicPlugin.c
aclocal. m4      configure.ac         Makefile      test_topicPlugin.h
autom4te. cache  dds-topic-plugin.c   Makefile.am   test_topic_publisher.c
config. guess    dds-topic-plugin.h   Makefile.in   test_topic_subscriber.c
config. h        dds-topic-support.c  missing       test_topicSupport.c
config. h.in     depcomp              stamp-h1      test_topicSupport.h
config. log      .deps                test_topic.c  test_topic.xml
config. status   install-sh           test_topic.h
```

El siguiente paso en el proceso es comoprobar que el código generado es perfectamente funcional,correcto y compilable. Para ello pasamos al siguiente paso: el de la compilación del código generado y posterior instalación. Para ello ejecutamos lo siguiente:

```
1   # make
2   # sudo make install
```

Llegados a este punto el nuevo topic plugin ha sido generado e instalado, por lo que ahora corresponde comprobar que se carga correctamente en la aplicación. Para ello lanzamos nuestra aplicación de mensajería instantánea *Gaim*, y entramos en el menú `Herramientas->complementos`. Tras abrirse la ventana, comprobamos tal y como aparece en la figura 6.12 la presencia del nuevo *topic-plugin* generado, por lo que la prueba resultó exitosa.

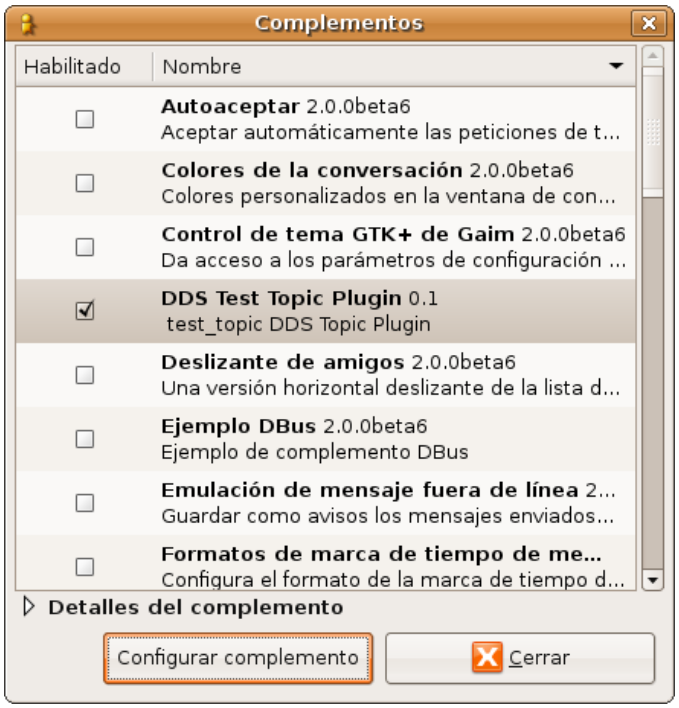


Figura 6.12: Ventana de complementos con el *topic-plugin* test_topic cargado

Llegados a este punto, se comprueba que para este caso el código generado, es correcto y funcional.

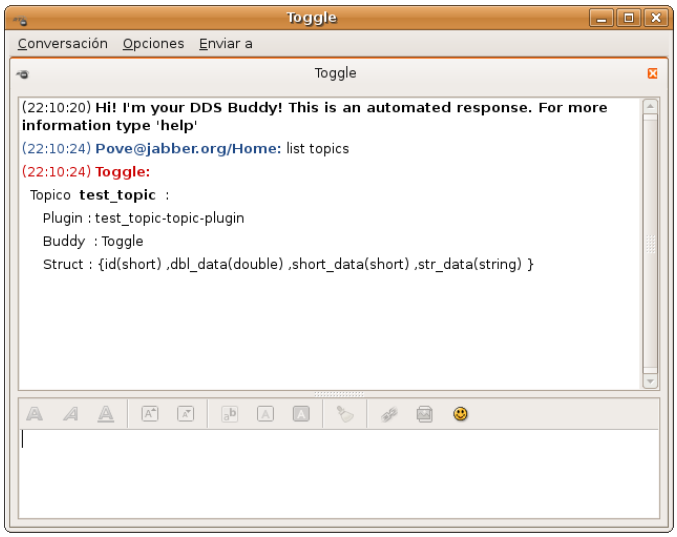


Figura 6.13: Ventana de conversación donde aparece el nuevo topic cargado

6.4. Pruebas de funcionamiento

Ya que se han realizado las anteriores pruebas, sólo resta realizar las pruebas más importantes: las de funcionamiento. En ellas se validará que la funcionalidad es correcta en varios escenarios, especialmente en aquellos más susceptibles de error. Probar el procedimiento a seguir incluye comprobar los distintos casos de uso que fueron definidos en los primeros capítulos. Más concretamente las pruebas se centrarán en los siguientes casos de uso:

- Inicio del sistema
- Carga de tópico
- Descarga de tópico
- Consultar dominios
- Consultar tópicos
- Cambiar QoS (y sus tres variantes)

6.4.1. Prueba de Inicio del sistema

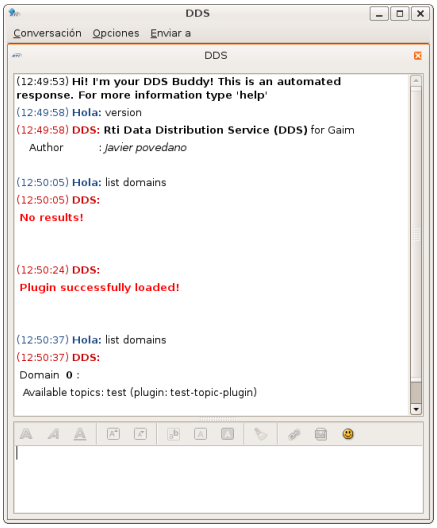
Esta prueba es la más sencilla de todas. Para llevarla a cabo hay que comprobar lo siguiente:

1. Se carga correctamente el *rtidds-prpl*
2. Se reconoce la gramática definida y proporciona respuestas adecuadas.

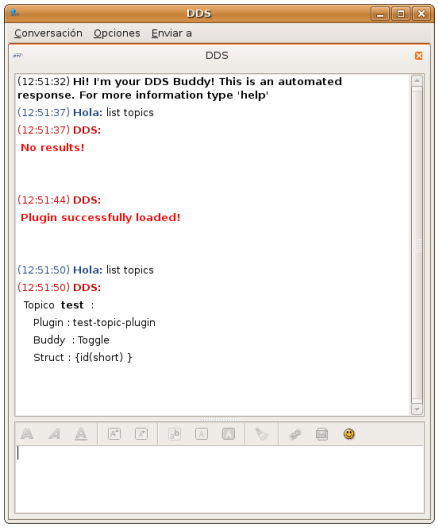
El primer punto se ha comprobado en secciones anteriores. Quedaría comprobar si la gramática que se definió en el capítulo 3 es soportada correctamente. Como muestra del correcto funcionamiento se disponen de las figuras **??**, **??** y **??**. En ellas se observa que la prueba ha sido exitosa. Se ha cargado cargado un topic plugin de nombre `test` para realizar esta prueba de funcionamiento de la gramática. Comos se observa la gramática funciona correctamente en estos casos, llevando a cabo la funcionalidad que se esperaba.

Por ejemplo, en las figuras **??**y **??** se observa como las sentencias de consulta se comportan correctamente: muestra una advertencia de que no hay resultados cuando no se encuentra ningún *topic-plugin* cargado, y muestra los datos en el caso de encontrarse cargado.

Por otro lado, en la figura **??** se observa que para un *topic-plugin* cargado las sentencias de cambio de QoS, son correctamente reconocidas y realizan correctamente lo que se esperaba de ellas.



(a) Comprobación de las sentencia de consulta de dominios



(b) Comprobación de la sentencia de consulta de tópicos

6.4.2. Carga de tópico

Aunque este caso de uso se ha comprobado ya indirectamente, en este punto haremos una prueba más exhaustiva. Para ello se llevarán a cabo una serie de pruebas que permitan verificar que la carga de un tópico se realiza correctamente. La carga de tópico implica las siguientes tareas:

Registrar tópico el tópico debe de ser reconocido por la gramática y aparecer en las sucesivas consultas al programa

Cambio de estado el buddy asociado al tópico, debe de cambiar de estado de acuerdo a la carga y descarga del mismo.

6.4.3. Descarga del plugin

Este caso de uso es el antagónico al anterior. En este caso lo que se desea es que un tópico no sea reconocido más en el sistema. Por lo tanto, la funcionalidad a llevar a cabo será la siguiente:

Desregistrar el tópico Debe de eliminarse toda la información almacenada respecto a dicho tópico, sin olvidarnos del nombre para el análisis sintáctico.

Cambio el estado El buddy encargado de dicho tópico debe de cambiar de estado si no tiene más buddies a su cargo

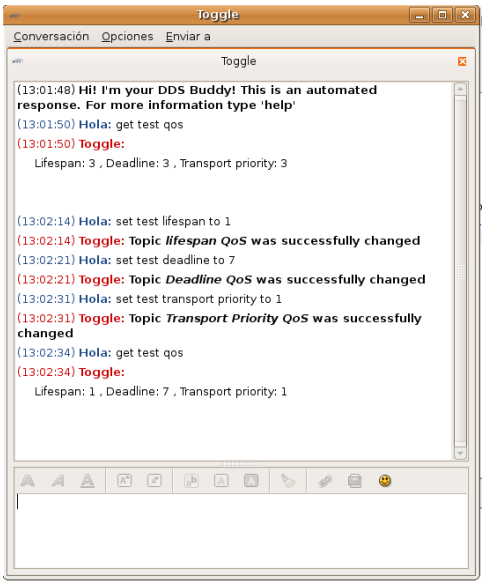
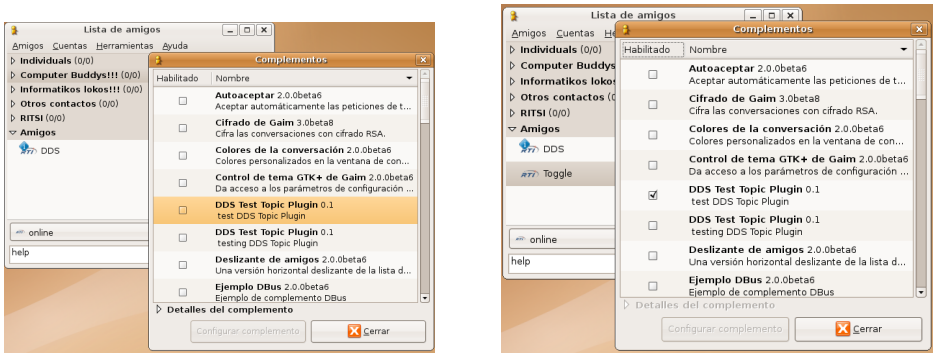


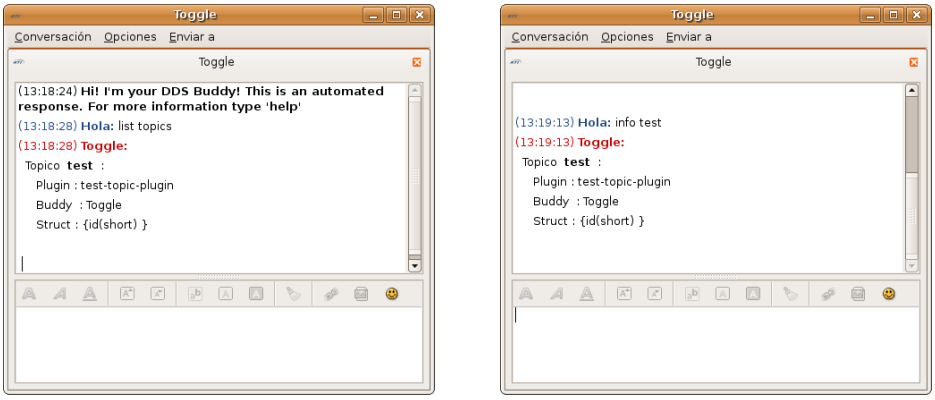
Figura 6.14: Comprobación de las sentencias de modificación de QoS



