Revisión de los Sistemas de Comunicaciones más empleados en Control Distribuido

Autor:

José 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: 0.4

Fecha revisión: 10 de noviembre de 2009

Contenidos

Figuras

Tablas

1 Introducción

1.1 Resumen

Los sistemas distribuidos basan su funcionamiento en la posibilidad de comunicación entre sus componentes. El sistema de comunicaciones cubre aspectos de diversa complejidad, ya que incluye cuestiones que van desde la conexión y direccionamiento de los mensajes hasta el formato en que se transmite el contenido de los mismos.

En este documento se realiza una revisión de los sistemas de comunicaciones que dan soporte a la distribución de componentes de los sistemas distribuidos. La revisión se ha planteado desde dos puntos de vista: el de los aspectos teóricos de los paradigmas clásicos de comunicación y el de la visión práctica de los sistemas actuales de comunicación. El documento expone los sistemas separadamente para posteriormente unirlos conceptualmente y analizarlos desde una perspectiva global.

1.2 Historial de revisiones

Nº revisión	Fecha	Comentarios
0.0	2007-02	Inicio del documento
0.1	2007-04	Separación del modelo DCPS de DDS en documento aparte.
0.2	2008-02	Final de revisión de los paradigmas de comunicación
0.3	2008-03	Separación de la calidad de servicio como documento aparte.
0.4	2009-07	Revisión global del documento.

1.3 Objetivos del documento

El objetivo principal del documento es realizar un análisis de los sistemas de comunicaciones más empleados, sin dejar de lado aquellos que por su especial interés deban ser revisados, para así poder extraer las características más destacables y poder aplicarlas en la elección del modelo de comunicaciones más adecuado a un sistema de control distribuido.

1.4 Alcance y audiencia

El documento cubre los sistemas de comunicaciones más empleados y conocidos en la actualidad, sin entrar en los detalles técnicos de implementación, ya que se trata de ofrecer una visión global de los mismos. El documento será de utilidad a diseñadores de sistemas distribuidos.

1.5 Organización del documento

El documento se inicia con una breve revisión sobre las características de los sistemas distribuidos y el papel de la calidad de servicio en los mismos, ya que es uno de los conceptos que van adquiriendo más importancia. En el capítulo 3 se revisan los sistemas de comunicaciones. Inicialmente se revisarán los paradigmas de comunicaciones, para continuar revisando los sistemas de comunicaciones más relevantes. En el capítulo 4 se dan las conclusiones analizando los sistemas de comunicaciones revisados en el capítulo anterior por medio de una contextualización de los mismos y con una revisión de las características más deseables.

2 Sistemas distribuidos

Un sistema distribuido se define como aquel en que los componentes localizados en computadores, conectados en red, comunican y coordinan sus acciones únicamente mediante el paso de mensajes [Coulouris et al., 2001]. Es habitual que los sistemas distribuidos se caractericen por la concurrencia de los componentes, la ausencia de un reloj común y la independencia en los fallos. La concurrencia permite que los recursos disponibles, incluidos los propios componentes, puedan ser utilizados simultáneamente. La ausencia de de un reloj común, o global, se debe a que la transferencia de mensajes entre los componentes no tienen una base temporal común sino que el factor temporal está distribuida entre los componentes. Finalmente el aislado de los fallos de los componentes implica que cada componente debe tener una independencia con respecto al resto de manera que pueda seguir funcionando aunque otro componente no se encuentre disponible. Esto último se puede matizar en la medida en que un componente dependa de otro para su funcionamiento.

La coordinación de los componentes de un sistema distribuido se debe realizar para lograr varios objetivos para los que dichos componentes son necesarios. Para alcanzar dichos objetivos, los componentes deber realizar diversas tareas de manera independiente y servirse concurrentemente de los recursos que tienen a su disposición. La comunicación entre componentes obliga a que exista un intercambio de información entre los mismos. Este intercambio de información se debe realizar a través de los medios de comunicación que el sistema distribuido proporciona a cada componente y además implicará una topología de las comunicaciones. El acceso de los componentes a los medios de comunicación disponibles obliga a la existencia de uno o varios protocolos de comunicaciones, que forzosamente deberán conocer los componentes que se comuniquen a través de los mismos.

El intercambio de información entre componentes supondrá que deberá existir una manera de enviar la información de manera que sea comprensible por los componentes implicados en el intercambio. Consecuentemente el contenido de los mensajes deberá disponer de un lenguaje, o al menos unas normas conocidas por los agentes que intervengan en el proceso de intercambio de información. La existencia de un lenguaje, supone la existencia de una gramática y consiguiente de una sintaxis. Estos aspectos del lenguaje de intercambio de información deberán ser tenidos en cuenta a la hora de diseñar un sistema distribuido y de intentar lograr un alto grado de estandarización del mismo [FIPA, 1997].

2.1 Características

Un sistema distribuido puede tener muchas propiedades deseables, estas enriquecerán el sistema de manera que lo hagan más eficiente. Entre las propiedades deseables de un sistema distribuido se pueden destacar las siguientes [Coulouris et al., 2001].

- Heterogeneidad. Se entiende como la variedad y la diferenciación de los componentes de un sistema distribuido. Generalmente la variación de los diversos componentes de un sistema justifica la creación de estándares que permitan a los componentes conocer las reglas de funcionamiento con un esfuerzo mínimo de adaptación.

- Extensibilidad. Este concepto es similar en algunos casos a la compatibilidad que el sistema puede ofrecer a nuevos componentes. La extensibilidad del sistema también justifica el uso de estándares, cuanto menos las "normas" del sistema deben ser conocidas por los componentes. También suele hablarse de sistemas abiertos cuando se refiere a sistemas extensibles.
- Seguridad. Actualmente, es una de las propiedades que proporciona mayor valía a un sistema distribuido. Sobre seguridad hay tres aspectos que son importantes tener en cuenta: disponibilidad, confidencialidad e integridad. La disponibilidad se refiere a que los recursos del sistema están accesibles para aquellos componentes que lo requieran, en el instante en que sea necesario. Confidencialidad es la protección que el sistema proporciona para no ser accesible por componentes no autorizados. Asimismo la confidencialidad debe proporcionar la suficiente seguridad a un componente, que los componentes con los que está trabajando son realmente los que él requería para trabajar, es decir, además de evitar la intrusión, un sistema confiable debe proporcionar suficientes garantías para evitar la suplantación. Finalmente la integridad se refiere a la protección que el sistema suministra contra la alteración, ya sea accidental como provocada, de los datos que circulan por el sistema.
- Escalabilidad. Un sistema distribuido es escalable en la medida en que se pueden aumentar sus componentes si que esto conlleve una pérdida de efectividad debida a los cambios. El objetivo final de un sistema, con respecto a la escalabilidad, es que apenas deben realizarse cambios en el mismo pese al aumento de los componentes. Normalmente es un gran logro que un sistema que se adapta a las necesidades de los componentes cuando la cantidad de estos cambia, no requiera de modificaciones.
- Soporte a fallos. Los componentes de un sistema distribuido pueden fallar y es normal que esos fallos antes o después se produzcan. Un fallo en uno o varios componentes hará que el resultado del sistema sea diferente o de menor calidad que el esperado, por ello la prevención y tratamiento de los fallos es un aspecto que deberá tener muy en cuenta un sistema distribuido. Algunas características que deben tener los sistemas distribuidos con respecto a los fallos son la detección, el enmascaramiento, la tolerancia y la recuperación. La detección del fallo debe realizarse lo antes posible para evitar que los errores en el procesamiento tengan consecuencias colaterales en el resto de los componentes. Una vez detectado el fallo, éste debe enmascararse o atenuarse, es decir que aunque el fallo es conocido, los componentes no se ven afectados. La tolerancia a fallos implica que los componentes conozcan el fallo y reaccionen frente al mismo de una manera conocida y controlada. Finalmente la recuperación es la fase en la que el sistema, una vez solucionado el fallo, vuelve a funcionar según los requerimientos.
- Concurrencia. La concurrencia es la capacidad que tiene un sistema para que un recurso, pueda ser utilizado simultáneamente. Realmente la simultaneidad puede no ser completa, sino a efectos de los componentes que acceden al recurso. La concurrencia supondrá el uso de diversas estrategias de acceso al recurso con el objetivo de mantener la integridad de la información.
- Transparencia. La transparencia de un sistema es la ocultación al usuario de los detalles que son propios de la gestión interna del sistema. De esta manera, los usuarios de un sistema no lo perciben con más detalles que el estrictamente

necesario. La transparencia se logra de diversas maneras, pero fundamentalmente a partir del encapsulado de componentes y del uso de interfaces.

