El modelo de comunicaciones DCPS

Autor:

Jose Luis Poza Luján

Revisores:

José Enrique Simó Ten

Juan Luis Posadas Yagüe

Instituto de Automática e Informática Industrial

(ai2)

Universidad Politécnica de Valencia

(UPV)

Versión: 0.3

Fecha: 10 de noviembre de 2009

Contenidos

Figuras

Tablas

Ecuaciones

1 Introducción

1.1 Resumen

De entre la gran cantidad de modelos de comunicaciones existentes, el modelo DCPS propuesto por la OMG es uno de los más completos en lo que el soporte a la calidad de servicio. Es por ello que se hace recomendable revisar con detalle los componentes y características del modelo. En el presente documento se revisan las características del modelo DCPS.

El modelo DCPS es un modelo de comunicaciones basado en el paradigma de publicación – subscripción. Sin embargo, en lo que más destaca el modelo es especialmente por el soporte a la calidad de servicio. En el documento se detallan todas las políticas de calidad de servicio organizadas en cuatro áreas: meta datos, aspectos temporales, gestión de flujo de mensajes y gestión de componentes.

1.2 Historial de revisiones

Nº revisión	Fecha	Comentarios
0.0	2007-10	Inicio del documento
0.1	2007-12	Inclusión de las políticas de calidad de servicio.
0.2	2008-04	Análisis del modelo DCPS
0.3	2009-07	Revisión global del documento.

1.3 Objetivos del documento

El objetivo principal del documento es ofrecer una visión detallada del modelo DCPS centrándose especialmente en lo respectivo a las políticas de calidad de servicio que éste modelo cubre.

1.4 Alcance y audiencia

El documento presenta una visión de los componentes del modelo DCPS, además se presentan con gran detalle, tanto conceptual como formal de las políticas de calidad de servicio.

El documento está dirigido a aquellas personas que quieran conocer el modelo DCPS y especialmente conocer con detalle las políticas de calidad de servicio que el modelo propone.

1.5 Organización del documento

El capitulo 2 ofrece una visión del modelo DCPS desde diversos puntos de vista, tanto la visión general del modelo como la especificación formal en UML. El siguiente capítulo, el 3, se centra en los detalles de las políticas de calidad de servicio organizadas en cuatro áreas generales (metadatos, aspectos temporales, gestión del flujo de mensajes y gestión de los componentes). A continuación, en el capítulo 4 se realiza un análisis de las funciones del modelo DCPS. Finalmente se exponen las conclusiones.

2 El modelo de comunicaciones DCPS

2.1 Ámbito

El paradigma de comunicaciones que posiblemente ofrece mejor infraestructura a los sistemas distribuidos de control inteligente es el de publicación-suscripción. Además del paradigma de comunicaciones, también se ha comprobado que el soporte a la calidad de servicio es cada vez más necesario en un sistema de comunicaciones que ejerza de middleware en un sistema distribuido. Por tanto, es necesario buscar modelos ya existentes con soporte a la calidad de servicio.

En los sistemas, y arquitecturas de comunicaciones expuestas anteriormente, la calidad de servicio, no es uno de los aspectos más tratados. Esto no sorprende cuando se trata de sistemas de comunicaciones basados en paso de mensajes, ya que el empleo de colas de mensajes o el uso de servidores y servicios en los nodos permiten llevar un control de diversos aspectos de la comunicación orientados a obtener unos parámetros de rendimiento que permitan ofrecer una calidad en las comunicaciones que cumpla los requisitos de los usuarios. A medida que se van haciendo más complejos los middlewares y se avanza en la arquitectura de los mismos van apareciendo diversos soportes a la calidad de servicio, en la medida en que actúan dichos middlewares [Aurrecoechea et al., 1998]

Una primera evaluación, más conceptual que orientada a indicadores, se puede encontrar en [Matteucci, 2003], donde se evalúan los middlewares basados en el modelo de publicación-suscripción en el ámbito de la robótica, pero extensible a los sistemas distribuidos de control inteligente. Este análisis es especialmente interesante en cuanto que conecta los indicadores de calidad de servicio vistos anteriormente, con las políticas de calidad de servicio que se verán posteriormente. De tal forma que definen unas áreas que las calidades de servicio deberán cubrir y que son las siguientes.

- Soporte a la entrega (*delivery support*).
    - Mejor esfuerzo (*best effort*)
    - Entrega garantizada (*guaranteed delivery*)
    - Entrega ordenada (*ordered delivery*)
- Soporte a la prioridad (*priority support*)
- Soporte a la movilidad (*movility support*)
- Soporte al tiempo real (*real-time support*).

No necesariamente esta aproximación es ideal, ya que ciertos aspectos como la seguridad o la redundancia en la información, no son contemplados en ésta aproximación. De entre los sistemas de comunicaciones, basados en el modelo de publicación-suscripción, con soporte a la calidad de servicio el basado en el modelo DDS de OMG es posiblemente, el que proporciona más soporte a poder establecer políticas de gestión de calidad de servicio con garantías a los componentes [Matteucci, 2003]

2.2 Modelo

El modelo *Data Distribution Service* (DDS) es una especificación para sistemas de distribución de datos basada en el modelo de publicación-subscripción propuesto por el *Object Management Group* (OMG) para las comunicaciones con soporte a la calidad de servicio [Pardo-Castellote, 2003]. Éste modelo cubre sistemas de comunicaciones con necesidades de tiempo real estricto (aunque dependiente del soporte de comunicaciones que se tenga por debajo) hasta sistemas sin necesidades de calidad de servicio [OMG, 2005]. El modelo se basa en el paradigma de publicación-suscripción que conecta a productores de información (publicadores) con consumidores de información (suscriptores) desacoplándolos en tiempo, espacio y flujo [Houston, 1998] con la comunicación basada en la negociación y gestión de las calidades de servicio.

DDS se divide en dos capas el *Data-Centric Publish-Subscribe* (DCPS) que es el responsable de la distribución de los datos, y el *Data Local Reconstruction Layer* (DLRL) que es la capa responsable de adaptar los datos a las aplicaciones locales. En un sistema DDS la capa DCPS es obligatoria, mientras que la capa DLRL es opcional dependiendo de la necesidad de adaptación de la información que tengan los componentes que se encuentren por encima del middleware.

La primera versión de DDS, la 1.0, data de diciembre de 2004, mientras que a fecha de hoy la versión más reciente es la 1.2 de marzo de 2007, en el resto del apartado se analizarán los detalles de la última versión. Actualmente hay diversos proyectos que implementan parte o la totalidad del modelo DDS, estos proyectos son los siguientes.

- Código abierto. Proyectos públicos o de investigación de código libre o con fines no comerciales.
    - OpenDDS. Es un producto de la compañía Object Computing Inc. Se puede ampliar información u obtener el código desde la Web de la compañía: www.ociweb.com.
    - RTjDDS: son las siglas de "*DDS on High-Assurance Java*". Es un proyecto de código abierto, financiado por "Iniziativa Software" y desarrollado por "*Vincenzo Caruso*". Es posible obtener los detalles en la Web: http://sourceforge.net/projects/rtjdds
- Productos comerciales
    - RTI NDDS: de la compañía Real-Time Innovations, Inc. Es posiblemente el producto basado en DDS más extendido. Contienen una amplia documentación en su Web: www.rti.com
    - OpenSplice DDS: Es un producto de la compañía Prism Tech. Junto con NDDS es uno de los más empleados. Se puede obtener más información en la Web: www.prismtech.com.
    - MilSOFT DDS: de la compañía turca MilSoft, los detalles del sistema se pueden obtener de la Web: dds.milsoft.com.tr
    - CoreDX: Es una implementación de DDS centrada en el DCPS, los detalles se obtienen de: http://www.twinoakscomputing.com/coredx.php
    - Component-Based CORBA+DDS: Un sistema integrado basdo en CORBA y DDS. Algunos detalles se pueden obtener de la Web: http://www.pocomatic.com/docs/whitepapers/corba/.

En la capa DCPS, las políticas de calidad de servicio (QoS) se emplean para describir el comportamiento de una conexión entre los productores y los consumidores. En la capa DCPS de DDS es donde se especifican las políticas de calidad de servicio. Esta capa contiene una serie de componentes, de entre los cuales, los más relevantes se verán en el siguiente apartado.

2.3 Componentes

La capa DCPS del modelo DDS tiene una gran cantidad de componentes, aunque conceptualmente son unos pocos los que intervienen en la comunicación. Por ello, parece conveniente exponer inicialmente el modelo conceptual y así localizar el papel que cada componente realiza. A continuación se realizará la revisión detallada del modelo UML [UML Group, 1997], ya que éste aporta una organización muy sensata de los componentes desde el punto de vista informático.

2.3.1 Modelo conceptual

En la figura 1, se observan los componentes y las interacciones entre ellos de la capa DCPS del modelo DDS. Los productores son las aplicaciones, o partes de la aplicación, que producen información. La producción de información puede ser bajo demanda, es decir a consecuencia de una solicitud recibida, o por propia iniciativa, a consecuencia del resultado de un proceso o la necesidad de información para realizar un proceso. Los consumidores de información son los que reciben datos de los productores, bien como respuesta de una solicitud previa o como consecuencia de un evento.

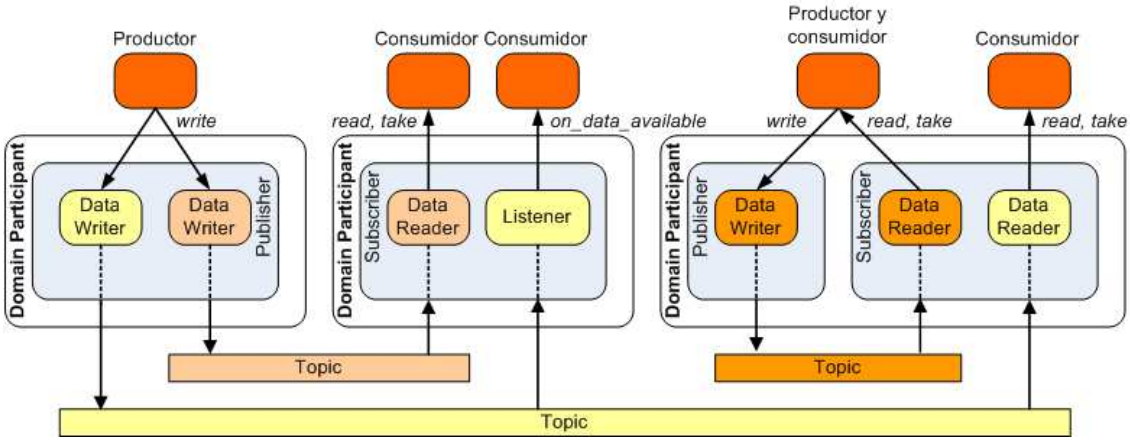

Figura 1. Elementos del modelo de comunicaciones DCPS de DDS.

Para interconectar productores y consumidores, el modelo DCPS proporciona una serie de componentes con diversos papeles. Desde el punto de vista de la capa DCPS los productores se consideran escritores de datos y los consumidores se consideran lectores de datos. A continuación se da una breve descripción de cada componente, para seguidamente describir el comportamiento.

- Domain. Construcción básica que une a las aplicaciones individuales con las comunicaciones. Todos los escritores o lectores de datos, se comunican en el mismo dominio. En DDS se pueden crear múltiples dominios, aislados unos de otros.
- Domain Participant. También se le conoce como "Participant". Las aplicaciones emplean un objeto llamado "DomainParticipant" que representa las actividades de la aplicación dentro de un dominio. Este objeto permite especificar unos valores de QoS por defecto para todos los escritores, lectores, suscriptores y publicadores, en el dominio en el que se definen.

- DataReader. Es el punto inicial de acceso para una aplicación a los datos recibidos por un "subscriber". Una vez se ha creado y configurado con la correspondiente QoS, una aplicación puede ser notificada de los datos disponibles por medio de dos modos.
    - Muestreo. A través de la lectura de los datos a iniciativa de la aplicación. El acceso a los datos se realiza mediante dos métodos: take() y read(). El primero elimina el dato del middleware, una vez es leido. El segundo método deja el dato en el middleware para poder ser leído en más de una ocasión.
    - Callback. En este modo, la aplicación recibe un aviso cuando el "DataReader" recibe un mensaje. Por medio de la combinación de "Conditions", s epueden establecer reglas de filtrado de mensajes para avisar sólo de aquellos que cumplan ciertos criterios.
- DataWriter Es el punto de acceso de una aplicación para publicar datos en DDS. Una vez se han configurado todos los parámetros de QoS, una aplicación sólo debe llamar a la función "write" de un objeto Writer para enviar la información.
- Topic Es el mecanismo por medio del cual los productores y consumidores se comunican, constituye el punto de conexión básico entre publicadores y suscriptores. Debe tener un "Topic name" y un "Topic type".El "Topic name" es una cadena de caracteres que identifica unívocamente al "Topic" dentro de un dominio. El "Topic type" es un tipo que define el contenido del "Topic". Los tipos que se pueden emplear son los definidos por las especificaciones IDL. Char, Octet, etc.
- Topic Key. Dentro de un "Topic", se puede escoger uno de los campos internos como clave (Key), este campo se emplea para ordenar los datos entrantes, de manera que se puede utilizar para seleccionar datos. El contenedor que almacena un "Key", se define como "Instance".
- Publisher. Es una entidad que contiene un grupo conjunto de "Data-Writers" individuales. Un desarrollador puede especificar un comportamiento por defecto basado en los parámetros de QoS, que se aplicará a los "Data-Writers" que pertenezcan al "Publisher".
- Subscriber. Los "subscribers" se emplean para agrupar "Data-Readers", esto permite configurar QoS por defecto al grupo de Data-Readers asociado a un "Subscriber".

Es importante entender el papel de cada uno de estos componentes ya pueden parecer escasos o excesivo y sin embargo tienen una ubicación y misión muy concreta dentro del sistema. Partiendo del "Topic" se puede justificar el resto de componentes. El "Topic" es una forma de localizar la información en el sistema, similar al nombre de una variable dentro del código de un programa.