(a) Previo a la carga del *topic-plugin*

(b) Después de la carga del *topic-plugin*

Figura 6.15: Comprobación de la carga del plugin (estado de la lista de *buddies*)



(a) Previo a la carga del *topic-plugin*

(b) Después de la carga del *topic-plugin*

Figura 6.16: Comprobación de la carga del plugin (Comprobación de la sintaxis)

Para la comprobación de la descarga del plugin hemos contado con el siguiente escenario: disponemos de un único *topic-plugin* cargado, llamado *test*, que es descargado, de modo que el buddy asociado a este tópico cambiará su estado a desconectado. Las consultas posteriores a la descarga no deben de dar ningún resultado ya que no existen más tópicos cargados en el sistema. Se ha elegido este escenario por ser el más susceptible a errores cuando se realizan descargas de *topic-plugins*, ya que el sistema debe de pasar a un estado en el que no hay tópicos de dicho tipo cargados.

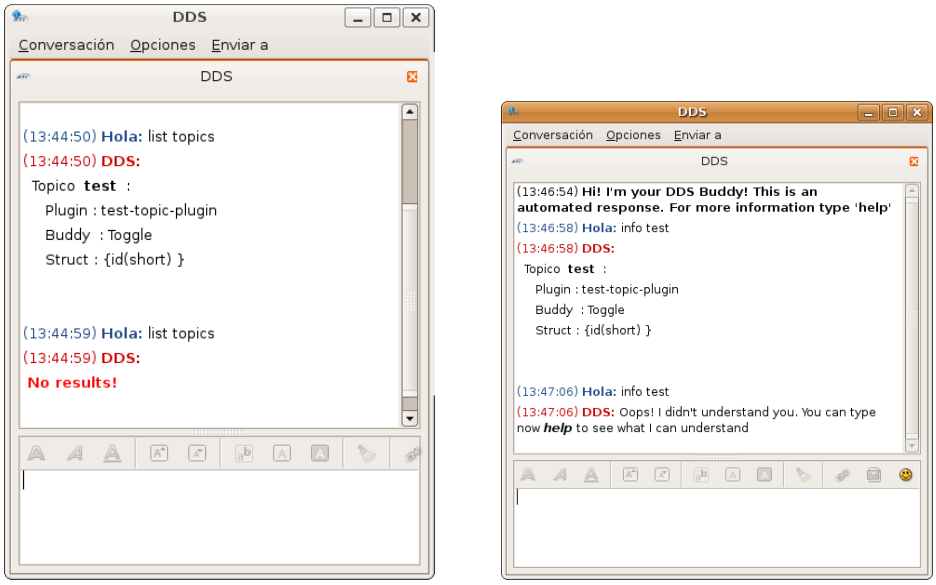
En la figuras **??** y **??** se observa que el cambio de estado es correcto para la descarga del plugin. Cuando el plugin es descargado, el buddy no dispone de más tópicos a su cargo, por lo que cambia su estado a fuera de línea.

En las figuras 6.16(a) y 6.16(b), se observa la respuesta del sistema ante la descarga de un *topic-plugin*. Como vemos en el caso de la orden `list`, la ocurrencia que se obtenía debido al tópico cargado, desaparece, no habiendo tópicos registrados en el sistema. Para el caso de la orden `info`, lo que ocurre es algo distinto, pero correcto. En este caso lo que ocurre es que el nombre del tópico (`test`) al descargarse el tópico, no se reconoce como nombre de tópico, por lo que el mensaje que anteriormente era válido, ya no lo es. En este caso la aplicación mostrará un mensaje de ayuda para orientar al usuario indicándole de que el nombre de tópico no corresponde a ninguno actualmente cargado.

(a) Previo a la descarga del *topic-plugin*

(b) Después de la descarga del *topic-plugin*

Figura 6.17: Comprobación de la descarga del plugin (estado del buddy)



(a) Comprobación de la orden `list` para el tópico test

(b) Comprobación de la orden `info` para el tópico test

Figura 6.18: Comprobación de la descarga del plugin (comprobación de sentencias)

6.4.4. Consultar dominios

Para comprobar la orden de consulta de dominios se han utilizado dos tópicos, cada uno suscrito a un dominio distinto (se han utilizado los dominios 0 y 1 por cuestiones de sencillez). Las ordenes de consulta de dominios son:

- `list domains`
- `info domain <integer>`

Para el ejemplo utilizado la asociación *tópico-dominio* empleada es:

- El tópico *test_topic* se encuentra asociado al dominio 0.
- El tópico *test_topic_2* se encuentra asociado al dominio 1.

En primer lugar se comprueba el correcto funcionamiento de la orden `list domains`, en la que se listan todos los dominios disponibles, así como información acerca de tópicos suscritos a los mismos. En la figura **??** se observa el resultado obtenido de la ejecución de dicha orden. Como se observa el resultado es correcto, ya que se observan todos los dominios activos, así como los tópicos suscritos a cada uno de ellos.

El siguiente paso es comprobar la orden de consulta de un dominio determinado por medio de la sentencia `info domain <integer>`, donde `<integer>` es el identificador de un dominio determinado. En la figura **??** se observa el resultado de la ejecución de la misma para cada uno de los dominios disponibles (hay que recordar que se han utilizado los dominios 0 y 1).

Si se contrastan los resultados de la figura **??** y **??**, se observa que coinciden en resultados, luego el funcionamiento es correcto para ambas órdenes.

6.4.5. Consultar instancias

En esta sección vamos a llevar a cabo las pruebas corespondientes a la consulta de instancias. Para ello, debemos comprobar que las consultas de instancias corresponden a la información que está siendo publicada en DDS. Para llevar la prueba a acabo, debemos tener conocimiento de cual es la información que está siendo publicada. La mejor manera de hacer esto es considerar varios publicadores deterministas (es decir, que publiquen tópicos de los que conozcamos su evolución, por ejemplo una secuencia de números).

Es interesante que además se compruebe en un escenario más o menos determinista y varios tipos de tópicos. En un entorno de pruebas con un único tópico siendo publicado, para comprobar el funcionamiento correcto de los mecanismos de consulta, obtendríamos una prueba parcial y nada útil. Para ello hemos creado al menos dos tópicos, número a partir del cual consideramos que los resultados pueden ser extrapolables. Hemos creado dos tópicos simples con nuestra herramienta, que son:

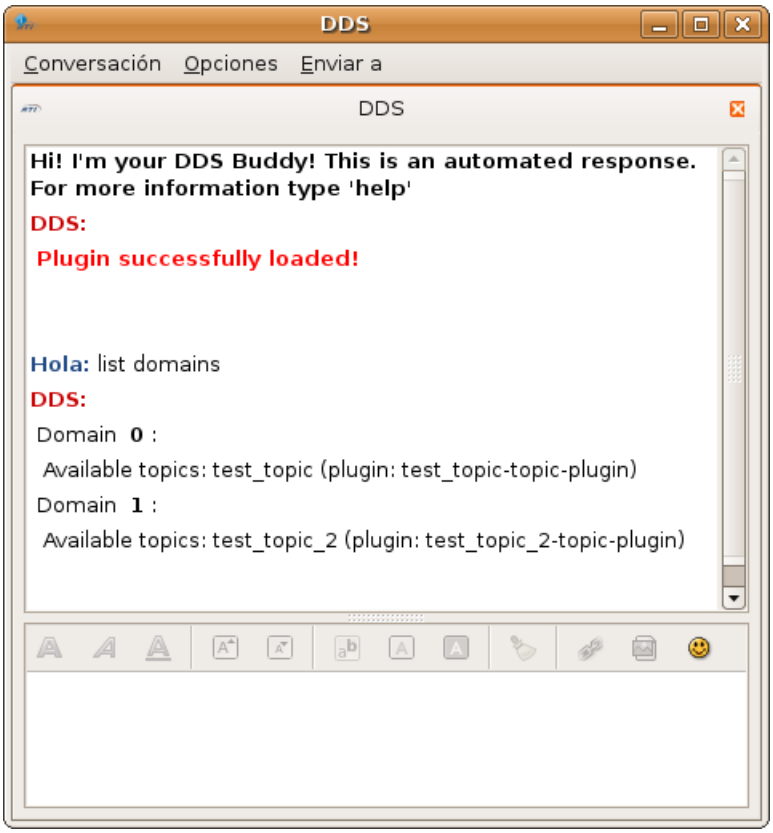


Figura 6.19: Listado de dominios disponibles