A medida que un sistema distribuido de soporte a las propiedades anteriores sus componentes, y consiguientemente el sistema en su conjunto verán aumentada su eficiencia e incrementado su valor. Para poder proporcionar estas características, se deberá realizar un esfuerzo considerable, tanto en el diseño, como en la monitorización, gestión y control del sistema, por lo que estos aspectos deberán ser tratados con especial atención en el desarrollo del sistema.

2.2 El papel de la calidad de servicio

Para lograr que un sistema distribuido cumpla unos requisitos adecuados, o lo que es lo mismo, que logre cumplir las características que se le requieren, es necesario determinar unos parámetros en los que basar las características. Estos parámetros, deben ser mesurables y representativos. En los sistemas distribuidos basados en componentes que ofrecen unos servicios a las aplicaciones que sobre ellos funcionan, los parámetros que se miden y permiten determinar el buen funcionamiento del sistema, se conocen como parámetros de calidad de servicio, se pueden observar en la figura 1.

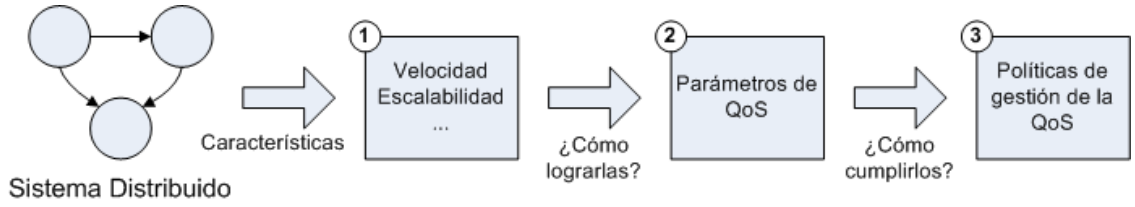

Figura 1. Necesidad de las políticas de QoS en un sistema distribuido.

Los parámetros de calidad de servicio proporcionan un medio por el que poder tomar decisiones acerca de los componentes, y de los mismos servicios, del sistema. Para poder cumplir esos requisitos se debe gestionar los parámetros entre los componentes. La gestión, incluida la negociación, de los parámetros de calidad de servicio entre los componentes se realiza por medio de políticas de calidad de servicio.

3 Sistemas de comunicaciones

La topología de las conexiones en las comunicaciones se refiere a cómo se conectan los componentes que se van a comunicar. Desde este punto de vista se pueden distinguir diversos modelos, tal como se describe en [Liu, 2004]. Para determinar el correcto ámbito en el que se puede ubicar cada uno de los paradigmas de comunicaciones, se puede atender al contenido que es comunicado, que puede ser datos, código o ambas cosas. En la tabla 1, se puede ver esta organización.

Tabla 1. Tipo de comunicación en función del contenido que se transmite.

Contenido de la comunicación		Tipo de comunicación
Datos	**Código**	
No	No	No existe comunicación. No hay transferencia de información, consecuentemente no es un paradigma de comunicación.
Sí	No	Comunicación clásica. El código del emisor envía la información al receptor por medio de los datos que el código del receptor procesará.
No	Sí	Código móvil. El emisor transfiere código al receptor, generalmente posteriormente se transfieren los datos a procesar.
Sí	Sí	Movimiento de componentes. Se transfiere el código y los datos con los que trabajar, por lo que normalmente se entiende como movimiento de procesos.

Los paradigmas de comunicación que se presentan en [Liu, 2004] se pueden organizar en función del contenido en los tres grupos anteriores. El más importante es el de las comunicaciones únicas de datos, donde se pueden ubicar prácticamente todos los paradigmas clásicos de comunicación.

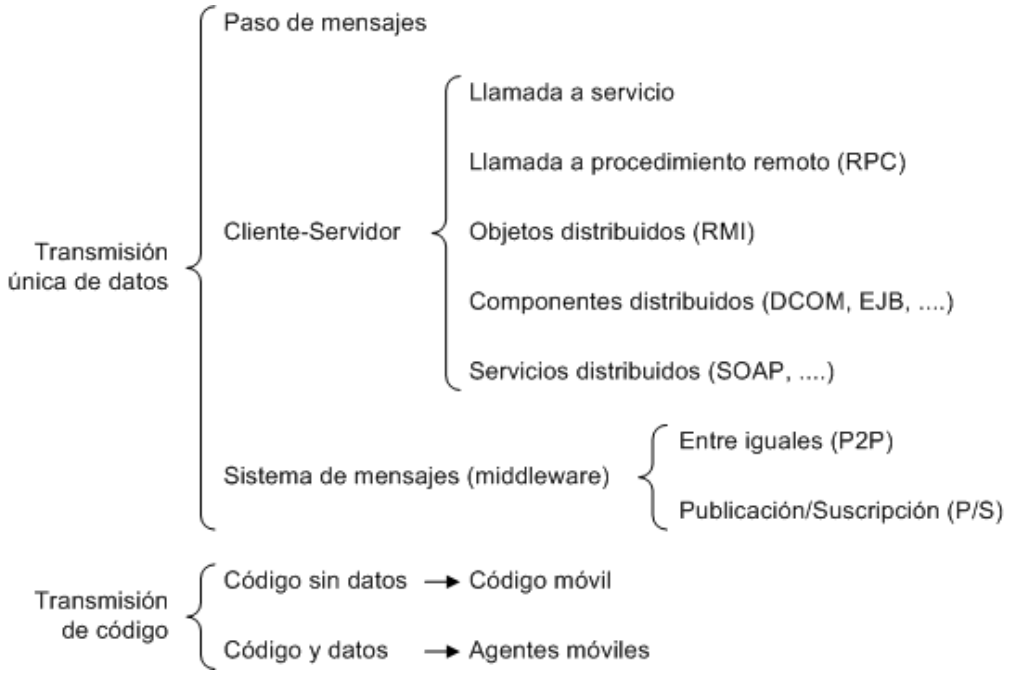

Figura 2. Paradigmas de comunicación en función del movimiento de datos y de código.

Cuando el contenido de los datos es código, entonces se pasa a hablar de movilidad. La transferencia de código precisa del soporte de comunicaciones de los paradigmas anteriores, y adicionalmente la infraestructura necesaria para la ejecución del código. En la figura 2, se puede observar la clasificación de los paradigmas más importantes de comunicaciones en función de la organización anterior.

Como se puede observar, en la transmisión de datos, hay tres grupos importantes, en función del papel que toman cada uno de los componentes de la comunicación. Estos son, de menor a mayor complejidad, los pasos de mensajes, el modelo cliente-servidor y los sistemas de mensajería. Los esquemas de estos tres modelos pueden verse en la figura 3.