Quienes escriben y leen en un "Topic" son el "Publisher" y "Subscriber" respectivamente. Estos dos últimos componentes de comunicación son el nexo con los componentes que quieran emplear el sistema de comunicaciones, es decir el "Publisher" y el "Subscriber" son la interfaz que el sistema de control ve del sistema de comunicaciones.

Los componentes “Publisher” y “Subscriber” residen en un “Data-Writer” y un “Data-Reader” respectivamente, éstos últimos dos componentes de comunicaciones son los que conocen al medio de comunicación y se encargan de realizar las operaciones de comunicación propiamente dichas.

Ésta separación entre los componentes que dialogan con el control y los que dialogan con el medio de comunicación permite organizar y gestionar las comunicaciones de manera eficiente, y dota al sistema del concepto de middleware que aísla el control de las comunicaciones y viceversa.

2.3.2 Modelo formal

La visión general del modelo, por medio de un diagrama de clases en UML, con comentarios, es la que se ve en la figura 2.

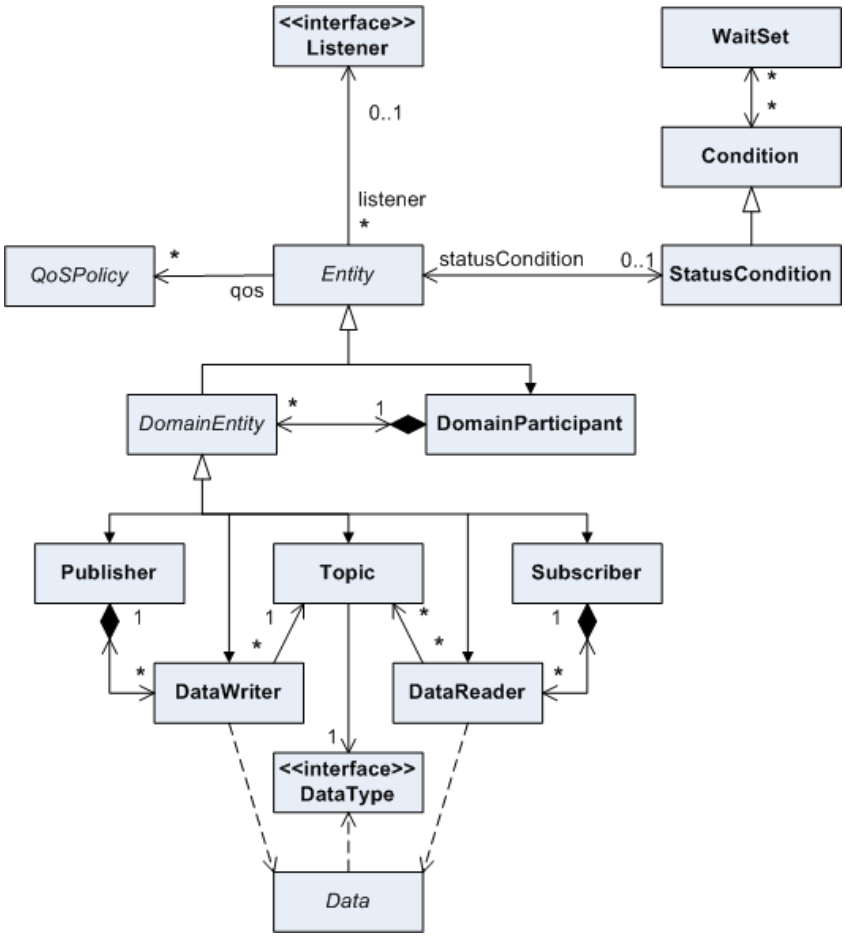

Figura 2. Componentes del modelo UML del DCPS de DDS.

Algunas de las relaciones anteriores tienen un cierto interés. A continuación se van a ver con detalle estas relaciones que condicionan la creación de un sistema que cumpla el modelo DDS.

2.4 Características

DDS destaca porque organiza las comunicaciones enfatizando diversos aspectos fundamentales para la comunicación de los sistemas. Estos aspectos son los siguientes.

- El sistema de las comunicaciones
- El sistema de soporte a las condiciones
- El sistema de soporte a la calidad de servicio

Cada uno de estos aspectos comprende una serie de clases, aunque en el DCPS de DDS se organicen de una forma diferente, inicialmente se pueden ver los dominios en el diagrama de clase como los siguientes. En la figura 3, se pueden observar las clases que comprenden cada uno de los sistemas.

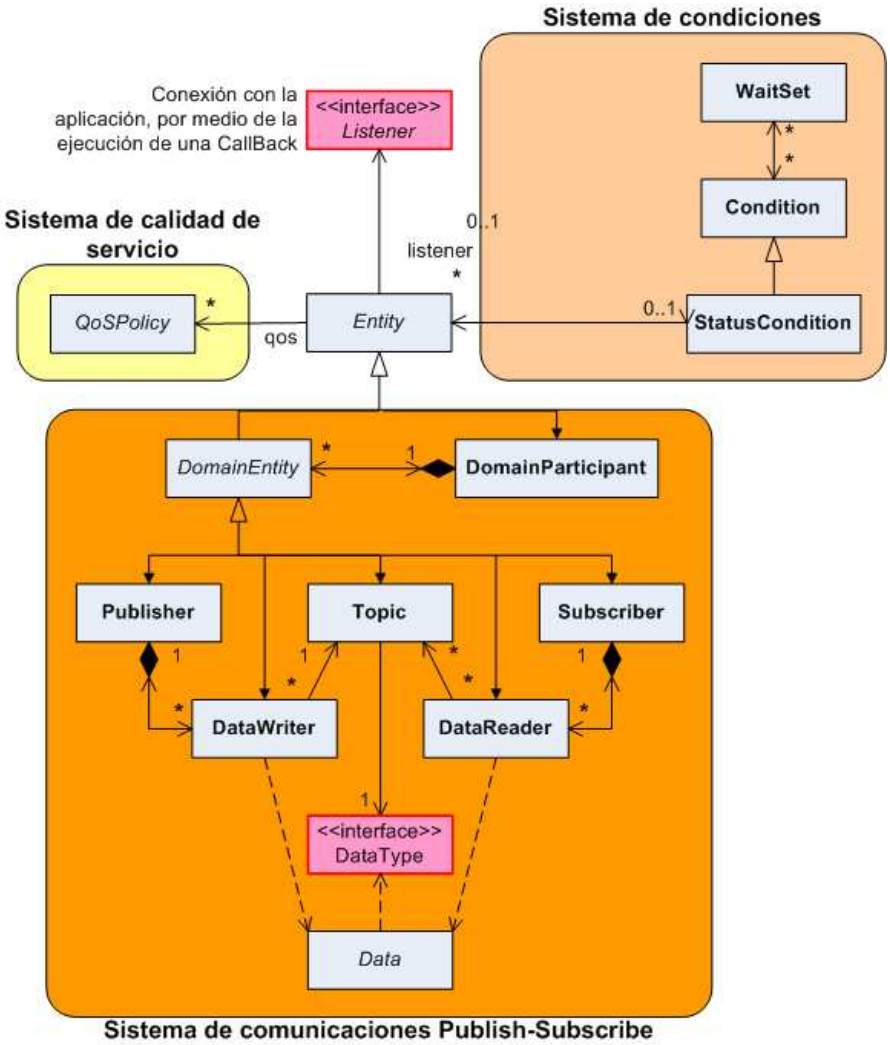

Figura 3. Componentes del modelo UML del DCPS de DDS.

La organización de módulos que realiza DDS del DCPS no se corresponde con la división funcional descrita anteriormente, aunque existe cierta relación que se puede ver a continuación.

Tabla 1. Organización en sistemas de los módulos de DDS

Subsistemas	Módulos DDS incluidos
Sistema de condiciones	Infraestructure Module
Sistema de calidad de servicio	
Sistema de comunicaciones	Domain Module
	Topic Module
	Publication Module
	Subscription Module

Como los tres aspectos contemplados en los sistemas son los que caracterizan a un sistema de comunicaciones que cumpla con la filosofía de DDS se analizarán a continuación. En la tabla 1, se distinguen el ámbito y funcionalidad de cada uno de los componentes anteriormente descritos.

2.4.1 Sistema de comunicaciones

El dominio de las comunicaciones es el que soporta el sistema publish-subscribe en el que se fundamente DDS. A este dominio pertenece todo lo que se hereda de la clase Entity, de manera que los elementos de las comunicaciones son los Topics, DataReaders, DataWriters, Publishers y Subscribers. El conjunto de todos ellos son los que forman el DomainParticipant, que se puede ver como el marco del modelo FSA, es decir un conjunto de elementos que dan soporte a las comunicaciones.

2.4.2 Soporte a eventos condicionales

Las comunicaciones en DDS permiten incluir condiciones por medio de las cuales los participantes del dominio de las comunicaciones (el grupo anterior) esperan que se den para participar en la misma. Los elementos principales de este grupo son los WaitSet, los Condition y los StatusCondition.

2.4.3 Soporte a las comunicaciones

El soporte a la calida de servicio en DDS es muy sencillo, ya que se compone de una única clase base, de la que derivan cada una de las clases que implementan las calidades de servicio.

Las calidades de servicio pueden asociarse a diferentes objetos “Entity”, tales como Topic, DataWriter, DataReader, Publisher y DomainParticipant. Puede ocurrir que algunos valores de QosPolicy (clase base de objetos de calidad de servicio) no sean consistente con otras. Estos casos se deben tener en cuenta.

3 Políticas de calidad de servicio

En el modelo DCPS de DDS, las políticas de calidades de servicio se implementan como una lista de calidades de servicio que debe cumplir el componente al que se asocie. Como se ve en la figura , las calidades de servicio son objetos derivados de una clase base “QoSPolicy” asociada a la clase base “Entity” de DCPS, lo que supone que todos los componentes de la comunicación (excepto “Listener” y los relacionados con las condiciones) pueden tener una serie de calidades de servicio asociadas.

El hecho de que todos los componentes de comunicaciones puedan tener asociada una política de calidad de servicio hace que se puedan especificar restricciones en todos los niveles de la comunicación, desde el acceso a un “Topic” específico por parte de un “DaaWriter”o “DataReader” hasta el conjunto de todos los componentes de comunicación de un nodo. Esta jerarquización hace muy flexible los puntos de la comunicación donde se desee aplicar una política de calidad de servicio.

Se debe tener en cuenta que las calidades de servicio que se solicitan por parte de un “Subscriber”, deben ser cumplidas por un “Publisher". Para la negociación se sigue el patrón “Subscriber” solicita y “Publisher” ofrece. Bajo este tipo de patrón, desde el punto de vista del subscriptor, se puede solicitar un valor, y el publicador también podrá ofrecer un valor. El servicio será quien determinará si los requerimientos son compatibles. Los servicios ya negociados pueden ser cambiados aun ya establecida la conexión, pero solo algunos de ellos.

De cada política de calidad de servicio, se describirán los elementos del modelo DDS a los que concierne, la característica solicitado/ofrecido que determinará si se debe dar alguna condición entre la calidad solicitada y la ofrecida, y finalmente se debe determinar si alguna de las características de las políticas de calidad de servicio pueden ser cambiadas cuando la comunicación ya se ha establecido. Es posible organizar las políticas de calidad de servicio en grupos, atendiendo a la funcionalidad que ofrecen o el ámbito de la comunicación que ofrecen. A continuación se describen las políticas de calidad de servicio del DCPS del modelo DDS.

3.1 Gestión de meta datos

3.1.1 UserDataQoSPolicy, TopicDataQoSPolicy, GroupDataQoSPolicy

Estas tres políticas de calidad de servicio, son muy similares, la única diferencia es el elemento del modelo DDS sobre el que se aplican. Todas ellas se emplean para enviar información del usuario, del “Topic”, o del “Group” al resto del sistema. Se puede emplear como cadena de bytes con contenido libre. En lo que respecta a la relación entre los valores solicitados y ofrecidos, en este caso, no es necesario que coincidan, ni existe ninguna relación entre ellos. Los diagramas de clases se pueden ver en la figura 4.

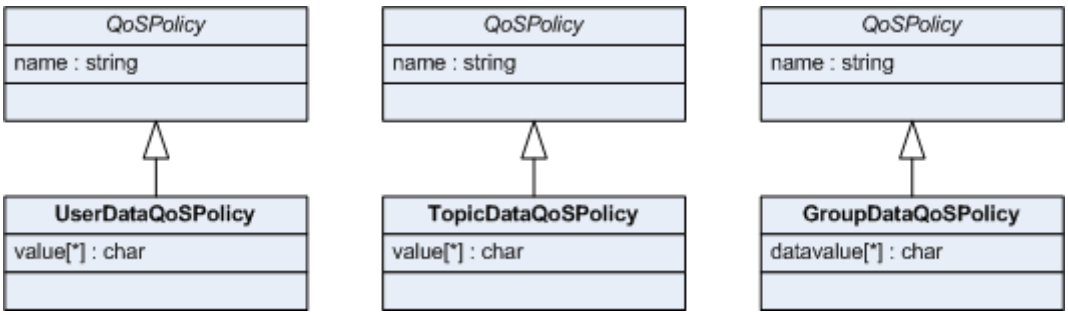

Figura 4. UML de UserDataQoSPolicy, TopicDataQoSPolicy y GroupDataQoSPolicy.

El valor por defecto es una cadena vacía. Esta característica de la calidad de servicio sólo es informativa al resto del sistema acerca del usuario, del "Topic" o del "Group". El propósito de esta calidad de servicio es permitir a la aplicación adjuntar o asociar información adicional a una "Entity" que se haya creado. Cuando una aplicación descubra cualquiera de los objetos heredados de "Entity", puede acceder a esta calidad de servicio para obtener información que pueda ser interpretada. Un ejemplo del uso de esta calidad de servicio es asociar las credenciales de seguridad a un objeto, que permita a una aplicación poder identificar correctamente al componente DCPS de que se trate. Además de la seguridad, cualquier uso está permitido para esta calidad de servicio.

3.2 Aspectos temporales

3.2.1 DurabilityQoSPolicy

El desacoplamiento entre el "DataWriter" y el "DataReader" que implica el paradigma "Publish-Subscribe", debe permitir a una aplicación poder escribir datos sin que se encuentren lectores en la red que los consuman. Además un "DataReader" que se una a la red en un momento concreto puede estar interesado en los valores más recientes, pero también en algunos valores pasados o históricos.