```
1 struct test
2 {
3         short keyid; //@key
4         long value1;
5 };
```

Listado 6.1: Fichero `test.idl`

```
1 struct testing
2 {
3         short keyid; //@key
4         string value1;
5 };
```

Listado 6.2: Fichero `test2.idl`

Como ya se ha visto, estos archivos 6.1 y 6.2 tienen la descripción de dos tópicos simples y convencionales, con una clave y un campo de los tipos básicos más comunes(short, string y long). A continuación lo que hacemos es iniciar una pequeña aplicación que publique instancias de dichos tópicos, para así comprobar el correcto funcionamiento de la aplicación.

Para que la publicación de dichos tópicos se pueda comprobar de una manera determinista, los valores de los tópicos se generarán de acuerdo a dos algoritmos simples, que se podemos ver en los listados **??** y **??**.

```
1  for(count=0;;count++)
2  {
3         instancia.keyid=count%3;
4         instancia.value1="Count " + count;
```

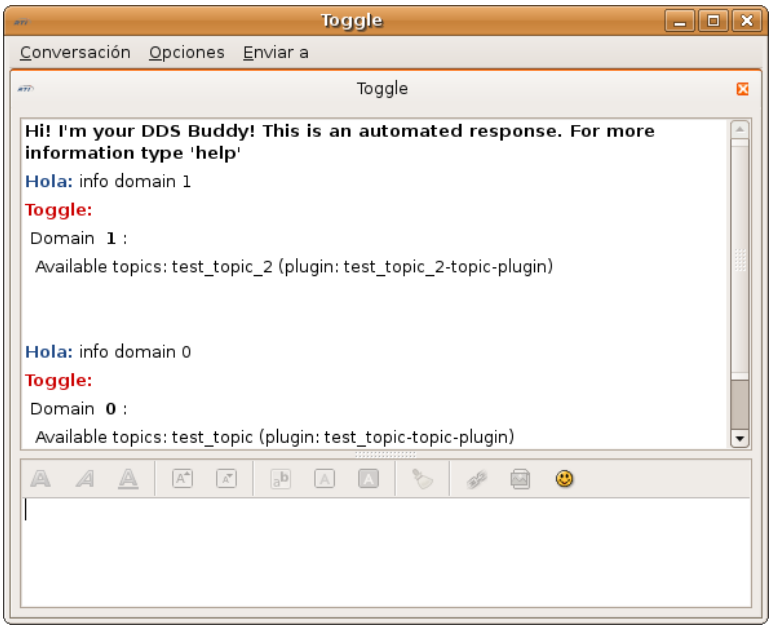


Figura 6.20: Comprobación de la orden de información de dominios