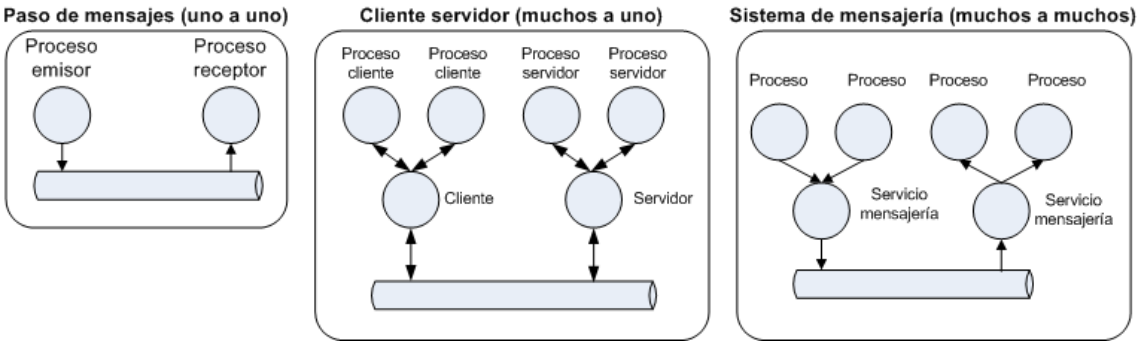

Figura 3. Modelos de comunicaciones basados en paso de mensajes.

El modelo de paso de mensajes es muy sencillo ya que la comunicación la realizan los procesos directamente. El modelo cliente-servidor implica el uso de servicios especializados en conectar y recibir información. Finalmente el modelo basado en sistemas de mensajería, establece unos servicios comunes a los elementos de comunicación.

Realmente el paso de mensajes es la base de los modelos cliente-servidor y de los sistemas de mensajería, ya que todos ellos implican paso de mensajes. El modelo cliente-servidor introduce el concepto de proceso específico servidor que ofrece unos servicios comunes a los procesos que son clientes de él. El modelo basado en un sistema de mensajería unifica cada nodo como cliente y servidor simultáneamente, lo que implica que en todos los nodos se debe tener un proceso especializado en la comunicación, que normalmente es común en todos los nodos. En este último caso suele hablarse de middleware a los procesos que intervienen en la gestión del sistema de mensajería [Gaddah and Kunz, 2003].

3.1 Paradigmas de comunicación

3.1.1 Paso de mensajes

Es el paradigma más sencillo de comunicación a nivel de procesos. Consiste en que dos procesos se comuniquen con mensajes básicos entre ellos. Un proceso envía una solicitud a otro que, al recibirla, puede generar una respuesta. Para este tipo de comunicación, sólo son necesarias dos operaciones de gestión de la comunicación: conexión y desconexión, y dos operaciones de transferencia de la información: enviar y recibir. La programación basada en sockets está basada en este paradigma.

3.1.2 Cliente – Servidor

Cliente-Servidor. El paradigma cliente-servidor se basa en la asignación de dos funcionalidades diferentes a cada uno de los componentes que se comunican. El proceso servidor proporciona una serie de servicios y espera la llegada de peticiones por parte de los clientes. En este modelo, el servidor tiene un papel pasivo, dedicándose a esperar las solicitudes de los clientes, que son los que toman el papel activo.

La mayor parte de los protocolos de Internet siguen este paradigma. El modelo cliente-servidor abarca diferentes formas de implementación, en función de los elementos que se comunican los clientes y los servidores. Se pueden distinguir los siguientes.

- Llamada a procedimiento remoto. Más conocido por sus siglas en inglés RPC (Remote Procedure Call), este paradigma intenta tratar el software distribuido de la misma forma que el local, de manera que en lugar de comunicarse con un mensaje que contiene información, éste mensaje es una llamada a un procedimiento o función, de la misma manera que podría hacerlo en local. Esta llamada evita que los componentes deban conocer todos los detalles del procesamiento, dejando que cada uno de ellos comparta los procedimientos que al resto pueden serles de utilidad. El proceso que llama al procedimiento remoto, debe conocer la interfaz de la llamada para poder realizar el paso de argumentos de manera correcta. Al finalizar el procedimiento, el proceso invocado devuelve el resultado al proceso que realizó la llamada.
- Objetos distribuidos. Al igual que el paradigma de llamada a procedimiento remoto extiende el paradigma de programación funcional a procesos distribuidos, el paradigma de objetos distribuidos extiende a un sistema distribuido el paradigma de programación orientada a objetos. En este paradigma, las aplicaciones acceden directamente a objetos que se encuentran distribuidos en la red, de la misma manera que lo podrían hacer a los locales. Para poder ser accedidos, los objetos deben proporcionar métodos que sea posible invocar como servicios. A esta característica del paradigma de objetos distribuidos se le conoce como Invocación remota de métodos o RMI (Remote Method Invocation) y es el componente principal del paradigma, funcionando de una manera muy similar a las llamadas a procedimientos remotos. Otros componentes son los servicios de red para el acceso a los objetos. El agente gestor de peticiones a objetos, más conocido como “object middleware” y el espacio de objetos.
- Componentes distribuidos. Los servicios de componentes son una extensión del modelos de objetos, pero ampliado a la programación basada en componentes. En ocasiones, el modelo de componentes se incluye dentro del paradigma de objetos distribuidos ya que, tanto objetos como componentes coinciden en esquemas y patrones de actuación. El modelo de componentes es más una tecnología y aplicación práctica que un modelo en sí, lo que hace que cada fabricante tenga sus propios esquemas. Existen varias tecnologías basadas en este modelo; entre ellas cabe destacar, debido a su difusión, “Microsoft’s COM” y su versión distribuida “Microsoft DCOM”, Java Bean, y Enterprise Java Bean (EJB). Realmente estas tecnologías son librerías de objetos diseñadas específicamente para la interacción de los mismos de manera distribuida. Por ello la diferenciación entre el paradigma de objetos distribuidos y el de componentes distribuidos es escasa.
- Servicios distribuidos. A partir de las arquitecturas de sistemas distribuidos, basados en componentes, cada componente puede ofrecer al resto del sistema una serie de servicios. El conjunto de los servicios ofrecidos conforma el paradigma de servicios distribuidos.

3.1.3 Sistemas de mensajería

Sistema de mensajes. Los sistemas de mensajes, constituyen un paradigma basado en una capa intermedia (middleware) que gestiona el paso de mensajes entre componentes. Por ello, se suele hablar de una capa intermedia del paradigma de paso de mensajes. El nombre más conocido de este paradigma es "Message-Oriented Middleware (MOM)". Existen dos modelos de implementación de este paradigma: punto a punto (point to point) y publicación/suscripción (publish/subscribe).

En el modelo punto a punto, el sistema de mensaje emplea colas que permiten desacoplar el emisor del receptor. Cuando un componente desea enviar un mensaje a otro, deposite el mensaje en la cola del receptor (lo que implica conocerlo) y se desentiende del mensaje.

En el modelo de publicación/suscripción, cada mensaje es asociado a un evento específico, de manera que los componentes que estén interesados en recibir los mensajes asociados al evento, se suscriben al mismo. El sistema de mensajes es el responsable de la gestión completa de los mensajes.

- "peer to peer". Con algunos conflictos en la traducción, suele hablarse de modelo entre pares, o entre iguales. En este paradigma las funcionalidades de cada componente son las mismas. En este caso las operaciones que puede hacer cada componente son: solicitud y respuesta. Protocolos como VoIP, usado por skype, están basados en este paradigma. Generalmente se suele combinar este paradigma con el de cliente servidor para establecer las conexiones.
- "Publish-subscriber". Este modelo soporta las comunicaciones de un emisor a muchos receptores con soporte a la redundancia tanto en las publicaciones, como en los receptores que se suscriben a las fuentes de información.

3.1.4 Transmisión de código

Paralelamente al movimiento de datos hacia los nodos de ejecución se puede plantear la posibilidad de mover el código hacia los datos o mover tanto datos como código. Cada uno de estos casos es un paradigma que se puede enmarcar dentro de la transmisión de código.