Para cubrir la posibilidad de que un componente de la comunicación pueda especificar el tipo de durabilidad, o ámbito de duración temporal, que tiene la información se tiene esta política de calidad de servicio que expresa el tiempo que debe sobrevivir un dato. Esta política de calidad de servicio afecta a los "Topic", los "DataReader" y los "DataWriter". Tienen restricciones acerca de los valores solicitados y ofrecidos que se detallará más adelante. Finalmente los valores no pueden cambiar una vez se ha establecido la comunicación. El diagrama de clase se puede ver en la figura 5.

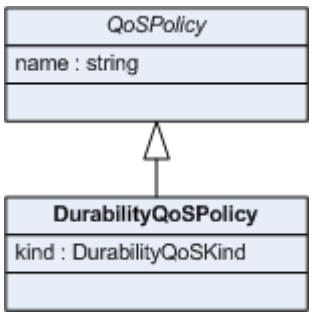

Figura 5. UML de DurabilityQoSPolicy.

La propiedad "kind" puede tomar diversos valores:

- VOLATILE. Valor por defecto. Significa que el Publisher sólo proporciona la información del dato a los Subscribers que existan en ese momento.
- TRANSIENT_LOCAL. TRANSIENT. Estos dos modos consisten en mantener algunas muestras para poder ser transmitidas a posteriores conexiones de un DataReader. Las características de estas muestras dependen de algunos otros parámetros de calidad de servicio. En el modo local (TRANSIENT_LOCAL), esta información permanece sólo en el lado del DataWriter, con las características de supervivencia que tenga el DataWriter. En el caso no local (TRANSIENT), permanece en memoria y no en soporte local.
- PERSISTENT. El dato se mantiene almacenado, por lo que sobrevive a toda la sesión.

Implícitamente hay un orden de precedencia en el valor de la propiedad "kind", desde la menor precedencia "VOLATILE" hasta la máxima "PERSISTEN", este orden puede verse en la ecuación 1, e implica que, por ejemplo, configurar a "PERSISTENT" ésta calidad de servicio, implícitamente incluye a las anteriores.

Ecuación 1. Orden de precedencia de la propiedad "kind" en la política de QoS Durability.

$$\text{VOLATILE} < \text{TRANSIENT\_LOCAL} < \text{TRANSIENT} < \text{PERSISTENT} \quad (1)$$

Se debe tener en cuenta que en el modo TRANSIENT, los datos permanecen en memoria, pero en el PERSISTENT permanecen por encima de la sesión de comunicación, lo que implica que ciertos mensajes pueden sobrevivir por encima de las sesiones de conexiones de comunicaciones.

3.2.2 DurabilityServiceQoSPolicy

La política "DurailityQoSPolycy" puede tener algunos efectos colaterales con otras políticas de calidad de servicio, esto hace que sea necesario definir algunos parámetros y las relaciones de la política DurabilityQos con las afectadas. Ésta política define la durabilidad del servicio, en este caso se aplica a los modos TRANSIENT o PERSISTENT de la política de calidad de servicio "DurabilityQoSPolicy". El diagrama de clases es el que se muestra en la figura 6.

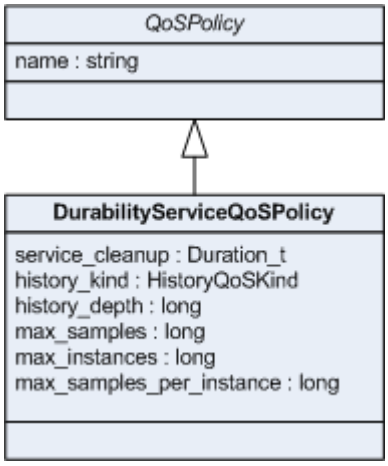

Figura 6. UML de DurabilityServiceQoSPolicy.

Los valores y el significado de sus propiedades son los siguientes

- service_cleanup_delay. Controla cuando el servicio debe eliminar toda la información de una instancia de un dato, por lo que contiene la duración del mismo.
- Dos propiedades relacionadas con la política de calidad de servicio "HistoryQosPolicy".
    - history_kind: es una propiedad del tipo "HistoryQosPolicy", o lo que es lo mismo es una "política de calida de servicio". Controla la calidad de servicio del historial de mantenimiento. El valor por defecto es KEEP_LAST.
    - history_depth. Es un entero que especifica la profundidad de los datos que se almacenan en la política anterior. El valor por defecto es 1
- Tres valores relacionados con la política de la calidad de servicio "ResourceLimitsQoSPolicy". Estos valores los debe implementar el DataReader que almacena el dato
    - max_samples.
    - max_instances.
    - max_samples_per_instance.

Esta política sirve de puente en las configuraciones de las políticas afectadas por los valores definidos en la política DurabilityQoSPolicy.

3.2.3 DeadlineQoSPolicy

Esta política de calidad de servicio, define el límite o plazo, que puede tener un elemento, afecta a un "Topic", a los "DataReader" y a los "DataWriter". Los valores ofrecidos y solicitados deben cumplir ciertos requisitos, que se detallarán más adelante. Puede cambiar a lo largo de la sesión de comunicación, siempre que los requisitos entre los valores ofrecidos y solicitados se cumplan. El diagrama de clase se puede observar en la figura 7.

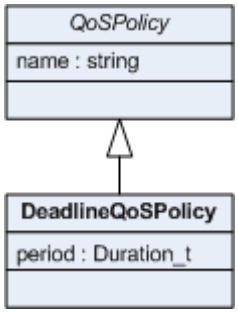

Figura 7. UML de DeadlineQoSPolicy.

Como propiedades, tiene tan solo la característica de duración o "period". El "DataReader", espera una nueva muestra actualizada al menos una vez cada plazo indicado en el "deadline". Debido a la importancia de ésta política de calidad de servicio, se muestra on detalle el funcionamiento en la figura 8.

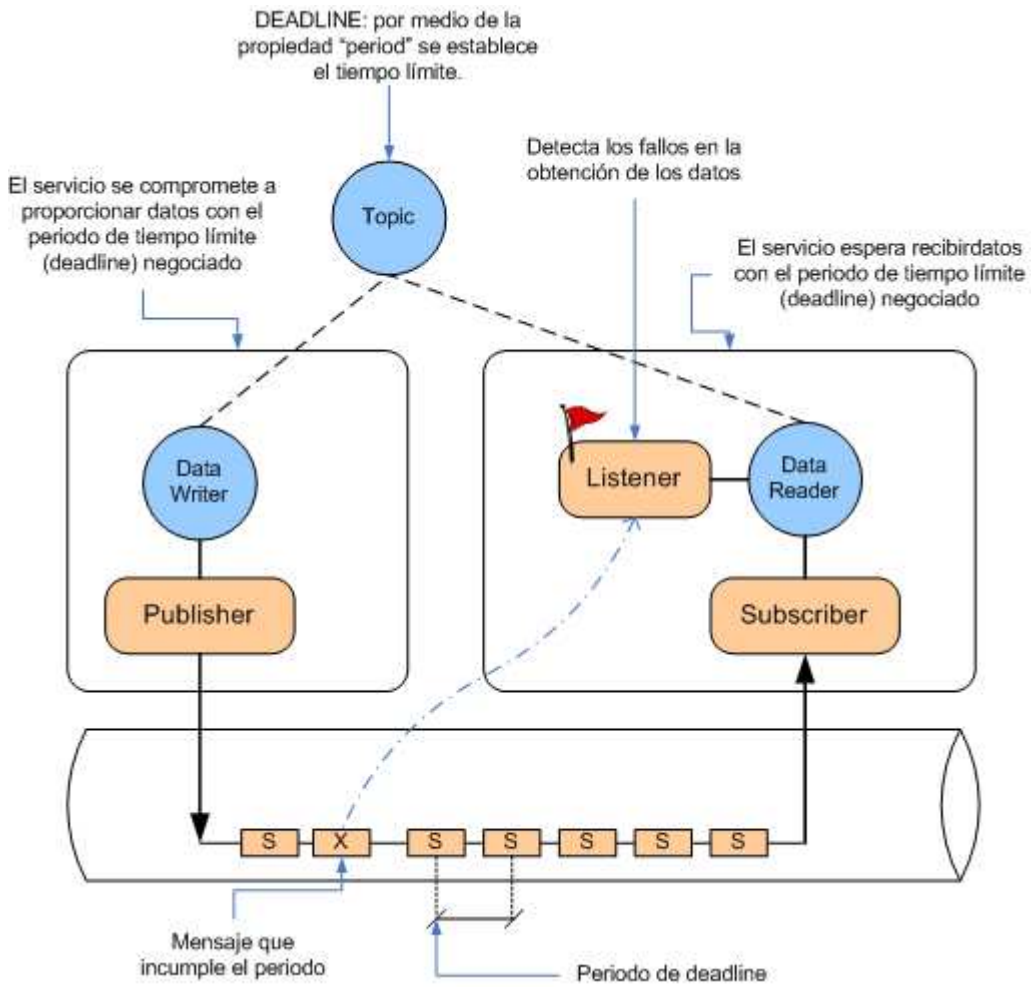

Figura 8. Comportamiento de Deadline dentro en los componentes DCPS.

Esta política de calidad de servicio es muy útil cuando un "Topic" espera tener cada instancia actualizada periódicamente. Desde el punto de vista del "Publisher", esta característica establece un contrato que la aplicación debe conocer. En el lado del "Subscriber", la característica establece un mínimo requerimiento que se espera que los "Publishers" remotos puedan cumplir.

Se debe tener en cuenta que cuando se ajustan los parámetros de calidad de servicio de un DataWriter y un DataReader, se evalúan cuáles son las características que pueden entrar en conflictos, y resolver las posibles incompatibilidades. Asumiendo que el "Reader" y el "Writer" finales de la comunicación tienen características compatibles, el servicio debe supervisar si esto se cumple en todo momento, para evitar que algún cambio altere estos cumplimientos. El valor que se ofrezca, deberá ser compatible con el valor solicitado, sí y sólo sí se cumple la desigualdad de la ecuación 2 se evalúa a TRUE.

Ecuación 2. Relaciones entre los periodos de deadline ofrecidos y solicitados.

"periodo ofrecido de deadline <= periodo solicitado de deadline" (2)

El valor de la calidad de servicio "DEADLINE" debe ser coherente con el que se tenga en la calidad de servicio "TIME_BASED_FILTER", la coherencia está cumplir la desigualdad que se muestra en la ecuación 3.

Ecuación 3. Coherencia en los periodos de deadline y la separación temporal entre mensajes.

$$\text{deadline period} >= \text{minimun\_separation} \quad (3)$$

Deadline es una política de calidad de servicio básica sobre la que se asienta el soporte a tiempo real que proporciona DDS.

3.2.4 LatencyBudgetQoSPolicy

Esta política de calidad de servicio, trata de los tiempos que la aplicación aprecia o solicita para el envío de mensajes. Los elementos a los que afecta son los "Topic", los "DataReader" y los "DataWriter". Los valores ofrecidos y solicitados deben cumplir ciertos requisitos que se detallarán más adelante. El diagrama de clase es el de la figura 9.

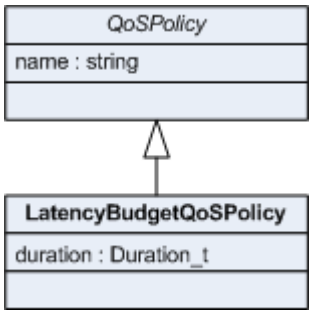

Figura 9. UML de LatencyBudgetQoSPolicy.

Esta política de calida de servicio, tiene tan solo una propiedad: "duration". Ésta propiedad especifica el máximo aceptable de retraso desde el momento en que el dato es escrito hasta que el dato se escribe en la caché del receptor y el receptor es avisado. El valor por defecto es cero, lo que indica que el retraso debe ser minimizado. Gráficamente se puede ver cómo actúa en la figura 10.

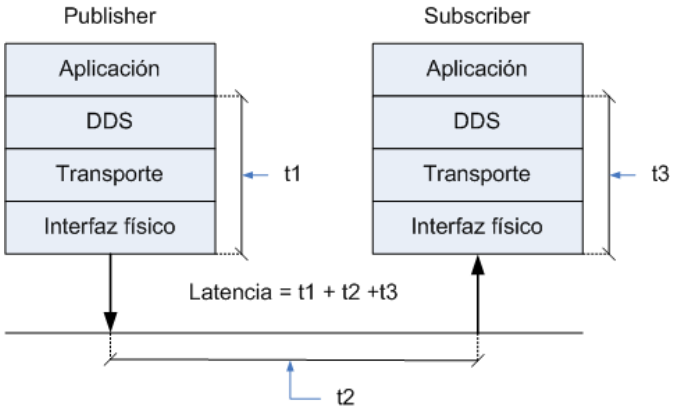

Figura 10. Comportamiento de la LatencyBudgetQoSPolicy.

Esta política de calidad de servicio proporciona a la aplicación una manera de indicarle al "middleware" la urgencia de las comunicaciones de los datos. Con estos parámetros se puede ajustar las operaciones de comunicaciones. Esta política se considera aconsejable, aunque no se especifica la forma en la que se debe implementar. El valor ofrecido se considera compatible con el solicitado, sí y sólo sí se cumple la desigualdad de la ecuación 4.

Ecuación 4. Relación entre las dureaciones de latencia ofrecidas y solicitadas en la política de calidad de servicio LatencyBudgetQoSPolicy.

$$\text{“duration” ofrecida} <= \text{“duration” solicitada} \quad (4)$$

Al igual que la política “Deadline”, “LatencyBudget” es una política de calidad muy vinculada a un parámetro de calidad de servicio, en este caso la latencia.

3.2.5 LivelinessQoSPolicy

Esta política de calidad de servicio controla el mecanismo y los parámetros necesarios por medio del cual el servicio se asegura que las entidades de la red que lo requieran (las que hayan negociado la política de calidad de servicio) se encuentran activas. Afecta al “Topic”, “DataReader” y “DataWriter”. En lo que respecta a los valores ofrecidos y solicitados, deben cumplir algunos requerimientos que más adelante se tratarán. Los valores no pueden variar a lo largo de la comunicación. El diagrama de clases es el de la figura 11.