```
5   }
```

Listado 6.3: Algoritmo de publicación

```
1   for(count=0;;count++)
2   {
3           instancia.keyid=1;
4           instancia.value=count;
5           publish(instancia);
6   }
```

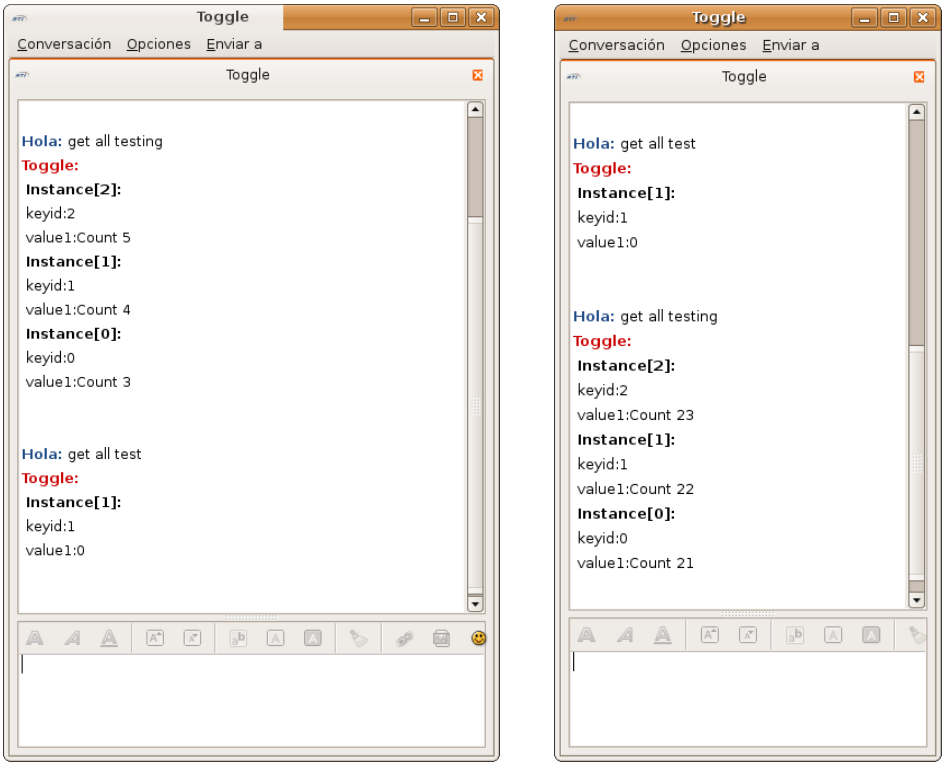
Listado 6.4: Algoritmo de publicación

En las figuras 6.17(a) y 6.17(b) se observa los resultados de la consulta en dos instantes distintos de tiempo, con lo que se puede observar el correcto funcionamiento.

6.4.6. Gestión de las políticas QoS

Este apartado se realizarán pruebas de gestión de las políticas QoS del servicio de distribución de datos. Como se mencionó en anteriores capítulos se disponen una serie de sentencias para la gestión de dichas políticas, que son:

- `get <topic-name> qos`
- `set <topic-name> lifespan to <integer>`
- `set <topic-name> deadline to <integer>`



(a) Consulta de Instancias (b) Consulta de instancias

Figura 6.21: Consulta de instancias

- `set <topic-name> transport priority to <integer>`

A continuación se mostrarán pruebas de ejecución de dichas sentencias. Para llevar a cabo dicha tarea, se hace necesario cargar un *topic-plugin*, para el que pueda modificarse pues sus políticas QoS.

En las figuras **??**, 6.18 y 6.19 se observa el funcionamiento de dichas pruebas, donde se ha cambiado cada una de ellas a valores totalmente aleatorios para comprobar el correcto funcionamiento. La comprobación se realiza mediante el empleo de la sentencia `get <topic-name> qos`.

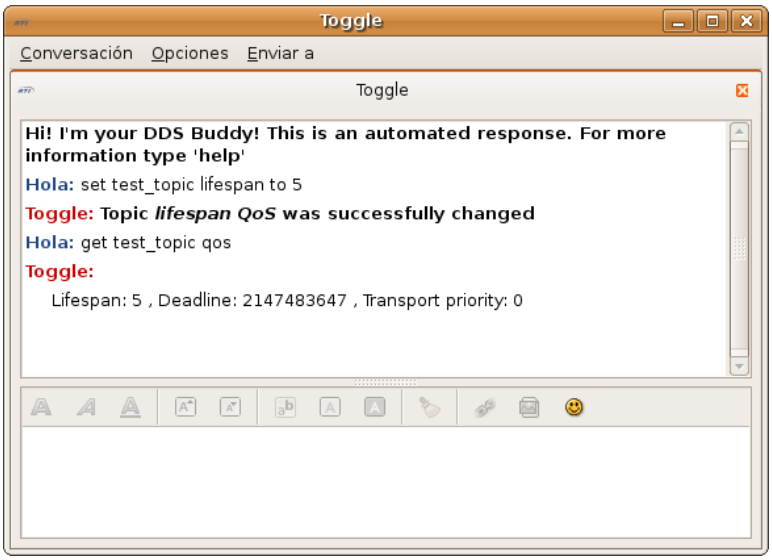


Figura 6.22: Cambio de la política *Lifespan*

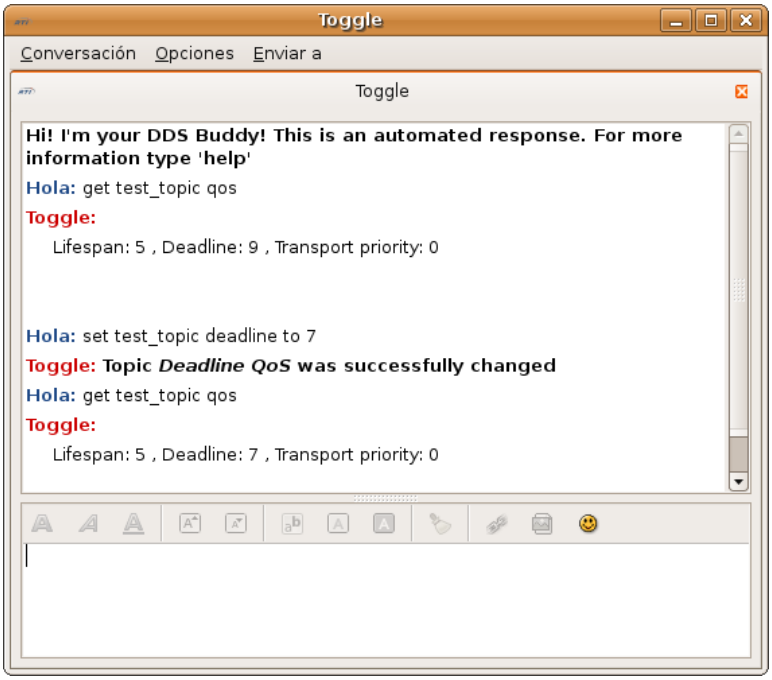


Figura 6.23: Cambio de la política *Deadline*

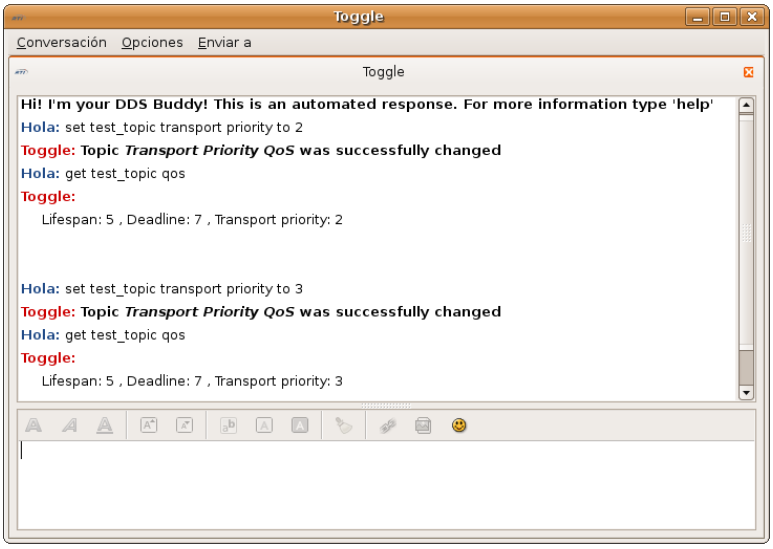


Figura 6.24: Cambio de la política *Transport Priority*

CAPÍTULO 7

Estimación de Costes

Esta sección se orienta a hacer un estudio económico de los costes de desarrollo del proyecto actual. Hay que tener en cuenta varios aspectos a la hora de estimar los costes:

- Se intentará, siempre y cuando fuera posible, el empleo de herramientas libres y gratuitas, para reducir el coste de inversión.
- El trabajo del programador se estimará en : 30 €/hora
- El trabajo del analista se estimará en : 48 €/hora

7.1. Tareas y temporización

Tarea	Descripción	Horas estimadas
1	**Estudio del estado del arte y antecedentes**	60
2	**Análisis y especificación del proyecto**	30
3	**Diseño** (total)	80
	Diseño del Plugin-Protocolo	20
	Diseño del arquitectura de plugin de tópico	40
4	**Implementación** (total)	60
	Implementación del Plugin-Protocolo	30
	Implementación del arquitectura de plugin de tópico	30
5	**Pruebas**	30
6	**Generación de la documentación**	20

Tabla 7.1: Tareas identificadas y su duración estimada

7.2. Costes

Infraestructuras necesarias Como se ha mencionado anteriormente, para el desarrollo del proyecto tendremos en cuenta que siempre que sea posible, utilizaremos programas libres y gratuitos.

En la sección 1.2 se identifican los recursos necesarios para desarrollar/explotar el software desarrollado. Teniendo en cuenta esto los costes estimados para un equipo de desarrollo queda detallado en la tabla 7.2.

Recurso	**Coste estimado**
Ordenador PC	600 €
Sistema Operativo (GNU/Linux)	0 €
Licencia de las aplicaciones empleadas	0 €
Licencia RTI-DDS	Desconocido
Total	600 € + licencia DDS €

Tabla 7.2: Costes estimados para el desarrollo por equipo empleado para el desarrollo/explotación del proyecto

Como se ha mencionado, no hay diferencias entre un equipo para el desarrollo de la aplicación y otro para la explotación de la misma. No hay requerimientos que los diferencien. La totalidad de las aplicaciones empleadas para el desarrollo/explotación del sistema (a excepción de la licencia del RTI DDS) se encuentran incluidas dentro de la distribución de GNU/Linux *Ubuntu* , que se puede obtener de manera gratuita.

7.2.1. Modelo COCOMO

Para estimar el coste del esfuerzo de desarrollo del programa (en el anterior apartado se ha estimado el coste del material necesario), utilizaremos el modelo *COCOMO* (**Co**nstructive **Co**st **Mo**del) . Este modelo es ampliamente utilizado en la estimación de costes de proyectos software. De todas las variantes existentes del modelo *COCOMO* se empleará el modelo básico, debido a que este proyecto software no es muy complejo ni tiene grandes requerimientos.

Dentro del modelo *COCOMO* básico existen tres modos de desarrollo: orgánico, semiencajado y empotrado. Debido a las características del software, se considerará que el que mejor se adecúa al mismo es el modelo *orgánico*.

En la fórmula (7.1), se puede observar la expresión del esfuerzo usando un modelo COCOMO básico.

$$E = a_i S^{b_i} m(X) \tag{7.1}$$

siendo:

- a_i, un coeficiente de COCOMO que para un modelo básico orgánico, tiene un valor de 2,4
- b_i, un coeficiente de COCOMO que para un modelo básico orgánico, tiene un valor de 1,05
- S, número de miles de líneas de código fuente, que para este proyecto se estima un valor aproximado de 2.
- $m(X)$, un coeficiente multiplicador, que en el caso del modelo básico tiene un valor igual a 1.

Teniendo en cuenta estos datos y la expresión 7.1, el proyecto abordado supondrá un esfuerzo de:

$$K_m = 2{,}4 \cdot 2^{1{,}05} = 4,96 personas/mes$$

y el tiempo de desarrollo en meses será:

$$t_d = 2{,}5 \cdot K_m \cdot 0{,}38 = 4,72 meses$$

CAPÍTULO 8

Resultados y Trabajo Futuro