En el caso de mover sólo código suele hablarse de código móvil, en el caso de mover código y datos ya se pasa a hablar de objetos, componentes o agentes móviles, dependiendo del paradigma de programación en el que se trate.

- Código móvil. En todos los paradigmas anteriores, la información que se transmitía en la red del sistema distribuido eran eventos y datos orientados a que un componente externo ofreciese alguna funcionalidad. Sin embargo, el conocimiento acerca del procesamiento de los datos, el código, debía residir previamente en el componente al que se le solicitaba que realizase alguna acción. En el paradigma de código móvil, lo que se transmite, de manera complementaria a los datos, es el código. Esto supone un paso más en la abstracción del sistema distribuido. La posibilidad de que parte del conocimiento pase de un componente a otro introduce nuevas posibilidades en cuanto al aprendizaje y evolución del sistema.
- Objetos y componentes móviles. El código puede viajar con los datos para ser procesados (código + estado) o bien con más información relativa a la ejecución en el nodo (código + contexto).

- Agentes móviles. El modelo de agentes distribuidos surge de manera paralela a la programación basada en agentes. Es similar al paradigma de objetos distribuidos (y por analogía al de componentes distribuidos), y se caracteriza por estar basado en la tecnología de agentes. Los agentes son una abstracción mayor que los objetos y componentes. Los agentes interactúan entre ellos intercambiando información y reaccionando a eventos de manera que son completamente independientes llegando a apreciarse comportamientos en la forma de actuar.

3.2 Modelos de sistemas de comunicaciones

En la mayor parte de los paradigmas vistos anteriormente de comunicación en los sistemas distribuidos debe existir un módulo de conectividad que permita la interconexión de los componentes. Cuando éste módulo no pertenece a las aplicaciones que comunican, se le conoce como "middleware". La traducción literal del término es "elementos de en medio", algunos autores hacen una traducción completa y hablan de "soporte lógico de intermediación", aunque suele aceptarse el término "software de conectividad" aunque está asimilado el término en inglés. La capa intermedia suele ser un software que se sitúa entre los servicios de la red y las aplicaciones; esta capa está encargada de proporcionar diversos servicios, necesarios para aumentar la efectividad de la comunicación, entre ellos están la identificación, autenticación, autorización, la estructuración jerárquica y movilidad.

Actualmente se pueden localizar desde librerías que dan soporte a la comunicación de forma sencilla, como IPC, DCE o MPI que no definen en sí mismas una arquitectura pero ofrecen el sistema de intercambio de mensajes básico, hasta sistemas comerciales que encapsulan la arquitectura y ofrecen un entorno de desarrollo completo, como Apache ActiveMQ, o IBM Web Sphere MQ que dan un soporte a colas de mensaje con una potencia muy grande.

Los sistemas de comunicación pueden ofrecer la comunicación, por medio de diversos paradigmas, generalmente encapsulándolos en niveles superiores. Existen una gran cantidad de entornos de herramientas que dan soporte a las comunicaciones en los sistemas de control inteligente. Algunos de los entornos son genéricos y se emplean la comunicación de una gran cantidad de sistemas distribuidos, mientras que otros son específicos de una arquitectura de control. A continuación se realiza una revisión de los principales entornos de comunicaciones, tanto genéricos como específicos y se encuadran en alguno de los paradigmas presentados anteriormente.

3.2.1 TCX

TCX [Fedor, 1994] es el sistema de comunicaciones que da soporte a la arquitectura TCA (Task Control Architrecture) desarrollado a partir de 1990 por Chris Fedor en la Universidad de Carnegie Mellon. Es un sistema de comunicaciones basado en el modelo de paso de mensajes, que ha sido empleado en una gran cantidad de sistemas de control inteligente, especialmente en sistemas de navegación de robots, y que ha dado lugar a un producto [Fong, 1998], comercializado como FPT (Fourth Planet Communication) aplicado en diversos robots en la Universidad de Carnegie Mellon [Fong et al., 2001].

TCX está escrito en C, funciona a partir de sockets sobre TCP/IP y se encuentra disponible para una gran cantidad de plataformas. TCX se emplea para dar soporte, tanto al modelo cliente/servidor como al de publicación/suscripción.

El componente principal es un servidor al que se conectan todos los módulos, este servidor funciona como un módulo más dentro de la arquitectura. El servidor mantiene las colas de mensajes, creando una cola por cada módulo que se inicializa, y que sirve de identificador del módulo de comunicaciones. La evolución de TCX son los sistemas IPT, RTC e IPC.

3.2.2 IPT

IPT (Inter-Process Toolkit) es un sistema de comunicaciones basado en el modelo de paso de mensajes, desarrollado por Jay Gowdy a partir del modelo establecido por TCX e incrementando su funcionalidad, lo que hace que TCX sea un subconjunto de IPT. Está orientado a la investigación en el control de robots [Gowdy, 1996].

En la figura 4, se puede observar un esquema del funcionamiento. La comunicación acerca de la estructura del sistema distribuido la realizan los servidores IPT. Los servidores proporcionan a los componentes, organizados en dominios, la información necesaria para localizar los módulos necesarios para comunicarse. Las transferencias de datos la realizan los componentes directamente entre ellos, aunque es necesaria la intervención de los servidores IPT para registrar los mensajes.

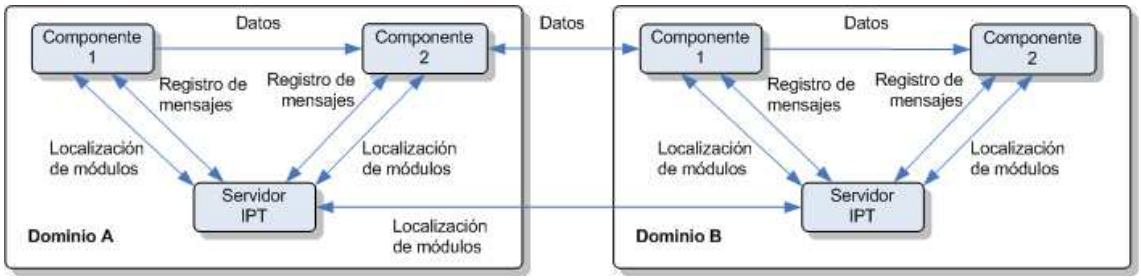

Figura 4. Arquitectura de comunicaciones IPT.

IPT está escrito en C++, y emplea sockets sobre TCP/IP para las comunicaciones de red. IPT extiende el concepto de servidor que tenía TCX, el resultado es un sistema donde los módulos mantienen una comunicación punto a punto. El servidor IPT proporciona principalmente dos servicios: el primero es conectar módulos a partir de una tabla de enrutamiento que abstrae los componentes de los detalles de la red, y como segundo servicio proporciona un espacio para convertir los nombres de los mensajes en números únicos con los que trabajar. Cada cliente, se registra en el servidor y determina una serie de "rutas" en las que está interesado.

3.2.3 RTC

RTC (Real-Time Communications) está desarrollado por Jorgen Pedersen [Pedersen, 1998] para dar soporte a diversos proyectos de robots de NREC (National Robotics Engineering Consortium) en la Universidad de Carnegie Mellon. RTC es un sistema de comunicaciones muy similar a IPT, extendiendo su funcionalidad con el soporte a memoria compartida y a tiempo real. En cuanto a esquema y componentes no difiere mucho de IPT (figura 5).