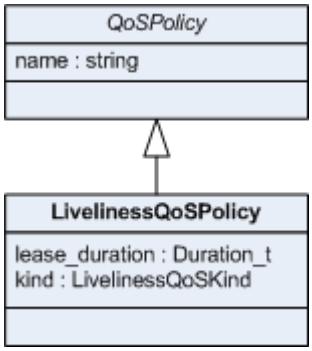

Figura 11. UML de LivelinessQoSPolicy.

Las características que pueden tener son las siguientes.

- kind. Determina el mecanismo y parámetros usados por la aplicación para determinar qué entidad (Entity) está activa. Se emplea para mantener la propiedad de una instancia en combinación con las características de la calidad de servicio OWNERSHIP. Puede tomar los siguientes valores. Los modos manuales son los que automáticamente es la aplicación la que toma la iniciativa de señalizar el valor de “liveness”.
    - AUTOMATIC. La infraestructura automáticamente señalizará el “liveliness” para los “DataWriters” a los requerimientos de “duration”.
    - MANUAL_BY_PARTICIPANT. El servicio asumirá hasta cuando al menos una entidad dentro de un DomainParticipant ha asumido su propio “liveliness”.
    - MANUAL_BY_TOPIC. El servicio sólo asume el “liveliness” del DataWriter si la aplicación ha impuesto liveliness en el DataWriter ella misma.
- duration. Duración propiamente dicha a la que se ha hecho referencia en los casos anteriores. El valor por defecto es infinito.

Esta calidad de servicio puede afectar a la calidad OWNERSHIP (Propiedad), por lo que los efectos colaterales deberán tenerse en cuenta. El comportamiento de esta

política de calidad de servicio, puede representarse gráficamente como se ve en la figura 12.

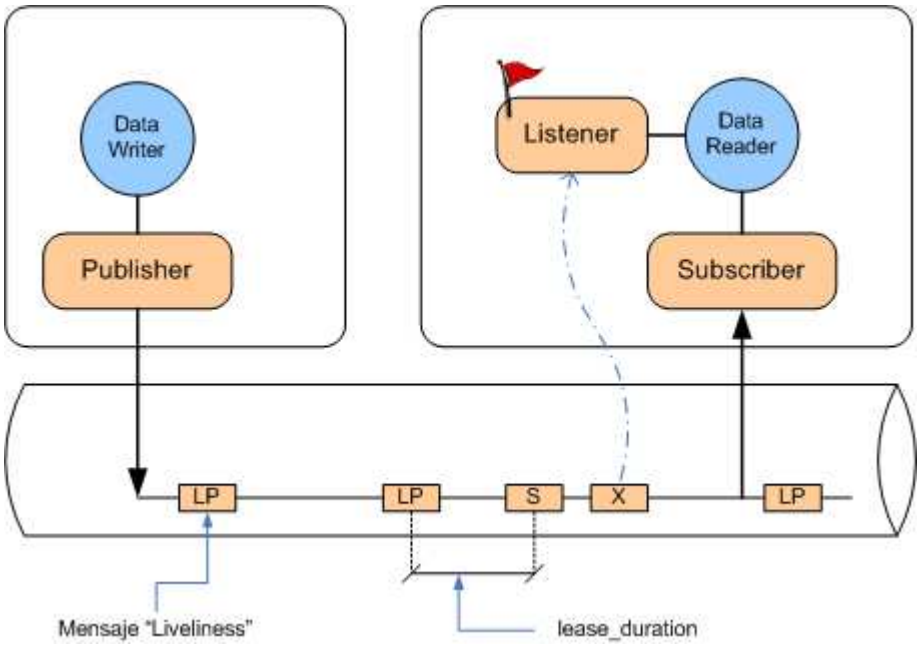

Figura 12. Comportamiento de los componentes de LivelinessQoSPolicy.

Esta política de calidad de servicio tiene varias propiedades que deben soportar los objetos que se estén comunicando, que son actualizadas periódicamente o esporádicamente en el caso de cambios. Esto permite la personalización o adaptación a las aplicaciones de los requerimientos en términos de los tipos de fallos que se detecten por medio del mecanismo de ésta calidad de servicio.

Si la propiedad "kind" tiene el valor AUTOMATIC, es adecuado para aplicaciones que sólo necesiten detectar fallos a nivel de proceso, pero no para errores lógicos dentro de un proceso. En este modo, el servicio toma la responsabilidad de renovar los contratos para que los componentes que participen en la comunicación tengan la certeza de que están todos activos. Este modo es el que produce una sobrecarga mínima.

El modo MANUAL (tanto el modo MANUAL_BY_PARTICIPANT, como el MANUAL_BY_TOPIC) requiere que la aplicación, en el lado del "Publisher", se responsabilice a afirmar periódicamente la vivacidad antes de que el contrato realizado expire a la entidad correspondiente (Participant, o Topic). Esta acción puede realizarse explícitamente por medio de la operación "assert_liveliness" o implícitamente por medio de una operación de escritura. Los dos modos, controlan la "granularidad" con la que se debe reafirmar la situación de vivo el elemento. El modo MANUAL_BY_PARTICIPANT, consiste en que con que sólo una "Entity" dentro del "Publisher" confirme su "liveliness", se deduce que el resto de "Entities" dentro del mismo DomainParticipant están todavía vivos. El modo MANUAL_BY_TOPIC requiere que al menos una instancia dentro del DataWriter sea la que afirme su vivacidad. El valor ofrecido, se considera compatible con el valor solicitado, sí y sólo sí las siguientes condiciones se dan: "kind ofrecido >= kind solicitado" Para esta desigualdad se considera el orden para la comparación el mostrado en la ecuación 5.

Ecuación 5. Orden de comparación de la propiedad "kind" en la política de calidad de servicio Liveliness.

$$\text{AUTOMATIC} < \text{MANUAL\_BY\_PARTICIPANT} < \text{MANUAL\_BY\_TOPIC} \qquad (5)$$

Los cambios deben ser detectados por el servicio con una granularidad temporal mayor o igual que la "lease_duration". Esto asegura que el valor de "LivelinessChangedStatus" se actualiza al menos una vez durante cada "lease_duration" y los "Listeners" y "WaitSets" sean notificados dentro de "lease_duration".

3.2.6 TimeBasedFilterQoSPolicy

Esta política de calidad de servicio determina un filtro que permite a un "DataReader", especificar que está interesado sólo (potencialmente) en un subconjunto de valores de los datos. Esta política de calidad se refiere únicamente a los "DataReader", en lo que a calidad ofrecida y solicitada, no es necesario que sea la misma en ambos lados de la comunicación. Las propiedades de esta política de calidad de servicio pueden variar a lo largo de la comunicación. El diagrama de clase es el mostrado en la figura 13.

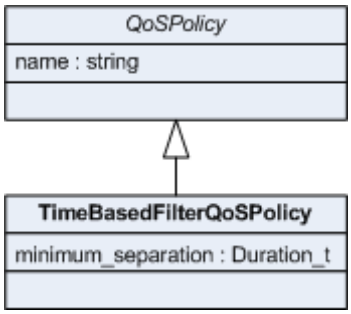

Figura 13. UML de TimeBasedFilterQoSPolicy.

Esta política de calidad de servicio, tiene una única propiedad llamada "miminum_separation". El filtro establece que el DataReader no quiere recibir más de un valor cada "minimun_separation", a pesar de lo rápido que ocurran los cambios. Es inconsistente para un DataReader tener un "minimun_separation" mayor que su periodo DEADLINE. Por defecto, el valor de "minimun_separation" vale 0, lo que indica que el DataReader está interesado en todos los valores.

En la figura 14, se muestra una descripción gráfica de ésta política de calidad de servicio. Esta política permite a un DataReader indicar que no es necesario observar todas las muestras de cada instancia que sean publicadas en un Topic. Para ello se puede solicitar un dato cada "minimun_separation". Esta calidad de servicio, se aplica separadamente a cada instancia. La utilidad de esta calidad de servicio es muy elevada, ya que permite a un DataReader, desacoplarse de un DataWriter en el caso de que las redes puedan variar su tiempo de transmisión.

El hecho de solicitar esta calidad de servicio (poniendo un valor mayor de cero en la propiedad "minimun_separation" no la hace incompatible (es decir, es compatible) con las calidades de servicio HISTORY y RELIABILITY. TIME_BASED_FILTER especifica las muestras en las que está interesado el DataReader. Las calidades HISTORY y TIME_BASED_FILTER afectan al comportamiento del "middleware", con respecto a las muestras se haya determinado que son interesantes para el DataReader, es decir, estas dos calidades de servicio se aplican después de que se haya aplicado el TIME_BASED_FILTER.

La propiedad “minimun_separation” debe ser compatible con la propiedad “period” de la calidad de servicio DEADLINE. Para que se considere compatible, se debe verificar la desigualdad de la ecuación 6.

Ecuación 6. Condición de compatibilidad entre las políticas de calidad de servicio TimeBasedFilter y Deadline.

$$period >= minimun\_separation \quad (6)$$

Las comprobaciones de la compatibilidad de las dos calidades de servicio deben hacerse en el momento en que sean instanciadas.

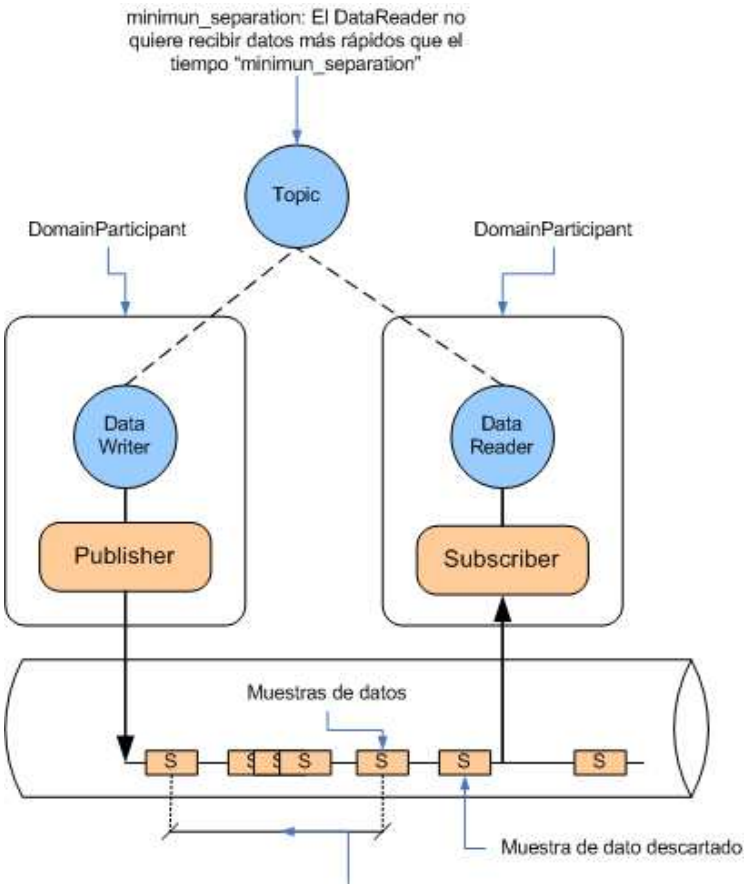

Figura 14. Comportamiento de los componentes de TimeBasedFilterQoSPolicy.

3.2.7 LifespanQoSPolicy

Ésta política de calidad de servicio, especifica la máxima duración de validez del dato escrito por el DataWriter. El valor por defecto es infinito. Esta política afecta al “DataWriter” y al “Topic”. A la hora de negociarse, se debe negociar en el “Publisher” o el “Subscriber”, pero no en ambos. El valor puede variar a lo largo de la sesión de comunicaciones. El diagrama de clases se muestra en la figura 15.

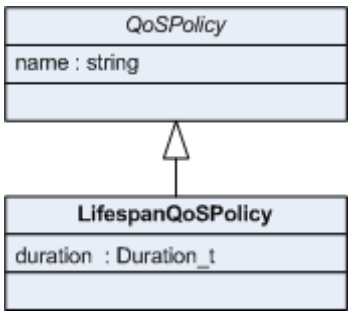

Figura 15. UML de LifespanQoSPolicy.

El propósito de esta calidad de servicio es evitar entregar datos pasados a la aplicación. Cada muestra de datos escrita por el "DataWriter" tiene asociado un tiempo de expiración, "expiration_time" más allá del cual el dato no se debe proporcionar a ninguna aplicación. Una vez el dato expira, el dato deberá ser eliminado de la caché del "DataReader". El tiempo de expiración de cada muestra, se computa añadiendo la duración especificada en el campo "duration", al "timestamp" que se haya calculado por parte del servicio o bien que se suministre por la aplicación. Esta calidad de servicio, presupone que los relojes están suficientemente sincronizados. En el caso de no haber sincronización, o no poder asegurarse, se permite que el "DataReader" use el "timestamp" de la recepción, en lugar del "timestamp" del envío.

3.3 Gestión del flujo de datos

3.3.1 PresentationQoSPolicy

Esta política de calidad de servicio, especifica cómo las muestras que representan cambios en las instancias de los datos se presentan a la aplicación que se ha suscrito a esos datos. Esta política afecta a cómo la aplicación puede especificar y recibir coherentemente los cambios y verlos relativamente ordenados. El diagrama de clases es el mostrado en la figura 16.

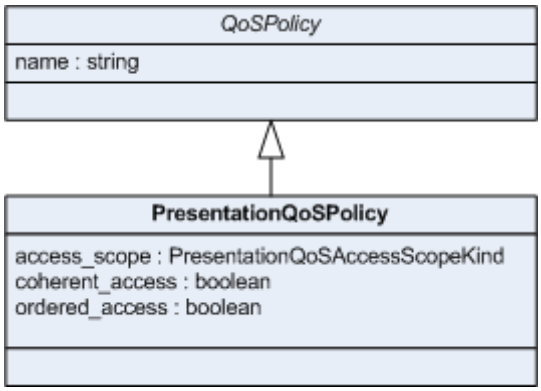

Figura 16. UML de PresentationQoSPolicy.