8.1. Resultados obtenidos

La aplicación resultante del desarrollo de este proyecto ha resultado totalmente funcional. Los requisitos funcionales y no funcionales han sido totalmente cumplidos. Son de destacar los siguientes hechos acerca de los resultados

Funcionalidad: el sistema cumple su cometido en este aspecto, ya que permite realizar las tareas para las que fue diseñado:

- Consulta de dominios
- Consulta de tópicos
- Consulta de instancias
- Consulta de QoS
- Gestión de QoS

Portabilidad: el sistema propuesto ha sido construido con herramientas y librerías que permiten la portabilidad, por lo que este requisito ha sido cumplido tambien.

Facilidad de uso: se ha comprobado que la tanto la instalación como el uso de la aplicación es bastante sencillo. Se basa en un paradigma sencillo como es el de mensajería instantánea, y los formatos de interacción entre usuario y máquina(gramática) son igualmente simples. Otro hecho que ha favorecido a este hecho es la existencia de una interfaz conocida y amigable así como la ausencia de opciones complicadas de configuración.

8.2. Trabajo futuro

Aumento de la funcionalidad disponible Actualmente la funcionalidad implementada se corresponde con las especificaciones iniciales del proyecto. No obstante, la funcionalidad de todo proyecto en general y del software en particular siempre puede ser ampliada o mejorada. Este caso no es distinto. Para ello se proponen varias líneas de desarrollo para trabajo futuro para así aumentar la funcionalidad:

- **Soporte para nuevos tipos de políticas de calidad de servicio de DDS:** actualmente, el programa solo soporta las políticas de calidad de servicio modificables dinámicamente (Lifespan,Deadline y Transport Priority). En futuros desarrollos, sería interesante tener en cuenta alguna de las restantes políticas y habilitar mecanismos para configurarlas.

- **Mecanismos para personalización de la gramática:** Al igual que se han desarrollado herramientas que generan el código para los *topic-plugins* a partir de un fichero de descripción, se podría trabajar en generar un analizador sintáctico fácilmente configurable en acciones, o bien adaptable.

- **Configuración de la presentación de la información:** Otra alternativa sería posibilitar la modificación de forma dinámica del formato de presentación de mensajes al usuario, proporcionando alguna herramienta más o menos sencilla. Dicha herramienta podría concebirse como un nuevo plugin para la aplicación aprovechando las características de la arquitectura desarrollada.

Adaptar a otros clientes y arquitecturas nuestra propuesta Como se puede observar, la arquitectura propuesta ha sido diseñada teniendo por objetivo su portabilidad. El hecho de que esté diseñada en capas, además del empleo de librerías portables como *GLib*, proporciona a la arquitectura gran versitilidad. Más concretamente, los principios que la hacen fácilmente portable y adaptable son:

- Empleo de estándares como *XML* y *XSLT* para el tratamiento de la información

- Uso de librerías de gran portabilidad a la hora del desarrollo como es el ejemplo de *GLib*, *libxml* y *libxslt*.

- Diseño organizado en capas

- Empleo de una librería de mensajería instantánea portable

Teniendo en cuenta esto se presentan varias alternativas para la portabilidad, como posibles líneas de trabajo:

- Portar la arquitectura propuesta para un cliente totalmente distinto a Gaim, aprovechando en este caso los módulos de análisis sintáctico, almacenamiento de la información y generación de mensajes. Únicamente habría que trabajar en la arquitectura interna.
- Portar la arquitectura propuesta a un cliente de mensajería que tenga como base Gaim (es decir, que use como núcleo *gaim-core* como por ejemplo *gaim-text*). En este caso, habría que trabajar en adaptar las funciones específicas del interfaz manteniendo el resto intacto.

Mejora de la localización Esta es una aplicación que basa su interacción en mensajes escritos en un lenguaje *semi-natural*. Actualmente esta interacción se realiza mediante un lenguaje seminatural en inglés. Por tanto, otra ampliación importante o mejora sería idear un mecanismo que permitiera modificar la gramática para adaptarla a otros idiomas todo ello de una manera lo más automatizada posible y con una retoque mínimo en el código fuente. El atractivo de esta funcionalidad es que si uno de los objetivos era precisamente acercar los sistemas distribuidos a la mayoría de los usuarios, que mejor manera de acercar que hacerlo que considerando su lengua materna.

La localización actualmente se encuentra soportada en esta versión, pero sólo para los mensajes estáticos que se muestran al usuario. Por tanto la mejora que se propone sería una modificación al análisis sintáctico y a la generación de mensajes.

Parte II

Apéndice

APÉNDICE A

Manual de usuario

A.1. Instalación

A.1.1. Dependencias

Para instalar correctamente la aplicación, usted debe de tener instalados en el sistema previamente los siguientes paquetes y librerias:

gaim	libxml2-dev	libxslt1.1	automake
gaim-dev	libxml2	libxslt1-dev	autoconf
libtool	intltool	autoheader	rtidds-host
rtidds-target	libglib2.0-dev	libglib2.0	

En caso de que no tuviera instalado alguno de estos paquetes proceda a instalarlo con su gestor de paquetes favorito (p. e. `apt-get`)

A.1.2. Instalación desde código fuente

1. Descomprimir el archivo