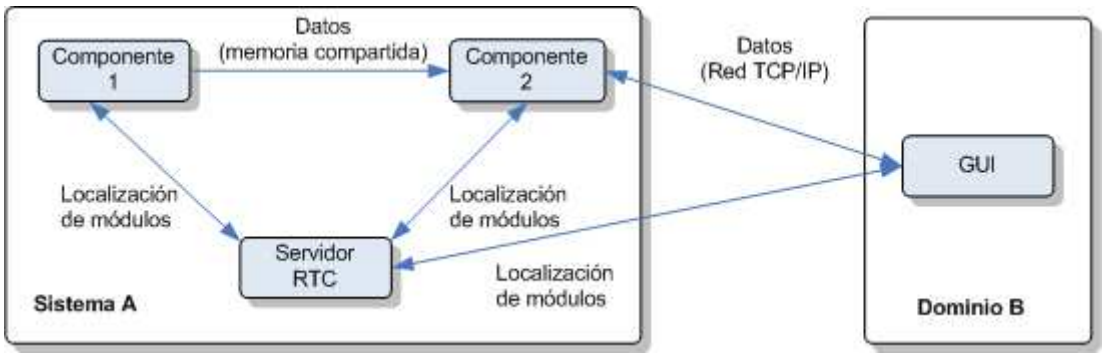

Figura 5. Arquitectura de comunicaciones RTC.

La diferencia fundamental con IPT es que en RTC los mensajes se envían directamente entre componentes, una vez han gestionado la localización y las colas en el servidor. El fundamento del soporte a tiempo real del sistema es el desdoblamiento del medio en que se envían los mensajes, de manera que los mensajes de tiempo real se comunican vía memoria compartida, mientras que el resto de mensajes se comunican por medio de los protocolos de red.

3.2.4 IPC

IPC (Inter Process Communication) la desarrolló Reid Simmons [Simmons and James, 2001] para la misión Deep Space 1 del programa New Millennium de la NASA, aunque finalmente no se empleó, sí que es usado por una gran cantidad de proyectos de la Universidad Carnegie Mellon y la NASA. Proporciona soporte tanto para mensajes centralizados como para mensajes "peer to peer", también tiene soporte a tiempos y mejora el paso de mensajes que realizaba el sistema básico de TCX.

IPC añade a las arquitecturas anteriores el concepto de servidor centralizado, por medio del cual el módulo de control se comunica directamente con los módulos de comunicaciones (figura 6).

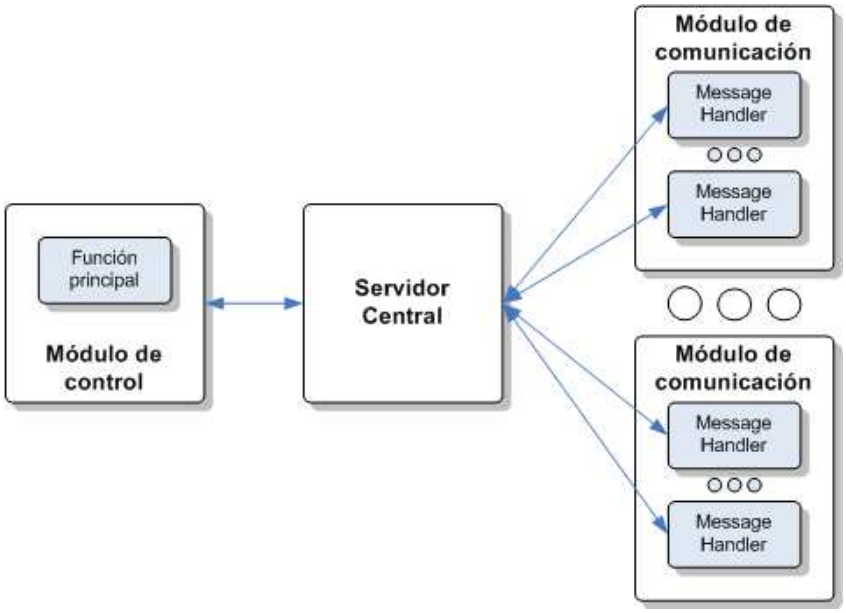

Figura 6. Arquitectura de comunicaciones IPC.

IPC es una librería que requiere un servidor central al que suscribirse los módulos de comunicación de los procesos que requieren recibir los mensajes a los que se han suscrito. Además se tiene un módulo de control con diferentes funcionalidades entre las que se encuentran las de sincronización temporal, envío de mensajes para sincronización de tareas de tiempos limitados y similares.

3.2.5 NML

NML (Neutral Messaging Language) es un paquete de servicios de comunicaciones basado en la información [Shackleford et al., 2000]. Está diseñado en el National Institute of Standards and Technology (NIST). En la figura 7, pueden verse los componentes básicos de funcionamiento. El componente principal es un buffer de memoria compartida, en la que los productores de información escriben sus datos y los consumidores los leen.

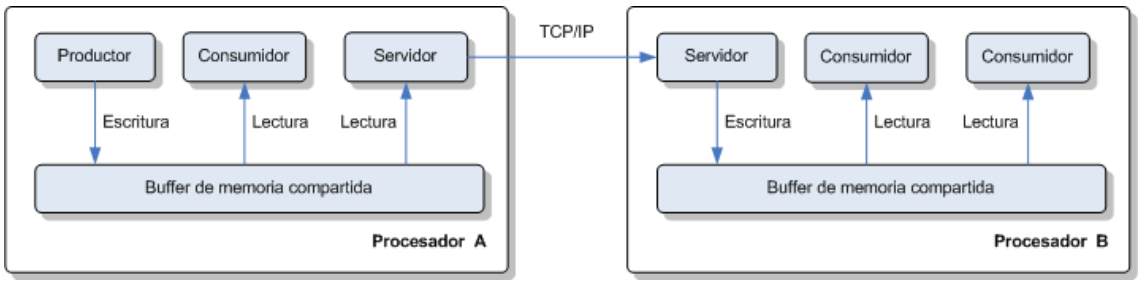

Figura 7. Arquitectura de comunicaciones NML.

En los casos en los que es posible, se emplea la tecnología de memoria compartida, mientras que si se debe realizar la transferencia de información a través de una red, se debe emplear un servidor que gestiona las lecturas y escrituras. Inicialmente se ha desarrollado para las comunicaciones entre robots basados en mensajes síncronos, aunque también tiene un soporte a los mensajes asíncronos. Actualmente se emplea en la arquitectura NASREM. El sistema permite cambios en la configuración, pero no mientras está realizando transferencias.

3.2.6 MPI

Message Passing Interface (MPI) o Interfaz de Paso de Mensajes, es un protocolo estándar de paso de mensajes para ser utilizado entre procesadores con el objetivo de realizar procesamiento paralelo. El protocolo se ha extendido a otros ámbitos aparte del interior de los computadores. Se debe destacar que MPI no define una arquitectura sino un protocolo de comunicación, que básicamente considera una red de nodos iguales con intercambio de mensajes entre ellos.

Entre las características más relevantes de MPI están la sencillez de los métodos, el control de tiempo real y la gran extensión ya que todos los sistemas operativos suelen dar soporte a la comunicación entre procesos por medio de ésta interfaz. Se debe destacar que MPI tiene una capacidad de portabilidad y escalabilidad muy grande, desde los sistemas multiprocesadores a los sistemas distribuidos, considerando cada nodo un sistema de procesamiento que se comunica con el resto.

3.2.7 ACE

ACE (Adaptive Communication Environment, es un entorno de código abierto orientado a C++ iniciado por Douglas C. Schmidt en la Universidad de California Irvine

[Schmidt, 1993]. De este sistema destaca el hecho de estar basado en patrones de diseño [Gamma et al, 1994]. Los patrones de diseño son patrones de soluciones aplicables a situaciones similares en la implementación de software. Uno de los objetivos principales de los patrones de diseño es ofrecer un esquema de actuación reutilizable en diferentes situaciones a lo largo del proceso de diseño del software.

Los patrones de diseño en los que se basa ACE son los siguientes. Conviene revisar la bibliografía correspondiente para comprender el funcionamiento de algunos y su inclusión en un modelo de objetos distribuidos, ya que algunos no están incluidos en los 23 patrones estándares "de facto".