Las propiedades que tiene son las siguientes

- access_scope. Determina el alcance de expansión de la coherencia de los cambios hacia las entidades. Puede tomar los siguientes valores
    - INSTANCE. En este caso el orden y la coherencia de los datos se aplica únicamente a cada instancia separadamente.

    - TOPIC. El ámbito son todas las instancias dentro del mismo DataWriter o DataReader, pero no en distintos.
    - GROUP. El ámbito es todo el grupo de DataWriter o DataReader, dentro del mismo Publisher o Subscriber.

- coherent_access. Valor booleano. Especifica si se soporta el acceso coherente. Es decir, la capacidad para agrupar un conjunto de cambios como una unidad en la publicación y por tanto en la recepción de la información.
- ordered_access. Valor booleano. Especifica si se soporta el acceso ordenado, es decir la capacidad de ver el suscriptor los cambios en el publicador en el mismo orden en el que se producen.

Esta política de calidad de servicio, controla el alcance que pueden tener los cambios, controlados por el "Subscriber", esto significa que el "Subscriber" puede acceder en un orden coherente. El valor ofrecido es considerado compatible con el valor solicitado, sí y sólo sí, se dan las condiciones de la ecuación 7, en la que el orden de prioridad en los valores de "access_scope" es el mostrado en la ecuación 8.

Ecuación 7. Condiciones para la coherencia en la política de calidad de servicio Presentation.

$$\text{“access\_scope” ofrecido} >= \text{“access\_scope” solicitado} \quad (7)$$

Ecuación 8. Orden de prioridad de la propiedad "access_scope" empleado en la ecuación 7.

$$\text{INSTANCE} < \text{TOPIC} < \text{GOUP} \quad (8)$$

En cuanto al valor de la coherencia en el acceso, contenido en la propiedad "coherent_access", deberá cumplir la condición presentada en las ecuaciones 9 y 10.

Ecuación 9. Condición de coherencia de "coherente_access" en la política de calidad de servicio Presentation.

$$\text{“coherent\_access” solicitado} = \text{FALSE} \quad (9)$$

Ecuación 10. Relacion entre las propiedades "coherent_access" solicitados y ofrecidos en la política de calidad de servicio Presentation.

$$\text{“coherent\_access” solicitado} = \text{“coherent\_access” ofrecido} = \text{TRUE} \quad (10)$$

Finalmente, en cuanto al valor del acceso ordenado, contenido en la propiedad "ordered_access", deberá cumplir las condiciones de las ecuaciones 11 y 12.

Ecuación 11. Condición de coherencia de "ordered_access" en la política de calidad de servicio Presentation.

$$\text{“ordered\_access” solicitado} = \text{FALSE} \quad (11)$$

Ecuación 12. Relacion entre las propiedades "ordered_access" solicitados y ofrecidos en la política de calidad de servicio Presentation

$$\text{“ordered\_access” solicitado} = \text{“ordered\_access” ofrecido} = \text{TRUE} \quad (12)$$

3.3.2 ReliabilityQoSPolicy

Esta calidad de servicio, indica el nivel de fiabilidad ofrecido o solicitado por el servicio. Tiene el diagrama de clase mostrado en la figura 17.

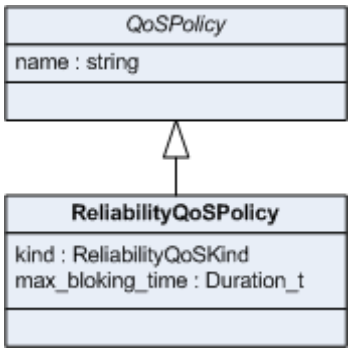

Figura 17. UML de ReliabilityQoSPolicy.

Un ejemplo gráfico de cómo funciona la política de calidad de servicio puede contemplarse en la figura 18. Las propiedades de la clase son las siguientes.

- kind. Tipo, que puede tomar los siguientes valores.
    - REALIABLE. Especifica que el Servicio intentará entregar todas las muestras de su historial. Se pueden obtener muestras perdidas. Este es el valor por defecto de los DataWriters.
    - BEST_EFFORT. Indica que es aceptable no reintentar la propagación de cualquier muestra. Este es el valor por defecto de los DataReaders y de los Topics.
- max_blocking_time. Duración. Indica el máximo tiempo que la operación DataWriter::write se permite que se bloquee

Las dos categorías se ordenan jerárquicamente de la forma especificada en la ecuación 13.

Ecuación 13. Orden jerárquico de las categorías de la política de calidad de servicio Reliability.

$$\text{BEST\_EFFORT} < \text{RELIABLE} \tag{13}$$

Esto quiere decir que un DataWriter que ofrezca la calidad de servicio RELIABLE, implícitamente está ofreciendo la calidad de servicio BEST_EFFORT. La configuración de esta calidad de servicio, tiene una dependencia con la configuración de la política de calidad de servicio RESOURCE_LIMITS. En el caso de que la propiedad "kind" de la calidad de servicio sea RELIABLE, la operación Write del DataWriter, puede bloquearse caso de que se pueda perder el dato, o en caso de que se excedan uno de los límites especificados en RESOURCE_LIMITS. En estos casos la propiedad "max_blocking_time" establecerá el máximo tiempo que la operación Write podrá estar bloqueada.

Si la propiedad "kind" contiene el valor "RELIABLE", las muestras originales de los datos que se escriban en el DataWriter, no podrán estar disponibles en el DataReader si hay muestras previas que no han sido recibidas. Es decir el servicio debe ser el responsable de reparar el error y transmitir el historial correcto de muestras.

Si la propiedad "kind" es BEST_EFFORT, el servicio no retransmite las muestras que se hayan perdido. Sin embargo, para las muestras originadas en cualquier DataWriter, el servicio deberá asegurarse que están almacenadas en el historial del DataReader en el mismo orden que se originaron en el DataWriter. En otras palabras, el DataReader puede perder algunas muestras de datos, pero nunca observar que el valor de un objeto de datos, cambia desde un valor reciente a un valor anterior.

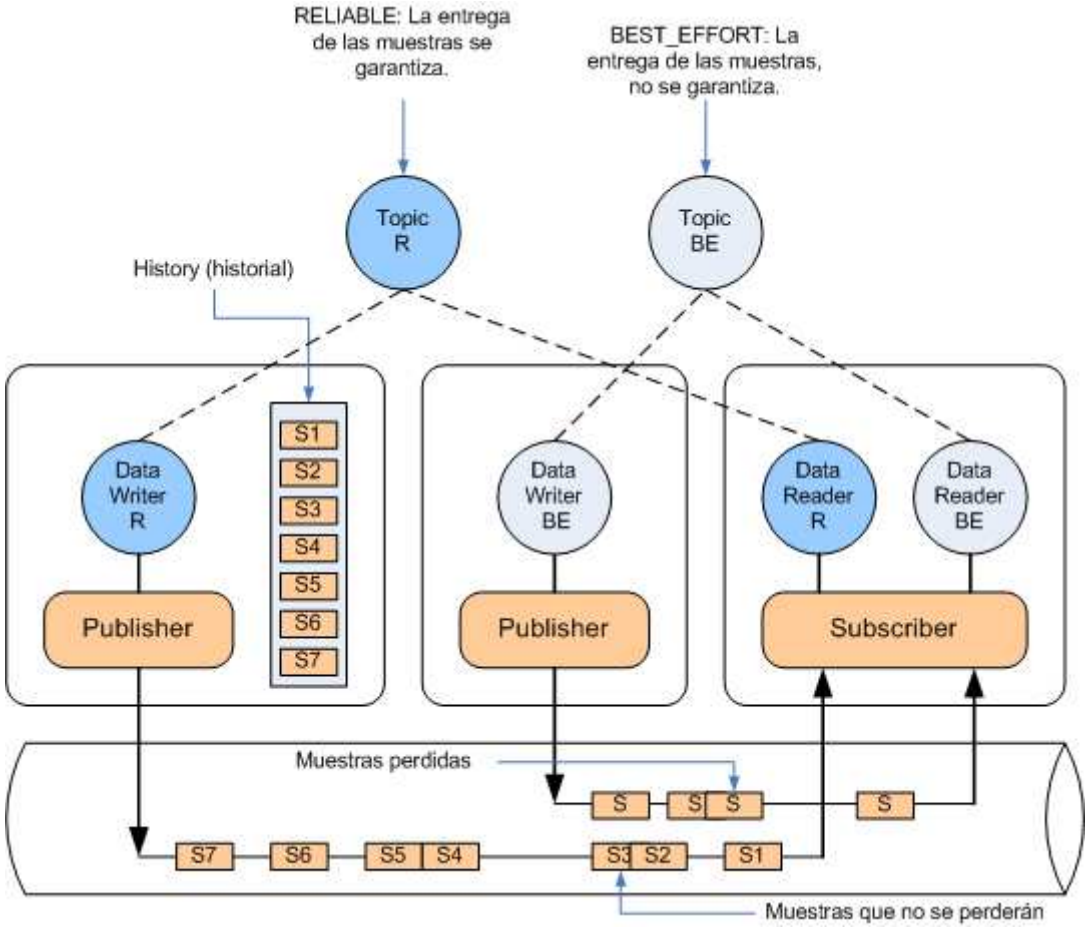

Figura 18. Comportamiento de los componentes de ReliabilityQoSPolicy.

El valor ofrecido se considera que es compatible con el valor solicitado, sí y solo sí se cumple (resultado TRUE) la desigualdad de la ecuación 14. Considerando el orden de categorías anteriormente citado.

Ecuación 14. Relación entre los valores ofrecidos y solicitados de la propiedad "kind" de la política de calidad de servicio Reliability.

$$\text{"kind ofrecido"} >= \text{"kind solicitado"} \qquad (14)$$

3.3.3 TransportPriorityQoSPolicy

Esta política de servicio es una propuesta a la infraestructura cómo debe poner la prioridad del transporte usado para enviar los datos. El diagrama de clase es el mostrado en la figura 19.

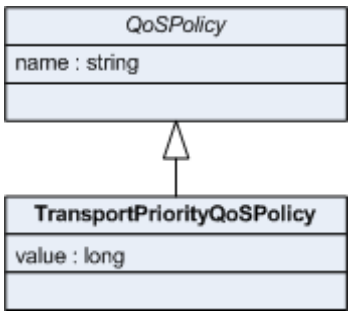

Figura 19. UML de TransportPriorityQoSPolicy.

Esta política de calidad de servicio es la que permite a la aplicación dar diferentes prioridades a los mensajes que se envíen. Por convenio se considera que la prioridad está relacionada de forma directa con el valor, de manera que los valores elevados del campo "value" suponen prioridades más altas. En el modelo DDS se deja a la aplicación el control de esta política de calidad de servicio.

3.3.4 DestinationOrderQoSPolicy

Esta política de calidad de servicio controla el criterio usado para determinar el orden lógico entre cambios realizados por las entidades Publisher a la misma instancia de datos. El diagrama de clase es el que se muestra en la figura 20.

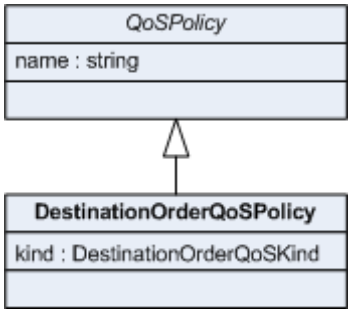

Figura 20. UML de DestinationOrderQoSPolicy.

La política de calidad de servicio tiene tan solo una propiedad: "kind", que puede tener dos posibles valores.

- BY_RECEPTION_TIMESTAMP. Indica que los datos se ordenarán basándose en el orden temporal de llegada al Subscriber. Debido a que cada Subscriber puede recibir los datos en diferentes momentos, no se garantiza que los cambios se envíen en el mismo orden. Consecuentemente es posible que para cada Subscriber, finalice con un valor final diferente.
- BY_SOURCE_TIMESTAMP. Indica que los datos se ordenan basándose en el timestamp puesto por la fuente (por el servicio o por la aplicación). En ningún caso esto garantiza una consistencia de los valores finales de los datos en todos los subscriptores.

Esta política de calidad de servicio determina cómo cada Subscriber resuelve el valor final de la instancia de datos que es escrita por múltiples DataWriter que pueden ser asociadas con diferentes objetos Publisher. Un ejemplo gráfico de cómo es su funcionamiento puede verse en la figura 21.

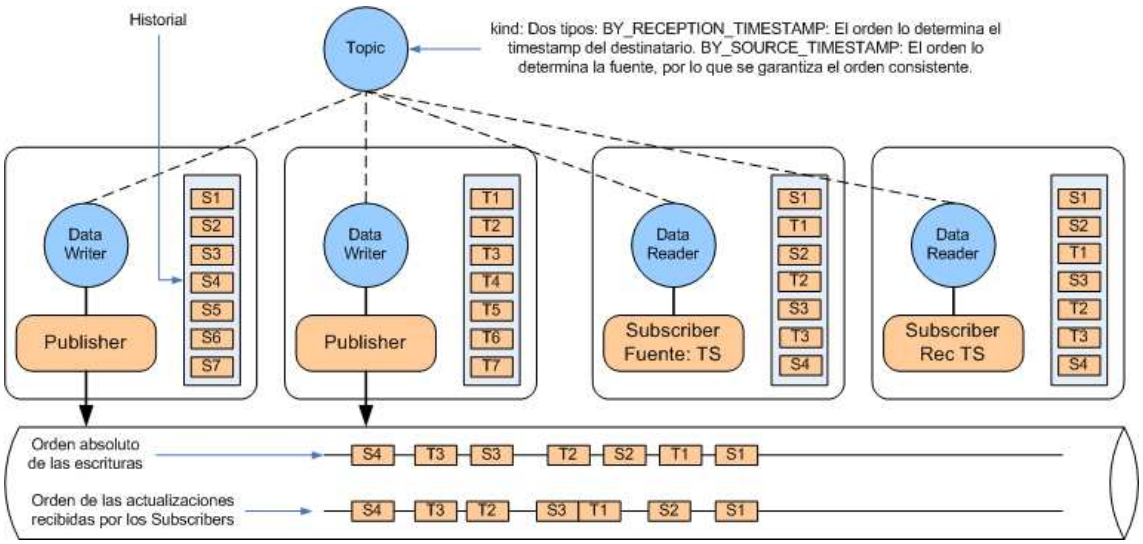

Figura 21. Comportamiento de los componentes de DestinationOrderQoSPolicy.