```
1 tar -zxvf gaim-rtidds-prpl-1.0.tar.gz
```

2. Compilar

```
1 cd
2 ./configure
3 make
```

3. Instalar

```
1 sudo make install
```

A.1.3. Instalación desde paquetes (Debian based linux)

Para evitar el mayor número de problemas a la hora de la instalación, debido a la falta de librerías, dependencias y similares, se ha diseñado el sistema de modo que se pueda instalar integrado con la arquitectura de paquetes de GNU/Linux Debian, que es utilizada por la mayoría de las distribuciones actuales (Ubuntu, Knoppix,. ..). Este sistema de paquetes permite controlar las dependencias entre paquetes de una forma fácil, llegando a descargar e instalar los paquetes necesarios en caso de no encontrarse instalados.

Este sistema da una versatilidad importante, por lo que para seguir dicha filosofía se han desarrollado paquetes para integrar RTI DDS en el sistema. Se presentan dos casos posibles:

- Que la distribución RTI DDS ya se encontrara instalada en el sistema
- Que no se encuentre instalada

A.1.4. DDS previamente instalado

En este caso se ha preparado un *dummy package* para poder instalar el sistema a partir de una distribución DDS previamente instalada. De este modo se satisfarán las dependencias del *gaim-dds-prpl*, que es el que queremos instalar

```
1 dpkg -i rtidds_4.1e-1.deb
2 dpkg -i gaim-dds-prpl_1.0-1.deb
```

A.1.5. DDS no instalado previamente

En este caso no disponemos de nuestra librería

```
1 dpkg -i rtidds-host_4.1e-1_i386.deb
2 dpkg -i rtidds-target_4.1e-1_i386.deb
3 dpkg -i gaim-dds-prpl_1.0-1.deb
```

A.2. Generación de nuevos plugins

Para generar nuevos *topic-plugins* necesitamos únicamente un archivo `.idl` que defina el tópico. Con la herramienta *rtiddsgaimgen* generaremos el código necesario para crear un *topic-plugin* automáticamente, de un modo similar a como se hace con *rtiddsgen*. A continuación veremos el proceso completo:

A.2.1. Generación de archivos

En primer lugar hay que generar los archivos necesarios que conformarán el topic plugin. Para ello ejecutamos la herramienta *rtiddsgaimgen*, incluida

en el paquete *gaim-dds-prpl* previamente instalado. Para evitar confusiones crearemos previamente un directorio `<topic-plugin-dir>` que será donde se generará el código.

```
1 mkdir <topic-plugin-dir>
2 cp <archivo. idl> <topic-plugin-dir>
3 cd <topic-plugin-dir>
4 rtiddsgaimgen <archivo. idl>
```

Tras una serie de mensajes y si el proceso ha sido correcto se generará en el directorio designado el código fuente necesario, además de los Makefiles y otros ficheros. Si todo ha ido bien, el directorio actual debe de presentar una estructura similar a esta (sustituyendo 'test' por el nombre del fichero .idl sin extensión).

```
# ls
aclocal. m4      configure            ltmain.sh    test.idl
autom4te. cache  configure.ac         Makefile     testPlugin.c
config. guess    dds-topic-plugin.c   Makefile.am  testPlugin.h
config. h        dds-topic-plugin.h   Makefile.in  test_publisher.c
config. h.in     dds-topic-support.c  missing      test_subscriber.c
config. log      depcomp              stamp-h1     testSupport.c
config. status   install-sh           test.c       testSupport.h
config. sub      libtool              test.h       test.xml
```

A.2.2. Compilación

En primer lugar debe de hacer un `configure` para adecuar los Makefiles a su sistema y poder realizar la compilación correctamente.

```
1 . /configure
```

Si todo ha ido bien deberá aparecerle al final un mensaje similar a este:

```
Type make to compile

Done!
```

Si no ha sido así, deberá comprobar si se encuentran todos los paquetes y librerías necesarios instalados.

Si la compilación del código ha sido correcta puede proceder a la compilación para la posterior instalación.