- Patrones de manejo de eventos (Event Handling Patterns): Reactor, Acceptor-Connector, Pro-actor, Items de finalización asíncrona (Asynchronous Completion Token)
- Patrones de sincronización (Synchronization Patterns): Scoped Locking, Strategized Locking, Thread-Safe Interface, Double-Checked Locking Optimization.
- Patrones de concurrencia (Concurrency Patterns): Thread-Specific Storage, Monitor Object, Active Object, Half-Synch/Half-Asynch, Leader/Follower.
- Patrones de servicio de acceso y configuración (Service Access and Configuration Patterns): Component Configurator, Interceptor, Extension Interface, Wrapper Facade.

ACE proporciona tipos comunes de datos y de métodos, lo que hace que pueda considerarse un sistema basado en un modelo de objetos distribuidos. Proporciona una forma estandarizada para una gran cantidad de sistemas operativos que permite, entre otros aspectos, gestionar la comunicación entre procesos, gestión de "threads" y de memoria. Actualmente es empleado en multitud de proyectos comerciales de Motorola o Boeing.

3.2.8 CORBA

En los paradigmas orientados a objetos, la unidad básica sobre la que se diseña el sistema es el objeto a transmitir. En los paradigmas anteriores la base del sistema era la información, sin embargo un objeto contiene la información (atributos) y unas capacidades de procesamiento (métodos). El hecho de que los objetos encapsulen la mayor parte de su funcionalidad hace que sea más sencillo invocar una acción de un objeto que ejecutar un procedimiento como se hacía en RPC, por tanto la abstracción que debe realizar el sistema en estos casos es mucho mayor. El mejor de los referentes de estos sistemas es la arquitectura CORBA (Common Object Resource Broker Architecture) [OMG, 1991].

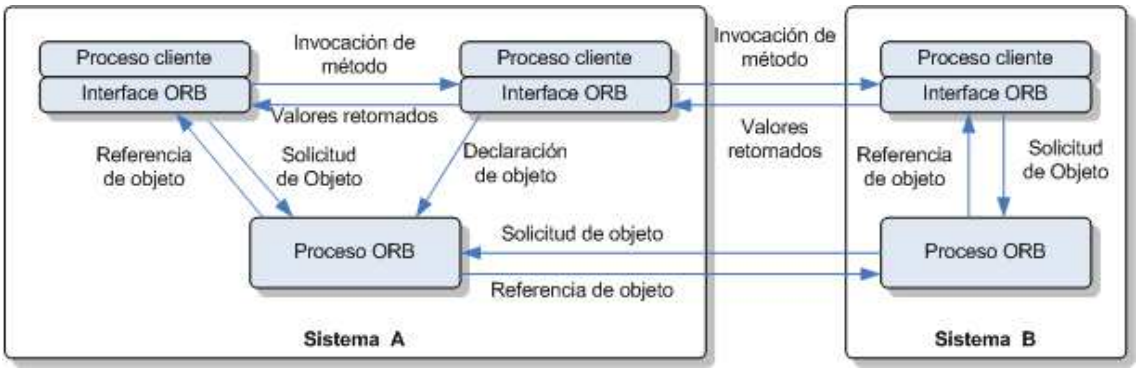

Figura 8. Arquitectura de comunicaciones CORBA.

La última de las especificaciones CORBA [OMG, 1995] ha sido desarrollada por la Organización Internacional OMG (Object Management Group). OMG se fundó en 1989 y está formada por un consorcio de 800 empresas. CORBA permite el desarrollo de software para entornos distribuidos heterogéneos separando la especificación de las aplicaciones de su implementación.

Los componentes y el funcionamiento pueden observarse en la figura 8. La especificación de una aplicación u objeto constituye su interfaz (ORB interface), que es independiente del lenguaje de desarrollo e indica el conjunto de servicios ofrecidos al resto de aplicaciones u objetos y la forma de invocarlos. CORBA proporciona un lenguaje estándar para definir interfaces independiente del lenguaje de programación llamado OMG IDL o CORBA IDL (Interface Definition Language). Los procesos cliente emplean dicha interfaz para comunicarse entre ellos. Para realizar dicha comunicación se emplea un núcleo importante de la arquitectura que es el ORB (Object Request Broker) Process que es el encargado de gestionar y sincronizar todas las comunicaciones.

Como arquitectura, CORBA tiene una gran variedad de implementaciones ya que se ha estandarizado como modelo de objetos distribuidos. De entre las implementaciones destacar (por orden alfabético):

- Borland Visibroker [Visigenic, 1998] como sistema comercial lo inició la empresa Visigenic que posteriormente compró Borland. Existen diferentes versiones en función del lenguaje orientado a objetos que se desee emplear: “VisiBroker for Java”, “VisiBroker for C++” compatible por completo con ANSI C++, finalmente “VisiBroker Smart Agent” proporciona un sistema dinámico distribuido con servicio de nombres, directorios, soporte a tolerancia de fallos y ajuste de carga del sistema.
- JacORB. Es una plataforma CORBA implementada completamente en JAVA y de libre distribución. Es de especial interés ya que implementa las políticas de calidad de servicio de DDS, modelo que se verá más adelante. La compañía PrismTech da soporte comercial a la distribución.
- Orbix [Iona, 1997]. Es un producto comercial que implementa la especificación de CORBA, actualmente comercializado por la compañía IONA.
- TAO [Schmidt et al., 1998] es una implementación de CORBA con soporte a tiempo real. El nombre es el acrónimo de The Ace ORB. Es la evolución de ACE, proporcionando además un inicio de calidad de servicio entre componentes. Actualmente hay versiones para diversos sistemas operativos.

Cabe destacar que diversos componentes de CORBA se emplean habituamente como parte de otros entornos de desarrollo de sistemas distribuidos. Como ejemplo más paradigmático de esto está "OpenFusion" que aúna parte de la implementación TAO con el enriquecimiento de módulos de CORBA y la ampliación con sistemas de calidad de servicio o de tolerancia a fallos. OpenFusion lo gestiona PrismTeche, empresa dedicada también a comercializar el modelo DDS, que se verá a continuación.

3.2.9 DDS

Data Distribution Service for Real-time Systems (DDS) es una especificación de sistema de publicación y suscripción con soporte a tiempo real, que complementa a CORBA. La especificación corresponde a OMG (Object Management Group). Las primeras especificaciones provienen de 2003, y actualmente hay una versión de enero de 2007.

DDS se trata de un sistema de comunicaciones basado en la información más complejo que los anteriores. El paradigma que emplea es el de productores y consumidores (figura 9), aunque no se distinguen entre ellos, y todos están encapsulados por medio de módulos. Este encapsulado hace que los productores desconozcan donde se consume la información que envían al sistema, y de manera análoga, los consumidores desconocen dónde es producida la información. Aunque el origen físico de la información se desconozca, sí que se conoce el origen lógico de la misma, ya que los consumidores se registran (o suscriben) a un agente de comunicaciones que es el encargado de gestionar las mismas.

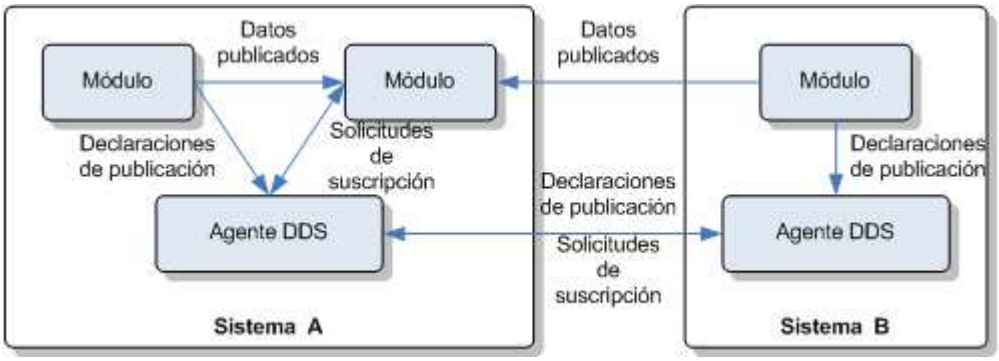

Figura 9. Arquitectura de comunicaciones DDS.