El tipo de calidad BY_RECEPTION_TIMESTAMP indica que la calidad OWNERSHIP, permite que el último valor recibido por la instancia deba ser uno cuyo valor se mantenga. El tipo de calidad BY_SOURCE_TIMESTAMP, indica que la calidad OWNERSHIP, permite incluir un "timestamp", esto debe permitir que ante datos similares o duplicados, todos los Subscribers terminen con el mismo valor. El mecanismo para el "timestamp" debe ser dependiente del "middleware". El valor ofrecido, se considera compatible con el valor solicitado, sí y sólo sí, la desigualdad definida de la ecuación 15 se evalúa como cierta.

Ecuación 15. Compatibilidad de valores de la propiedad "kind" de la política de calidad de servicio DestinationOrderQoSPolicy.

$$\text{"kind ofrecido} >= \text{kind solicitado"} \tag{15}$$

Para resolver la desigualdad, se considera el orden de tipos siguiente.

Ecuación 16. Orden de tipos que se aplica en la fórmula 15.

$$\text{BY\_RECEPTION\_TIMESTAMP} < \text{BY\_SOURCE\_TIMESTAMP} \tag{16}$$

3.3.5 HistoryQoSPolicy

Esta política de calidad de servicio, especifica el comportamiento del servicio en el caso de que el valor de una muestra cambia (una o más veces) antes de que pueda ser comunicado satisfactoriamente a uno o más Subscribers. Esta política de calidad de servicio controla en qué caso el servicio debe entregar el dato más reciente o los datos intermedios. En el lado del Publisher esta política de calidad de servicio controla las muestras que deben ser mantenidas por el DataWriter favorables a las entidades existentes DataReader. En el lado del Subscriber, esta política de calidad de servicio controla las muestras que deben ser mantenidas hasta que la aplicación las recoge desde el servicio. El diagrama de clases se observa en la figura 22.

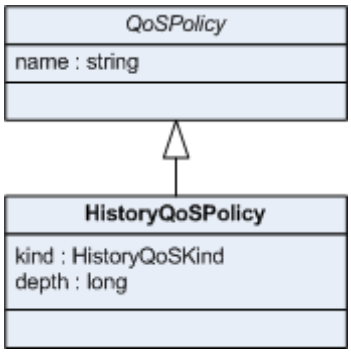

Figura 22. UML de HistoryQoSPolicy.

Esta política de calidad de servicio tiene dos propiedades, que son las siguientes.

- kind. Tipo de Historia que se mantiene en las entidades correspondientes.
    - KEEP_LAST. En el lado del Publicador, el servicio sólo se encargará de mantener un número determinado de muestras de cada dato, indicado por el parámetro "depth" gestionadas por el DataWriter. En el lado del Suscriptor, el DataReader deberá mantener las últimas "depth" muestras de cada instancia (identificadas por sus "key"). El valor por defecto del tipo (kind) es éste, con un "depth" de 1. Estas muestras permanecen hasta que se hace una lectura por medio de "take".
    - KEEP_ALL. En el Publicador el servicio deberá mantener todas las muestras (representando cada valor escrito) de cada instancia de los datos (identificada por su "key") gestionados por el DataWriter. En el lado del Subscriptor indica que deben mantenerse todas las instancias de los datos (identificadas por sus "key") gestionadas por su DataReader. Estas muestras se mantienen mientras hasta que se lean por medio de una operación "take". El valor de depth no se tiene en cuenta. Esto último supone que el valor será LENGTH_UNLIMITED.
- depth. Profundidad de muestras que deben ser mantenidas.

La configuración del valor "depth" de la calidad de servicio HISTORY, debe ser consistente con el valor de la propiedad "max_samples_per_instance" de la política de servicio RESOURCE_LIMITS de tal manera que debe constatarse la desigualdad mostrada en la fórmula 17.

Ecuación 17. Fórmula que define la consistencia en la política de calidad de servicio History.

$$\text{“depth”} <= \text{“max\_smples\_per\_instance”} \quad (17)$$

Un ejemplo del comportamiento en los componentes por parte de esta política de calidad de servicio, puede verse en la figura 23.

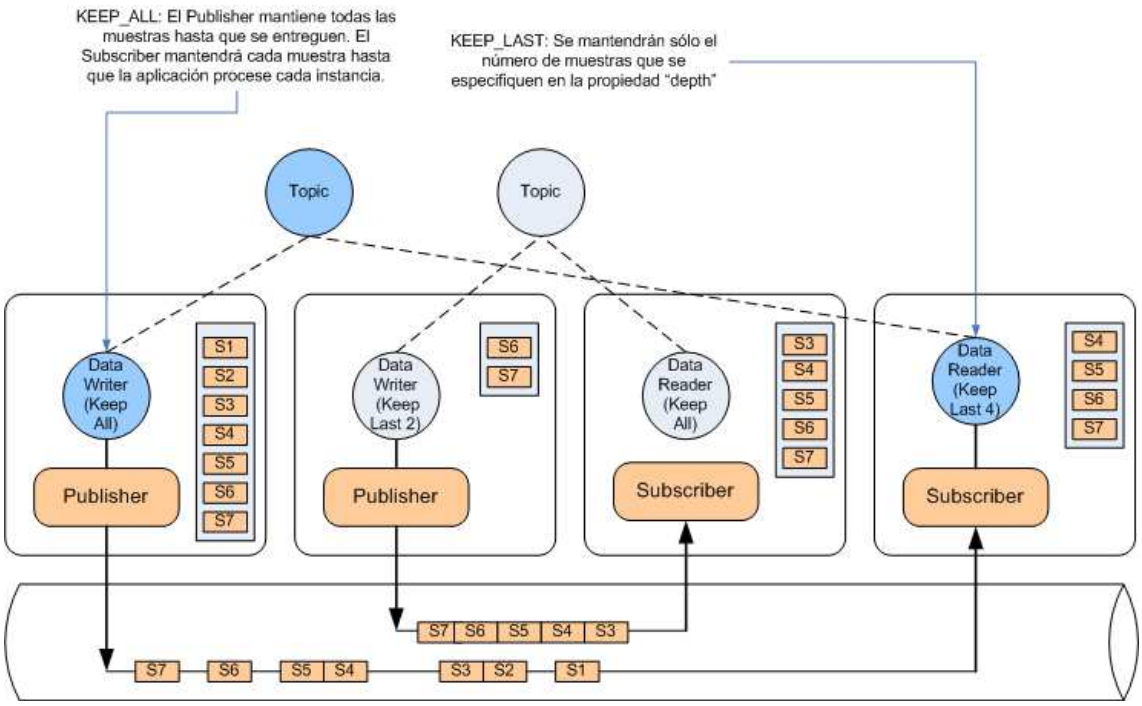

Figura 23. Comportamiento de los componentes de HistoryQoSPolicy.

3.3.6 ResourceLimitsQoSPolicy

Esta característica de la calidad de servicio especifica los recursos que el Servicio puede consumir de acuerdo con la QoS que se haya determinado. El diagrama de clases es el que se muestra en la figura 24.

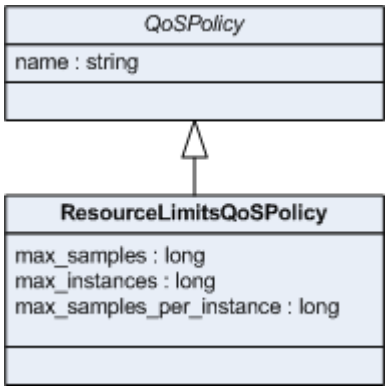

Figura 24. UML de ResourceLimitsQoSPolicy.

Las propiedades que tiene esta política de calidad de servicio son las siguientes.

- max_samples. Determina el número máximo de instancias de datos, que un DataWriter o un DataReader pueden gestionar de entre todas las instancias asociadas con él. Representa el máximo número de muestras que el middleware puede almacenar. Es inconsistente para este valor que sea menor que el número max_samples_per_instance.
- max_instances. Representa el máximo número de instancias que un DataWriter puede gestionar.

- max_samples_per_instance. Representa el máximo número de muestras de cualquier instancia.

Esta política de servicio controla los recursos que el servicio puede o debe usar para cumplir con los requerimientos que la aplicación u otras políticas de calidad de servicio soliciten. En el caso de que los objetos DataWriter estén enviando datos más rápido de los que el DataReader puede recibir, el "middleware" puede, eventualmente, ir en contra de algunas de las limitaciones de recursos impuestas por algunas de las calidades de servicio. El comportamiento, en estos casos, depende de la calidad de servicio RELIABILITY, si está configurada como BEST_EFFORT, entonces se le permite al servicio perder muestras. Si está configurada como RELIABLE, el servicio bloqueará al DataWriter o descartará muestras en el DaraReader, para no perder muestras. La cantidad de muestras que se pueden mantener en un DomainParticipant (especialmente en lo que concierne al DataWriter y sobre todo al DataReader). En DDS se especifica una constante: LENGTH_UNLIMITED que se puede emplear para indicar la ausencia de límites particulares en los parámetros de ésta calidad de servicio. Las condiciones que deben darse para que ésta calidad de servicio sea coherente son las siguientes. La propiedad max_samples debe ser coherente con max_samples_per_instance tal que se cumpla la ecuación 18.

Ecuación 18. Condición de coherencia de la política de calidad de servicio ResourceLimits.

$$\text{“max\_samples”} >= \text{“max\_samples\_per\_instance”} \quad (18)$$

El valor de la propiedad "max_samples_per_instance" debe ser coherente con el valor "depth" de la calidad de servicio HISTORY, verificándose que se cumple la desigualdad definida en 19.

Ecuación 19. Condición de coherencia entre las políticas de calidad de servicio ResourceLimits e History.

$$\text{depth} <= \text{max\_samples\_per\_instance} \quad (19)$$

Es importante que cuando se configure esta calida de servicio, se comprueben los valores para que sean consistentes. Un ejemplo gráfico del comportamiento de la política de calidad de servicio se puede observar en la figura 25.

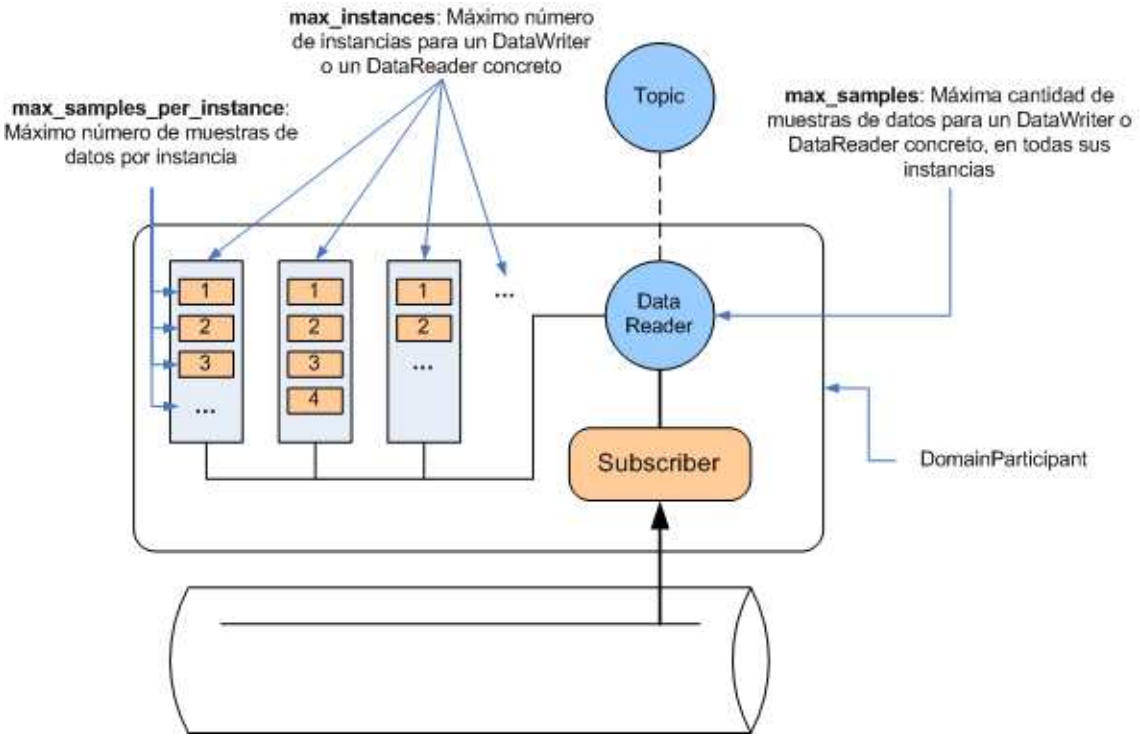

Figura 25. Comportamiento de los componentes de ResourceLimitsQoSPolicy.

3.3.7 OwnershipQoSPolicy

Esta política de calidad de servicio, controla si el servicio permite a varios objetos "DataWriter" actualizar la misma instancia. El diagrama de clase es el mostrado en la figura 26.

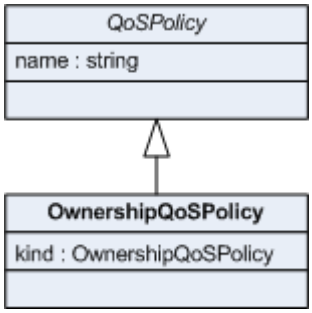

Figura 26. UML de OwnershipQoSPolicy.

La propiedad que tiene es "kind", que puede tomar los siguientes valores, que especifican si se permite a muchos DataWriters escribir la misma instancia

- SHARED. Indica que el propietario es compartido para cada instancia. Esto implica que múltiples escritores tienen permitido escribir sobre la misma instancia, y todas las actualizaciones serán accesibles por parte de los lectores. Lo que es lo mismo, no hay un concepto de propiedad para las instancias. Este valor es por defecto si la calidad de servicio no puede soportar esta característica.
- EXCLUSIVE. Indica que cada instancia es propietaria de un DataWriter. Aunque el propietario puede variar dinámicamente. La selección del propietario es controlada por la política de calidad OWNERSHIP_STRENGTH.