```
1 make
2 sudo make install
```

Si todo ha ido bien, el plugin *rtidds-prpl*, debe de encontrarse instalado y listo para usar en Gaim.

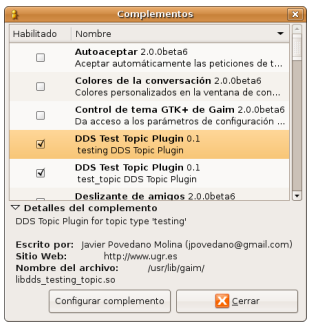
A.3. Ejecución

1. Iniciar Gaim

2. Crear una cuenta

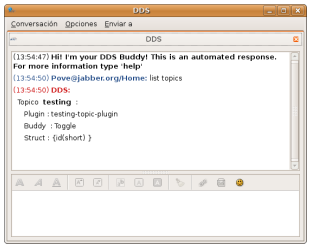
3. Ir al menú *Herramientas/Complementos*

4. Seleccionar el topic que desea instalar y si todo ha ido bien, su sistema estará listo para ser usado. Si por el contrario ha tenido algún tipo de problema, consulte la sección de *Solución de problemas*.



5. Configure dicho tópico, para ello cambie la configuración del mismo en el botón *Configurar plugin*

6. Comprobamos que el *topic-plugin* se ha cargado correctamente. (en este caso hemos instalado *test topic-plugin*



7. Para que el sistema funcione correctamente y tenga sentido, deben de existir publicadores del tópico en el mismo dominio que ha configurado para el *topic-plugin*, ya que en caso contrario no podremos listar las instancias que se publiquen.

A.4. Interaccionando con los buddies

Los plugins cargados, se encargan de ser el punto de contacto entre el usuario y DDS por medio de un lenguaje natural. Para ello únicamente hay que conversar con los buddies de tipo DDS (se identifican mediante icono DDS) haciéndoles consultas acerca del DDS. Para ello se usa una gramática bien definida. Hay varios tipos de sentencias disponibles en la gramática definida en el sistema, que son:

- Consultas de dominios
- Consultas de tópicos
- Gestión de calidad de servicio
- Sentencias de ayuda

A.4.1. Consultas de dominio

Este tipo de consultas sirven para obtener información acerca de los dominios a los que se encuentra suscrito el sistema. Para ello tenemos varias sentencias:

`list domains` Muestra una lista de los dominios disponibles en la aplicación, que serán aquellos para los cuales algún tópico participa por medio de un *topic-plugin*.

`info domain <integer>` Muestra información acerca de un determinado dominio identificado por `<integer>`.

A.4.2. Consultas de tópicos

Este tipo de consultas se utilizan para obtener toda la información acerca de los tópicos además de las e instancias disponibles en el sistema. Disponemos de varias sentencias, cada una con su funcionalidad:

`get all <topic-name>` Muestra todas las instancias de un determinado tipo de tópico `<topic-name>`

`get <topic-name>value with key <key>` Muestra la instancia de un tipo de tópico `<topic-name>` con clave de instancia `<key>`. La clave puede ser o bien un identificador o bien un número de cualquier tipo (short, long, real,...). Serían ejemplos de esta sentencia:

```
get sampletopic value with key 1
get topic2 value with key 100
```

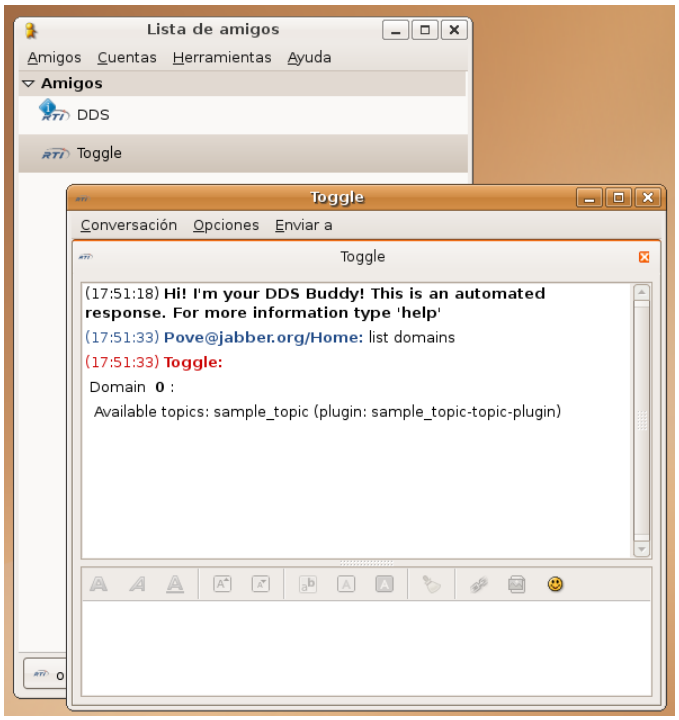


Figura A.1: Lista de dominios y topics disponibles

`info <topic-name>` Muestra información acerca de un determinado tipo de tópico `<topic-name>`

`list topics` Lista todos los tópicos registrados en el sistema. Esta orden es útil para tener conocimiento en todo momento de los topics disponibles (registrados) en el sistema para hacer consulta sobre ellos. Se proporciona un ejemplo de su uso en la figura A.2.

A.4.3. Gestión de calidad de servicio

Una de las ventajas que ofrece este sistema es que permite el acceso a ciertas políticas de calidad de servicio (QoS) mediante la gramática de interacción. Se ofrecen por tanto una serie de sentencias para acceder a ellas. En este caso las políticas disponibles son:

- Deadline
- Lifespan
- Transport Priority

A continuación se muestran todos los tipos de sentencia disponibles en la aplicación.

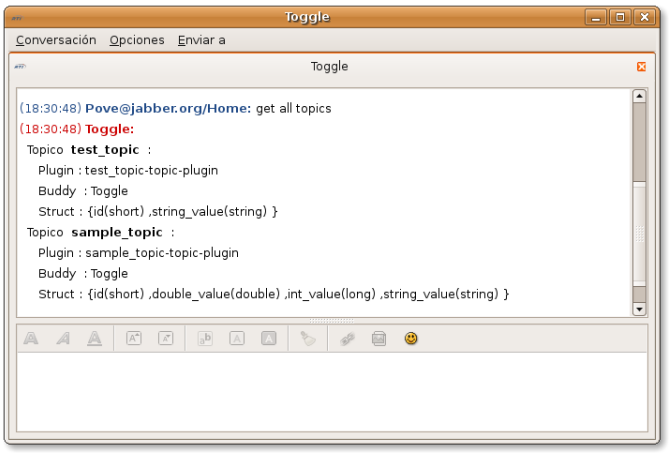


Figura A.2: Listado de tópicos (varios tópicos activos)

`get <topic-name>qos` Hace una consulta acerca de los parámetros de calidad de servicio (QoS) del tópico `<topic-name>`. Para ver una lista de los tópicos disponibles recuerde que puede hacer uso de la sentencia `list topics`. Por ejemplo si tuviéramos el tópico `sample_topic` la sentencia sería

```
get sample_topic qos
```

`set <topic-name>lifespan to <integer>` Cambia la política QoS *Lifespan* a un número de segundos igual a `<integer>`. La política *Lifespan* se refiere al tiempo que son válidos los mensajes emitidos por DDS. Dicho número entero debe de ser positivo.

```
set sampletopic lifespan to 100
```

`set <topic-name>deadline to <integer>` Cambia la política QoS *Deadline* de un tópico `<topic-name>` a un número de segundos igual a `<integer>`. La política *Deadline* se refiere al ratio mínimo de recepción de mensajes. Si tal umbral se supera, el usuario será advertido de que posiblemente un publicador haya finalizado accidentalmente.