La fiabilidad de las comunicaciones es responsabilidad del agente DDS. Los cambios en la topología del sistema se pueden realizar cuando este está funcionando, lo que le confiere cierta potencia en comparación con los sistemas anteriores, donde la configuración del mismo era estática a lo largo de todo el proceso.

La especificación describe dos niveles de interfaces [Houston, 1998]. El más bajo, y obligatorio si se quiere cumplir con la especificación, es el DCPS (Data-Centric Publish-Subscribe), y tiene el objetivo de proporcionar la eficiencia en la entrega correcta de la información a los destinatarios. La segunda de las capas, es DLRL (Data Local Reconstruction Layer); es una capa opcional que tiene como objetivo integrar DDS en la capa de aplicación. De entre los componentes importantes de DDS cabe destacar los siguientes.

- DomainParticipantFactory. Es una instancia única del punto de entrada para las aplicaciones que empleen los servicios de DDS.

- DomainParticipant. Es el punto de entrada de las compunicaciones para un dominio concreto. Representa a la aplicación que emplea DDS dentro de un dominio específico y proporciona los componentes de DDS necesarios para la comunicación.
- Topic. Instaciación de un componente TopicDescription, que es la descripción básica de un dato que puede ser publicado y al que se pueden suscribir otros componentes.
- ContentFilteredTopic o MultiTopi. TopicDescription especializados.
- Publisher. Objeto responsable de la diseminación de datos cuando estos deben ser publicados.
- DataWriter. Objeto que permite a una aplicación instanciar un valor de un dato para que sea publicado por un Topic.
- Subscriber. Objeto responsible de la rececpción de los datos resultante de las suscripciones.
- DataReader. Objeto que permite a la aplicación declarar los datos en los que está interesada recibir información (creando una subscripción usando un Topic, ContentFilteredTopic o MultiTopic) y accede a los datos recibidos usando un Subscriber asociado.

Como arquitectura, DDS tiene diversas implementaciones, entre ellas destacan:

- NDDS [RTI, 1999] con origen en los sistemas de tiempo real [Castellote et al., 1997], siendo actualmente uno de los entornos de desarrollo de DDS más empleados.
- Proyecto Open DDS [Busch, 2007], en el que se ofrece una implementación de código abierto en C++ del modelo DDS. El soporte comercial lo ofrece la compañía Object Computing, Inc.
- OpenSplice [PrismTech, 2007]. Es la distribución del modelo DDS que realiza la empresa PrismTech. El sistema que ofrecen está muy probado y se vende como sistema de comunicaciones para misiones críticas.
- MilSOFT DDS [MilSOFT, 2006]. Desarrollado por la empresa del mismo nombre, es un sistema DDS con soporte para JAVA, a través de la capa JNI.

Se puede apreciar que DDS tiene una difusión muy similar a CORBA, eso se debe a que DDS es un enriquecimiento de CORBA, tal como destaca la OMG en su descripción del modelo de comunicaciones. La gran ventaja sobre CORBA está en la estandarización de las políticas de calidad de servicio y en la gran simplicidad del modelo DDS frente a CORBA.

3.2.10 MSMQ

MSMQ es una herramienta de Microsoft para realizar el manejo de colas de mensajes. La misma viene con Windows NT/2000 y cuenta con una interfaz que permite escribir programas en lenguaje C o utilizando componentes COM para realizar el envío y recepción de mensajes [Lewis, 1999]. MSMQ provee otras funcionalidades como ser la autenticación y el encriptado de mensajes, y el uso de transacciones externas, por ejemplo administradas por SQL Server, lo que facilita la implementación de las aplicaciones que utilizan la cola de mensajes como forma de realizar sus tareas. Además

se cuenta con la herramienta MSMQ Triggers que permite disparar procesos cada vez que un mensaje con ciertas características llega a la cola. La definición de los triggers permite disparar los procesos por ejemplo cuando la etiqueta o el cuerpo del mensaje contienen determinado texto; cuando la prioridad del mensaje es igual, mayor o menor a cierto valor; cuando el usuario que envía el mensaje es un usuario determinado. Los componentes y el esquema de funcionamiento pueden verse en la figura 10.

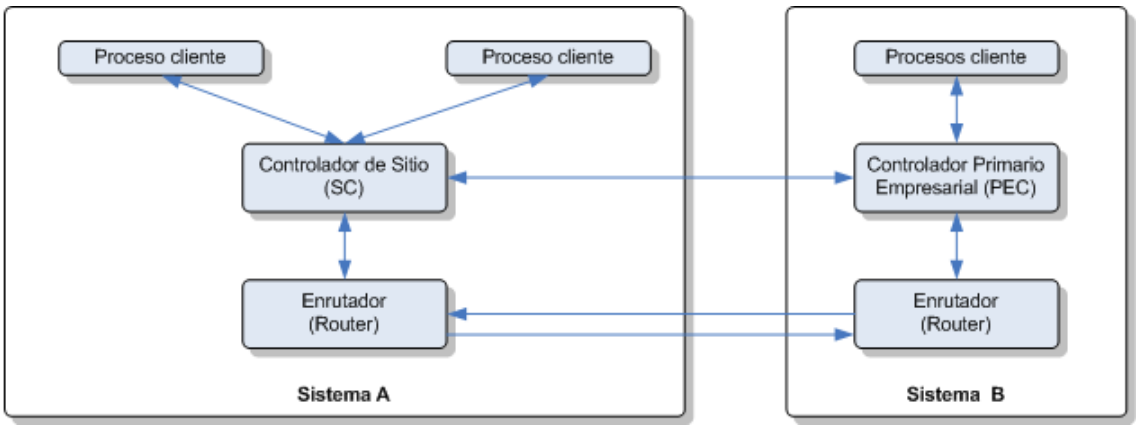

Figura 10. Arquitectura de comunicaciones MSMQ.

Para acceder a una cola en .NET existe una API llamada System.Messaging. Inicialmente está pensada para trabajar con MSMQ pero realmente la API es extensible para soportar otros sistemas basados en colas. System.Messaging proporciona clases que permiten conectar, supervisar y administrar las colas de mensajes en la red, así como enviar, recibir o leer mensajes. La clase principal de System.Messaging es MessageQueue. Los miembros de la clase MessageQueue incluyen diversos métodos para leer y escribir mensajes en la cola.

- Send. El método Send permite a la aplicación escribir mensajes en la cola. Las sobrecargas del método permiten enviar el mensaje con una clase Message (que proporciona un control exhaustivo sobre la información enviada) o cualquier otro.
- Receive. Los métodos Receive, ReceiveById, y ReceiveByCorrelationId proporcionan funciones para leer los mensajes de una cola. Al igual que el método Send, estos incluyen sobrecargas compatibles con el procesamiento de colas transaccionales. Dichos métodos ofrecen además sobrecargas con parámetros de tiempo de espera que permiten que el proceso continúe si la cola está vacía. Dado que dichos métodos son ejemplos de procesamiento sincrónico, estos interrumpen el subproceso actual hasta que haya un mensaje disponible, salvo que se especifique un tiempo de espera.
- Peek. El método Peek se asemeja a Receive, aunque no quita un mensaje de la cola una vez que se haya leído. Dado que Peek no modifica el contenido de la cola, existen dos sobrecargas que admiten procesamiento transaccional. Sin embargo, como Peek y Receive leen los mensajes de la cola sincrónicamente, las sobrecargas del método permiten que se especifique un tiempo de espera para impedir que el subproceso espere de forma indefinida.