Esta calidad de servicio se utiliza para "capturar" una instancia (identificada por el binomio: Topic+Key) de un objeto de datos. Cuando se activa, sólo unos DataWriters concretos podrán escribir en las instancias de datos. Esta calidad de servicio restringirá por tanto a los DataWriters que también empleen la calidad de servicio OWNERSHIP_STRENGTH que es la que determina el orden de preferencia de escritura en los posibles conflictos de DataWriters que pueden aparecer. Por tanto ambas calidades de servicio están estrechamente relacionadas.

3.3.8 OwnershipStrengthQoSPolicy

Esta política de calidad de servicio especifica la "fuerza" que tiene un "DataWriter" a la hora de negociar algunos aspectos en el envío de mensajes. El diagrama de clase correspondiente es el expuesto en la figura 27.

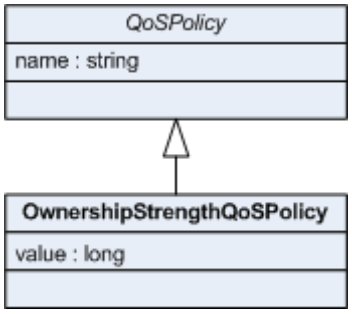

Figura 27. UML de OwnershipStrengthQoSPolicy.

Esta política de calidad de servicio tiene una única propiedad, "value" que especifica el valor de potencia de propiedad de los objetos "DataWriter". Se emplea para arbitrar entre múltiples "DataWriters" cuando desean escribir en la misma instancia. En la figura 28, se puede ver con detalle cómo funciona esta política.

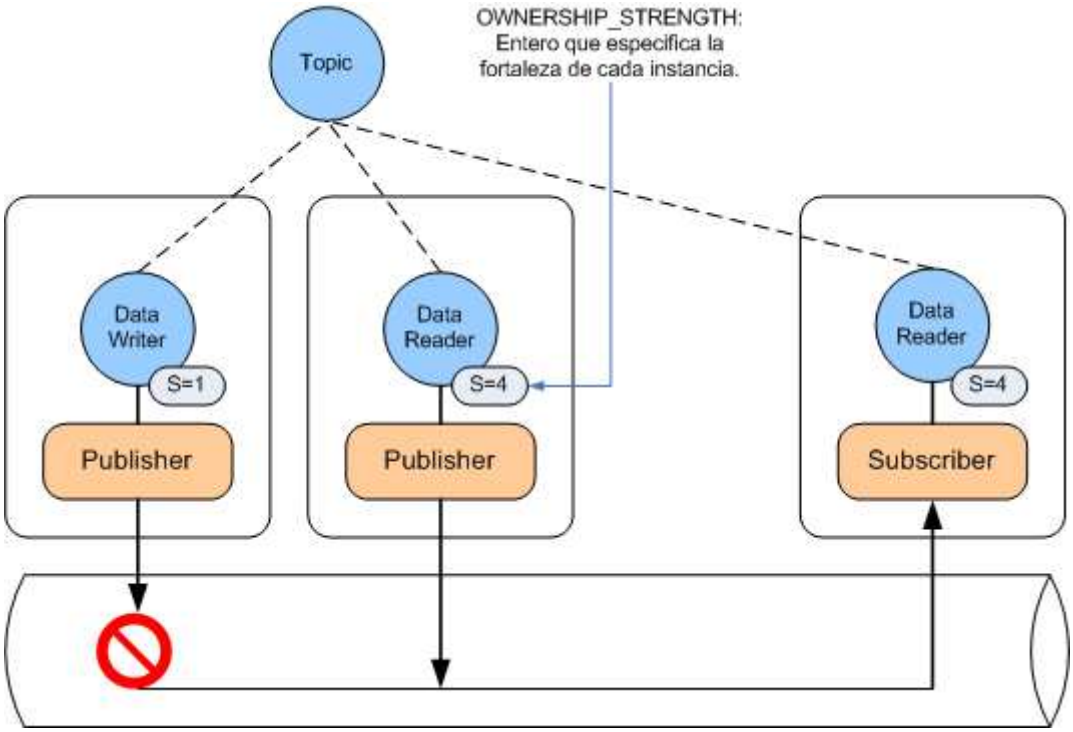

Figura 28. Comportamiento de los componentes de OwnershipStrengthQoSPolicy.

El valor de cada OWNERSHIP_STRENGTH se emplea para aportar una prioridad de escritura frente al resto de los "DataWriter". El arbitraje de la calidad de servicio la realiza el "DataReader".

3.4 Gestión de los componentes

3.4.1 PartitionQoSPolicy

Esta política de calidad de servicio, permite introducir el concepto de la partición lógica de componentes dentro de una partición física de componentes. Ésta partición está inducida por medio de un dominio. El diagrama de clase es el mostrado en la figura 29.

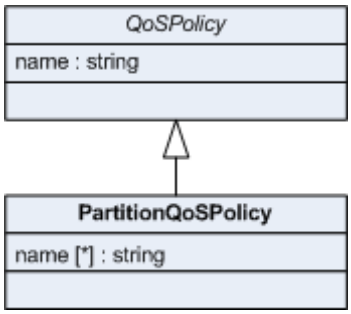

Figura 29. UML de PartitionQoSPolicy.

Esta política de calidad contiene una sola característica: "name". Esta propiedad es un conjunto de cadenas de caracteres que introduce una partición lógica entre los Topics visibles por el Publisher y el Subscriber. Un DataWriter dentro de un Publisher sólo se comunicará con un DataReader en un Subscriber en el caso de que el Publisher y el Subscriber tengan en común el partition name.

Para que un DataReader pueda ver los cambios hechos en una instancia de un DataWriter, no sólo debe coincidir el Topic, sino también deben compartir una "Partition" común. Cada "string" en la lista definida en una QoS, define el nombre de una partición.

Los errores en el "matching" de esta calidad de servicio, no se considera una incompatibilidad (lógicamente). Esta política es variable. Un cambio en esta política, puede potencialmente modificar el "matching" de un DataReader ya existente, o de las QoS de un DataWriter ya existente. Los nombres de las particiones, pueden ser expresiones regulares, que incluyan caracteres comodines, tal como se define en POSIX API (1003.2-1992 sección B.6). Una Entity puede pertenecer sólo a un dominio, pero puede estar en múltiples particiones. En lo que concierne al servicio DDS, cada instancia única de un dato se identifica por la t-upla (domainId, Topic, Key), esto implica que las particiones van a aislar ciertos datos. En la figura 30, se pueden ver gráficamente éstos aspectos.

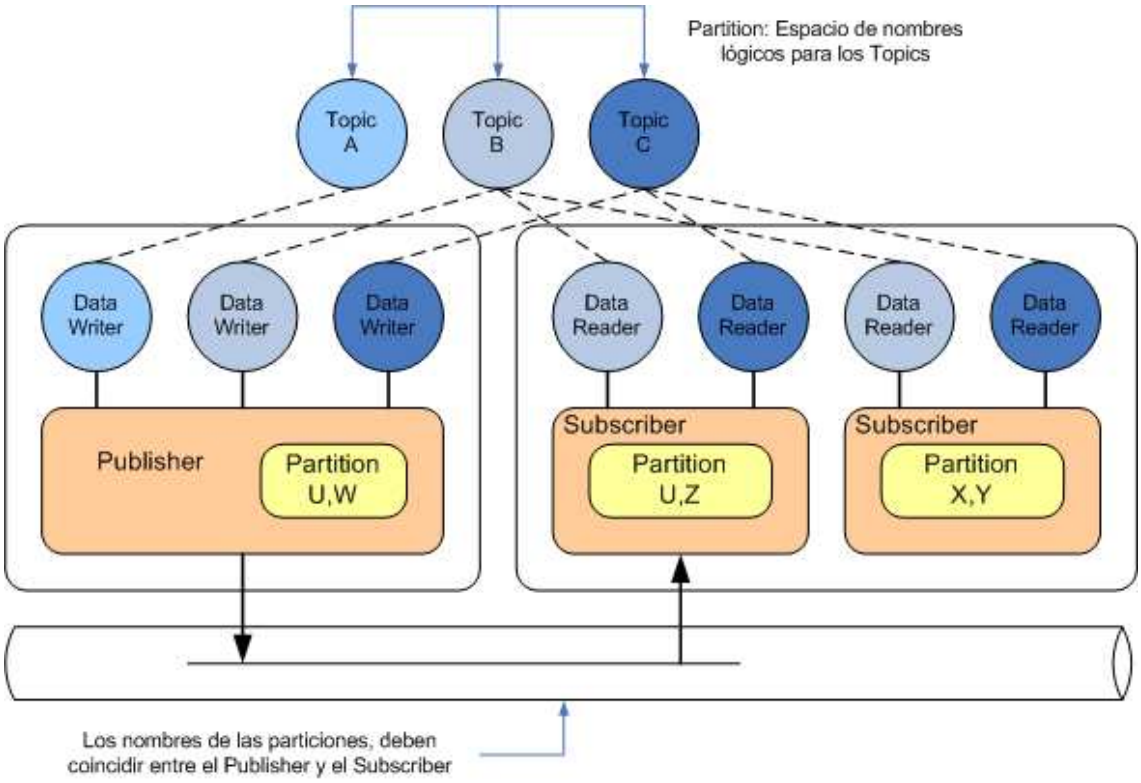

Figura 30. Comportamiento de los componentes de PartitionQoSPolicy.

3.4.2 EntityFactoryQoSPolicy

Esta política de calidad de servicio, controla el comportamiento de una entidad, cuando actúa como factoría de otras entidades. En otras palabras, controla los efectos de las funciones de creación y eliminación de entidades. El diagrama de clases es el mostrado en la figura 31.

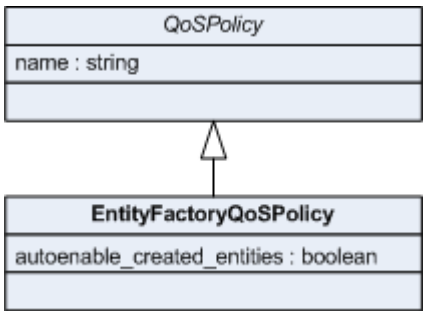

Figura 31. UML de EntityFactorytQoSPolicy.

La única propiedad que tienes es "autoenable_created_entities" y especifica si la factoría habilita automáticamente cada entidad que crea.

3.4.3 WriterDataLifecycleQoSPolicy

Define el comportamiento del DataWriter tendrá sobre las instancias de dato que gestiona. El diagrama de clases es el mostrado en la figura 32.

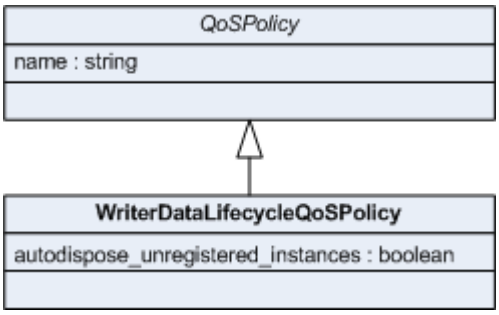

Figura 32. UML de WriterDataLifecycleQoSPolicy.

La única propiedad que tiene es "autodispose_unregistered_instances", esta propiedad indica que las instancias no registradas de DataWriters también están dispuestas a funcionar.

3.4.4 ReaderDataLifecycleQoSPolicy

Especifica el comportamiento del DataReader de acuerdo con el ciclo de vida de las instancias de datos que gestiona. El diagrama de clases de la política es el mostrado en la figura 33.

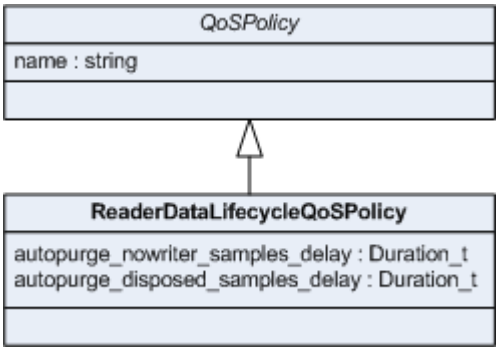

Figura 33. UML de ReaderDataLifecycleQoSPolicy.

Esta política contiene dos propiedades, que son las siguientes.

- autopurge_nowriter_samples_delay. Indica la duración que el DataReader debe mantener o retener la información de las instancias que tengan en instance_state en modo NOT_ALIVE_NO_WRITERS.
- autopurge_disposed_samples_delay. Indica la duración que el DataReader debe mantener o retener la información de las instancias que tengan en instance_state en modo NOT_ALIVE_DISPOSED.

4 Análisis

4.1 Resumen de las políticas de calidad de servicio de DDS

Las políticas de calidad de servicio especificadas en el DCPS del modelo DDS propuesto por OMG, son muy variadas, por lo que parece aconsejable ofrecer una visión general de todas las políticas. Para ello se han organizado en la tabla 2, en la que se exponen las políticas de calidad de servicio en filas, y en columnas las características de cada política de calidad de servicio.

- Columna con los elementos del modelo DCPS sobre los que se aplica la política de calidad de servicio concreta. Estos elementos son los siguientes:
  - DP = Domain Participant
  - DR = Data Reader
  - DW = Data Writer
  - Tp. = Topic
  - Pb. = Publisher
  - Sb. = Subscriber
- Columna RxO (Requested/Offered) con las características de negociación de los valores de las políticas de calidad de servicio, esta columna puede tomar los siguientes valores.
  - Sí. Indica que la política de calidad de servicio debe estar tanto en el "Publishing"[1] como en el "Subscriber" y la combinación de valores de los elementos implicados debe ser compatible.
  - No. Indica que la calidad de servicio debe estar tanto en el "Publishing" y en el "Subscriber", pero los dos valores pueden ser independientes. Esto implica que todas las posibles combinaciones son compatibles.
  - N/A. Indica que la política puede estar tanto en el lado del "Publishing" como del "Subscriber", pero no en ambos. No se aplica ninguna comprobación de compatibilidad.
- Columna Ch, (Changeable) que muestra la variabilidad de la política de calidad de servicio. Esta columna indica si la calida de servicio puede ser alterada después de que la Entidad a la que se está aplicando la calidad de servicio esté activada, o lo que es lo mismo en funcionamiento.