```
set sampletopic deadline to 100
```

`set <topic-name>transport priority to <integer>` Cambia la prioridad de los mensajes acerca del topic `<topic-name>` enviados a la capa de transporte a `<integer>`. Este valor sólo tendrá sentido si la capa de transporte soporta mensajes conprioridades

```
set sampletopic transport priority to 1
```

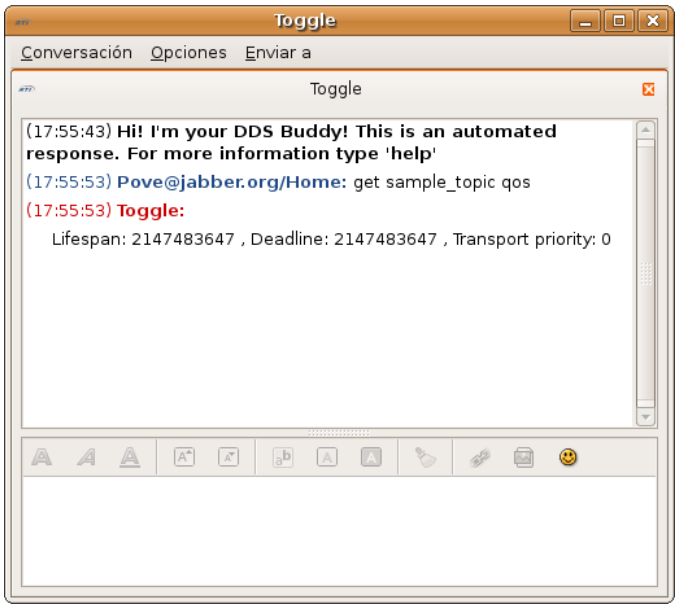


Figura A.3: Parámetros QoS de un tópico

A.4.4. Sentencias de ayuda

`help <help-topic>` Muestra ayuda contextual acerca de `<help-topic>`. Los `<help-topic>` disponibles son:

- `get`: Muestra ayuda acerca de las sentencias `get`
- `set` Muestra ayuda acerca de las sentencias tipo `set`
- `list` Muestra ayuda acerca de las sentencias de tipo `list`
- `info` Muestra ayuda acerca de las sentecias `info`
- `qos` Muestra ayuda acerca de las políticas QoS

`version` Muestra información acerca de la versión actual del programa

A.5. Solución de problemas

A.5.1. No consigo instalar gaim-dds-prpl_1.0-1.deb

Si le da un problema de dependencias, anote el/los paquetes que faltan e instálelos con su gestor de paquetes favorito. Por ejemplo, si falta libxml2-dev pruebe `apt-get install libxml2-dev`

A.5.2. No consigo instalar rtidds_4.1e-1.deb

Esto se puede deber a que ya se encuentran instalados paquetes rtidds-host y rtidds-target en su sistema.

A.5.3. El *topic plugin* x, no llega a cargar

Las causas para esto pueden ser muy diversas. En primer lugar compruebe que se encuentra instalado el plugin *rtidds-prpl* en Gaim y que existe una cuenta válida. Si esto es correcto, compruebe la compilación del *topic-plugin* generado, y busque errores. Otra posibilidad es que existan problemas con DDS: soluciónelos y vuelva a intentar cargarlo.

A.5.4. Mi *topic-plugin* x no aparece en la lista de plugins disponibles

Compruebe que su plugin se encuentra instalado en el directorio de plugins de Gaim. Para instalarlo recuerde que debe hacer `make install` en el directorio del *topic-plugin*

A.5.5. No aparecen instancias de los tópicos cargados, a pesar de que existen es funcionando

Compruebe que el tópico y el publicador se encuentran funcionando en el mismo dominio de DDS

A.5.6. No aparecen instancias de los tópicos cargados, a pesar de que existen publicadores funcionando y he comprobado que se encuentran en el mismo dominio

Configure el parámetro *INITIAL_PEERS* de DDS (Consulte el manual de usuario del RTI DDS)

BIBLIOGRAFÍA

[1] Peter Naur and Brian Randell, editors. *Software Engineering: Report on a conference sponsored by the NATO Science Committee*. NATO Scientific Affairs Division, January 1969.

[2] Pardo-Castellote G. and Farabaugh B. Introduction to DDS and Data Centric Communications, 2005.

[3] Real-Time Innovations Inc. *RTI Data Distribution Service. Users' Manual (version 4.1)*, 2007.

[4] Plato. `http://thinkofit.com/plato/dwplato.htm`.

[5] Josina M. Arfman and Peter Roden. Project athena: Supporting distributed computing at MIT. *IBM Systems Journal*, 31(3):550–563, 1992.

[6] A. Kohl, DellaFera M. W., Eichin R. S., French D. C., Jedlinsky J. T., and W. E. Sommerfeld. *The Zephir Notification Service*, 1988.

[7] P. Saint Andre Ed. Extensible messaging and presence protocol (xmpp): Instant messaging and presence. Internet Request For Comments, October 2004.

[8] P. Saint Andre Ed. Extensible messaging and presence protocol (xmpp): Core. Internet Request For Comments, October 2004.

[9] Oscar. `http://iserverd.khstu.ru/oscar`.

[10] J. Oikarinen and D. Reed. Internet relay chat protocol. Internet Request For Comments, 1993.

[11] R. Streeter A. Collins, P. Blackey. A distributed anonymous private messenger. http://www.cs.kent.ac.uk/pubs/ug/2005/co600/dapim/report.pdf.

[12] PPServer. Onobee suite. http://www.ppserver.com.

[13] Hushmail. Hush messenger. https://www.hushmail.com/services.php?subloc=messenger.

[14] Lakaim distributed messenger. http://lakaim.dev.java.net.

[15] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashock, and Hari Balakrishman. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *Proceedings of the ACM SIGCOMM*, pages 149–160, San Diego, California, Agosto 2001.

[16] Miranda IM Team. Miranda im. http://miranda-im.org/.

[17] Cerulean Studios. Trillian. http://www.ceruleanstudios.com.

[18] Gaim Team. Gaim instant messenger. http://sourceforge.net/projects/gaim/.

[19] Gnu Project. http://www.gnu.org/software/bison/.

[20] Extensible markup language (XML) 1.0. Technical Report REC-xml-19980210, W3C, August 2006.

[21] XML Path Language (XPath) Version 1.0, W3C Recommendation, November 1999. http://www.w3c.org/TR/xpath.

[22] James Clark. XSL transformations (XSLT) version 1.1. World Wide Web Consortium, Working Draft WD-xslt11-20010824, August 2001.

[23] *Client/Server Economics Letter*, volume 2, June 1995.

[24] Mirabilis Inc. Icq messenger. http://www.icq.com.

[25] Hush messenger. https://www.hushmail.com/services.php?subloc=messenger.

[26] IEEE. Std. 1003.1-2001 standard for information technology – portable operating system interface (posix), 2003.

[27] OMG. Data distribution system for real-time systems specification, Diciembre 2005.

[28] Gadu gadu. http://es.wikipedia.org/wiki/Gadu-Gadu.

[29] S. Shepler et al. Nfs network file system (nfs) version 4 protocol. RFC 3530, Abril 2003.

[30] Sun Microsystems Inc. Rpc: Remote procedure call protocol specification: Version 2. RFC 1057, June 1988.

[31] Sun Microsystems Inc. Xdr: External data representation standard. RFC 1014, Junio 1987.

[32] Michael Kay. XSL transformations (XSLT) version 2.0. World Wide Web Consortium, Recommendation REC-xslt20-20070123, January 2007.

INDEX