En la tabla, se ha creído conveniente escribir los nombres de las políticas de calidad de servicio con la misma denominación que tienen como clases en el modelo de clases que propone OMG para el DCPS.

[1] Sería interesante determinar si usar los términos en inglés o las traducciones en castellano. Quizás emplear el término en inglés sea más sencillo, ya que no hay que complicarse con la traducción (muy adecuado en casos como "middleware".

Tabla 2. Resumen de las características de las políticas de calidad de servicio.

Grupo	Política de calidad de servicio	Elementos						RxO	Ch.
		DP	DR	DW	Tp.	Pb.	Sb.		
Meta Datos	UserDataQoSPolicy	Sí	Sí	Sí	Sí			No	Sí
	TopicDataQoSPolicy				Sí			No	Sí
	GroupDataQoSPolicy					Sí	Sí	No	Sí
Gestión temporal de mensajes	DurabilityQoSPolicy		Sí	Sí	Sí			Sí	No
	DurabilityServiceQoSPolicy			Sí	Sí			No	No
	DeadlineQoSPolicy		Sí	Sí	Sí			Sí	No
	LatencyBudgetQoSPolicy		Sí	Sí	Sí			Sí	Sí
	LivelinessQoSPolicy		Sí	Sí	Sí			Sí	No
	TimeBasedFilterQoSPolicy		Sí					N/A	Sí
	LifespanQoSPolicy			Sí	Sí			N/A	Sí
Gestión del flujo de mensajes	PresentationQoSPolicy					Sí	Sí	Sí	No
	ReliabilityQoSPolicy		Sí	Sí	Sí			Sí	No
	TransportPriorityQoSPolicy			Sí	Sí			N/A	Sí
	DestinationOrderQoSPolicy		Sí		Sí			Sí	No
	HistoryQoSPolicy		Sí	Sí	Sí			No	No
	ResourceLimitsQoSPolicy		Sí	Sí	Sí			No	No
Gestión de los componentes	OwnershipQoSPolicy				Sí			Sí	No
	OwnershipStrengthQoSPolicy			Sí				N/A	Sí
	PartitionQoSPolicy					Sí	Sí	No	Sí
	EntityFactoryQoSPolicy	Sí				Sí	Sí	No	Sí
	WriterDataLifecycleQoSPolicy			Sí				N/A	Sí
	ReaderDataLifecycleQoSPolicy		Sí					N/A	Sí

En la tabla anterior se puede observar cómo los elementos básicos de la comunicación, los DataWriter, los DataReader y los Topic, son los que tienen más políticas de calidad de servicio que les afecten. Mientras que de las cuatro políticas de calidad de servicio que afectan al Publisher y al Subscriber, tres son exclusivas y tienen funciones de gestión de la topología de las comunicaciones.

Tabla 3. Resumen de los parámetros en las funciones de las políticas de calidad de servicio.

Grupo	Política de calidad de servicio	Propiedades (Nombre : Tipo)
Meta Datos	UserDataQoSPolicy	value[*] : char
	TopicDataQoSPolicy	value[*] : char
	GroupDataQoSPolicy	value[*] : char
Gestión temporal de mensajes	DurabilityQoSPolicy	kind : DurabilityQoSKind
	DurabilityServiceQoSPolicy	service_cleanup : Duration_t
		history_kind : HistoryQoSKind
		history_depth : long
		max_samples : long
		max_instances : long
		max_samples_per_instance : long
	DeadlineQoSPolicy	period : Duration_t
	LatencyBudgetQoSPolicy	period : Duration_t
	LivelinessQoSPolicy	lease_duration : Duration_t
		kind : LivelinessQoSKind

Grupo	Política de calidad de servicio	Propiedades (Nombre : Tipo)
	TimeBasedFilterQoSPolicy	minimum_separation : Duration_t
	LifespanQoSPolicy	duration : Duration_t
Gestión del flujo de mensajes	PresentationQoSPolicy	access_scope : PresentationQoSAccessScopeKind
		coherent_access : boolean
		ordered_access : boolean
	ReliabilityQoSPolicy	kind : ReliabilityQoSKind
		max_bloking_time : Duration_t
	TransportPriorityQoSPolicy	value : long
	DestinationOrderQoSPolicy	kind : DestinationOrderQoSKind
	HistoryQoSPolicy	kind : HistoryQoSKind
		depth : long
	ResourceLimitsQoSPolicy	max_samples : long
		max_instances : long
		max_samples_per_instance : long
Gestión de los componentes	OwnershipQoSPolicy	kind : OwnershipQoSPolicy
	OwnershipStrengthQoSPolicy	value : long
	PartitionQoSPolicy	name [*] : string
	EntityFactoryQoSPolicy	autoenable_created_entities : boolean
	WriterDataLifecycleQoSPolicy	autodispose_unregistered_instances : boolean
	ReaderDataLifecycleQoSPolicy	autopurge_nowriter_samples_delay : Duration_t
		autopurge_disposed_samples_delay : Duration_t

Como se puede observar en la tabla 3, la mayor parte de los parámetros de configuración de las políticas de calidad de servicio pertenecen al ámbito de control temporal de los mensajes y el control del flujo de mensajes. Además, como se vio en la tabla , los componentes del modelo DCPS en los que estas dos áreas se centran son los "DataWriter", "DataReader" y los "Topic".

El hecho de que estos tres componentes sean los que acumulan la mayor cantidad de políticas de calidad de servicio, da lugar a emplearlos como elementos responsables de la comunicación con los elementos de control. Para conocer con detalle cómo se definen estos elementos, se verán a continuación.

4.2 Métodos de los elementos de comunicación

Los elementos principales de las comunicaciones en el modelo DCPS, son los que interactúan con la información. Los principales son los objetos "Publisher", "DataWriter", "Subscriber", "DataReader" y los "Topic". Debido a que se pueden organizar en áreas comunes independientemente de los componentes de que se traten.Los elementos más sencillos en el modelo DCPS son los Topic, que sólo tienen un método diferente a los comunes (que se verán más adelante) el método es "get_inconsistent_topic_status", cuya funcionalidad es devolver el estado del "Topic" en caso de que éste tenga algún problema de inconsistencia, especialmente en lo que a gestión de las políticas de calidad de servicio. Para el resto de los elementos, parece conveniente agrupar por áreas los métodos, ya que existen muchas similitudes, en lo que a funcionalidad se refiere, entre ellos.

Los componentes "Publisher" y "Subscriber" son comunes, independientemente del tipo de dato con el que se trabaje. En estos dos componentes, que albergarán a los

"DataWriter" y "DataReader", propios de cada tipo de dato que se comunicará. En la tabla 4, se muestran las funciones de éstos componentes, organizadas en tres áreas principales, destacando la gestión de las políticas de calidad de servicio.

Tabla 4. Métodos de los componentes "Publisher" y "Subscriber".

Área	Publisher	Subscriber
Gestión de las políticas de calidad de servicio	get_qos	get_qos
	set_qos	set_qos
	copy_from_topic_qos	copy_from_topic_qos
	set_default_datawriter_qos	set_default_datareader_qos
	get_default_datawriter_qos	get_default_datareader_qos
Gestión de los componentes	get_listener	get_listener
	set_listener	set_listener
	create_data_writer	create_datareader
	delete_data_writer	delete_datareader
	lookup_datawriter	lookup_datareader
	get_participant	get_participant
	delete_contained_entities	delete_contained_entities
		get_datareaders
		notify_datareaders
Gestión de la comunicación	begin_coherent_changes	begin_access
	end_coherent_changes	end_access
	suspend_publications	
	resume_publications	
	wait_for_acknowledments	

Los componentes finales que gestionan la comunicación directamente con los datos, son los "DataReader" y "DataWriter", estos componentes son la base para la creación de los "DataReader" o "DataWriter" específicos al tipo de datos que se desea comunicar a través del "Topic". En la especificación oficial de OMG se detallan los métodos mínimos que deben tener los "DataWriter" y "DataReader". En la tabla 5, se muestran los métodos organizados en las mismas áreas que la tabla anterior, y se escriben entre paréntesis.

Tabla 5. Métodos de los componentes "DataWriter" y "DataReader".

Área	DataWriter	DataReader
Gestión de las políticas de calidad de servicio	get_qos	get_qos
	set_qos	set_qos
	get_offered_deadline_missed_status	get_requested_deadline_missed_status
	get_offered_incompatible_qos_status	get_requested_incompatible_qos_status
	get_liveliness_lost_status	get_liveness_changed_value
	assert_liveliness	
Gestión de los componentes	get_listener	get_listener
	set_listener	set_listener
	(register_instance)	
	(register_instance_w_timestamp)	
	(unregister_instance)	
	(unregister_instance_w_timestamp)	
	(lookup_instance)	(lookup_instance)
	(get_key_value)	(get_key_value)

Área	DataWriter	DataReader
	get_topic	get_topicdescription
	get_publisher	get_subscriber
Gestión de la comunicación	(write)	(read)
	(write_w_timestamp)	(take)
	(dispose)	(read_w_condition)
	(dispose_w_timestamp)	(take_w_condition)
	wait_for_acknowledgements	(read_next_sample)
	get_publication_matched_status	(take_next_sample)
	get_matched_subscription_data	(read_instance)
	get_matched_subscriptions	(take_instance)
		(read_next_instance)
		(take_next_instance)
		(read_next_instance_w_condition)
		(take_next_instance_w_condition)
		get_subscription_matched_status
		get_matched_publication_data
		get_matched_publications
		create_readcondition
		create_querycondition
		delete_readcondition
		get_sample_lost_status
		get_sample_rejected_status
		delete_contained_entities
		wait_for_historical_data
		(return_loan)

Cabe destacar que los métodos que se consideran imprescindibles para la implementación, no incluyen las calidades de servicio como imprescindibles. Sin embargo, en lo que a los "DataWriter" se refiere, sí que se recomienda el empleo del "timestamp" en las funciones de escritura, mientras que en los "DataReader", se recomienda el empleo de la lectura condicional basada en los componentes "Condition".

Todos los componentes principales de la comunicación, en el modelo DCPS, tienen dos aspectos básicos comunes, la asignación de las políticas de calidad de servicio y de los componentes "Listener". Formalmente esto implica que las políticas de calidad de servicio se pueden aplicar a todos los elementos y que la conexión con la aplicación, que proporcionan los elementos "Listener", se puede realizar en cualquiera de los niveles de comunicación.

5 Conclusiones

La cantidad de información que manejan los sensores complejos exige, por parte del sistema de comunicaciones, unas características, tanto de su arquitectura física (HW) como lógica (SW) que no son habituales en la mayor parte de middlewares de comunicaciones. Entre las características más destacables está la variabilidad de las características de la información, especialmente en lo que se refiere al volumen, a los requisitos temporales y el tipo de flujo de mensajes que se requiere. El sistema de comunicaciones empleado debe ser capaz de gestionar la variabilidad de las características.

El OMG DDS (Data Distribution Service) es una especificación de sistemas de distribución de datos según el modelo de publicación-suscripción, que conecta a los productores de información con los consumidores, todos ellos anónimos y desacoplados en tiempo, espacio y flujo de datos, proporcionando un modelo independiente de la plataforma orientado a sistemas distribuidos de tiempo real. La configuración se realiza mediante calidades de servicio (QoS) que se emplea como la forma de describir el comportamiento de un servicio en función de unos parámetros concretos negociados. Se estructura formalmente en dos componentes: el DCPS (Data Centric Public Subscribe) que distribuye los datos en el sistema distribuido y el DLRL (Data Local Reconstruction Layer) que permite a las aplicaciones acceder a los datos globales adaptados como si fueran locales. El DCPS es obligatorio, mientras que el DLRL depende de las necesidades que tengan los sistemas para realizar abstracciones de datos.

Al especificar una interfaz de implementación que puede adaptarse a las necesidades del sistema, DCPS constituye una plataforma adecuada para dar un soporte de comunicaciones eficaz a los sistemas de control distribuido, ya que organiza de forma sencilla los componentes básicos que se precisan en las comunicaciones entre los diversos componentes de un sistema distribuido. El hecho de emplear la calidad de servicio para definir una gran cantidad de parámetros temporales y de flujo de información sobre los que basar la comunicación de los componentes, ofrece la posibilidad de adaptar el sistema a las condiciones particulares que se puedan dar en el sistema a lo largo de todo el proceso de control.

El modelo DCPS permite gestionar los requisitos de tiempo real, tanto estricto como blando, el volumen de información que se debe transmitir y el flujo de mensajes, por lo que parece ser un modelo muy adecuado para la gestión de los sensores complejos, en función de las características que tengan éstos.

6 Referencias

[Aurrecoechea et al., 1998] Aurrecoechea, C., Campbell, A.T. and L. Hauw, "A Survey of QoS Architectures", ACM/Springer Verlag Multimedia Systems Journal, Special Issue on QoS Architecture, Vol. 6 No. 3, pg. 138-151, May 1998.

[Houston, 1998] P. Houston. Building distributed applications with message queuing middleware – white paper. Technical report, Microsoft Corporation, 1998.

[Matteucci, 2003] Matteucci., M. "Publish/Subscribe Middleware for Robotics: Requirements and State of the Art", Technical Report N 2003.3, Dipartimento di Elettronica e Informazione - Politecnico di Milano, Milano I, 2003.

[OMG, 2005] OMG. “Data Distribution Service for Real-Time Systems, v1.1.” Document formal/2005-12-04. December 2005.

[Pardo-Castellote, 2003] Pardo-Castellote, G. OMG Data-Distribution Service: architectural overview. Proceedings of 23rd International Conference on Distributed Computing Systems Workshops. Providence, USA. Vol. 19-22, pp. 200-206. (2003)

[UML Group, 1997] The UML Group. UML Metamodel. Version 1.1, Rational Software Corporation, Santa Clara, CA-95051, USA, September 1997.