Los métodos BeginPeek, EndPeek, BeginReceive y EndReceive ofrecen formas de leer los mensajes de la cola asincrónicamente. No interrumpen el subproceso actual mientras esperan a que llegue un mensaje a la cola.

Otros métodos de la clase MessageQueue proporcionan funciones para recuperar listas de colas según los criterios especificados y averiguar si determinadas colas existen. GetPrivateQueuesByMachine permite recuperar las colas privadas de un equipo. Los métodos derivados de Get, son GetPublicQueuesByCategory, GetPublicQueuesByLabel y GetPublicQueuesByMachine que ofrecen algunas formas de recuperar las colas públicas según unos criterios comunes. Una sobrecarga de GetPublicQueues ofrece mayor nivel de detalle para seleccionar las colas según una serie de criterios de búsqueda.

Finalmente, otros métodos de la clase MessageQueue son los que crean y eliminan colas de Message Queue Server, utilizan un enumerador de mensajes para desplazarse entre los mensajes de una cola, utilizan un enumerador de cola para iterar las colas del sistema, métodos para establecer derechos de acceso o para trabajar directamente con el contenido de las colas.

3.2.11 JMS

Java Message Service aparece con el objetivo de aislar de las interfaces y de la configuración del servicio de mensajería. En la especificación J2EE, JMS proporciona las interfaces para la comunicación por medio del servicio de mensajería JAVA [Hapner et al., 2002].

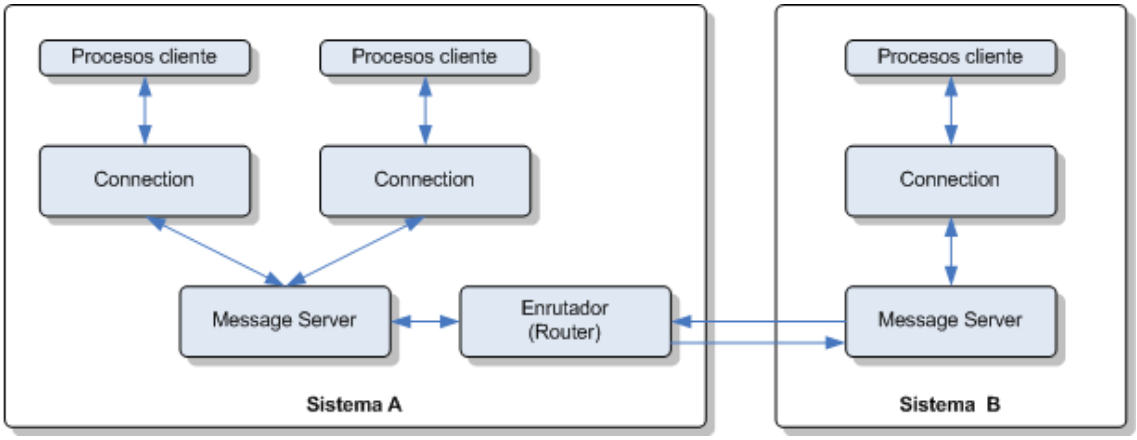

Figura 11. Arquitectura de comunicaciones JMS.

JMS Admite tanto mensajes punto a punto, como de publicación/suscripción (figura 11). Cuando un mensaje es sólo entre un emisor y un receptor, se emplea el dominio punto a punto. En este modelo, para asegurar que un mensaje llega al receptor, aunque éste no esté disponible, se encola el mensaje en una cola FIFO para ser recibido por el destinatario cuando éste esté accesible. Cuando el mensaje puede provenir de diferentes fuentes hacia diferentes destinos se emplea el dominio de publicación/suscripción. En el modelo de publicación/suscripción se tienen varios clientes que pueden publicar información (tópicos) o eventos y otros que se suscriben a esos tópicos. Ambos modelos pueden ser síncronos mediante el método receive() y asíncronos por medio de un MessageListener que es el responsable de escuchar los posibles cambios en los tópicos a los que se suscriba un cliente. Las aplicaciones JMS constan de tres elementos principales.

- Objetos administrados: Objetos JMS que gestionan la conexión y los mensajes.
- Mensajes: información que intercambian los clientes JMS.

- Clientes JMS. Son las aplicaciones que envían o reciben mensajes empleando JMS.

Los objetos administrados son los elementos que emplea JMS para enviar y recibir mensajes a través de JMS. Estos objetos son los que deben implementar las interfaces JMS y están situados en JNDI (Java Naming and Directory Interface) para que cualquier cliente pueda solicitar el uso de cualquier objeto administrado. Hay dos tipos de objetos administrados.

- ConnectionFactory: Se emplea para crear una conexión al proveedor del sistema de mensajes.
- Destination: Son los destinos de los mensajes que se envían y a su vez el contenedor de los mensajes que se reciben.

Los mensajes están formados por tres elementos: Header (cabecera), Properties (propiedades) y body (cuerpo del mensaje). La cabecera contiene una serie de campos que permiten a los proveedores y a los clientes identificar las características generales de los mensajes con los que trabajan. Las propiedades un conjunto de características de cada mensaje en particular. Finalmente el cuerpo del mensaje es el contenedor de la información que se desea intercambiar. El cuerpo puede ser de diferentes tipos.

- StreamMessage: Contiene una cadena secuencial de datos que la aplicación debe conocer.
- MapMessage: Contiene campos con pares nombre-valor.
- TextMessage: Contiene una cadena de caracteres.
- ObjectMessage: Contiene un objeto que implemente la interfaz serializable.
- BytesMessage: Contiene una cadena de bytes.

Se considera clientes en JMS tanto a los componentes que suministran los mensajes, como a los que los reciben. Para poder comunicarse todos deben seguir unos pasos muy concretos.

1. Solicitar, a través de JNDI, un objeto del tipo “ConnectionFactory”.
2. Conseguir, por medio de JNDI, un destino a través de un objeto “Destination”.
3. Usar el “ConnectionFactory” para conseguir un objeto “Connection”.
4. Usar el objeto “Destination” para crear un objeto “Session”.

Una vez creados los objetos, se pasa al intercambio de mensajes. Como se trata de intercambio de información entre objetos, ya no hay una aplicación que envía y otra que recibe, sino que son varias las aplicaciones que reciben los mensajes, en concreto aquellas que están suscritas a la cola a la que se envían los mensajes desde un objeto específico.

JMS está ampliamente extendido ya que, como sistema muy ligado a JAVA es de gran interés cuando se busca la compatibilidad. Existen gran cantidad de productos que soportan JMS, pero entre ellos cabe destacar Active MQ de “Apache” por la importancia de la compañía y la amplia implantación como servidor. JMS se emplea como sistema de mensajería de soporte al sistema de colas. Sun Microsystems tiene su propia implementación Sun Java System Message Queue (SJSMQ).

3.2.12 MidArt

MidArt [Mizunuma et al., 1996] es el acrónimo de "Middleware and network Architecture for distributed Real-Time systems". MidArte trata de simplificar la complejidad inherente en CORBA, aislando a las aplicaciones de los detalles de la red y proporcionando transacciones en tiempo real. Para ello organiza el sistema en tres capas con funciones específicas. Cada capa se compone de diversos módulos.

- Capa de gestión. Es la capa responsable de las interacciones entre los programas que se ejecutan en diferentes nodos, y que no precisan de soporte de tiempo real en sus transacciones. Se compone de un único módulo, Global Manager (GM), que a su vez tiene dos componentes: el Global Server (GS) y el Global Connection Admission Control (GCAC).
- Capa de aplicación. Tiene un único componente, el Application Programming Interface (API), es una librería que permite a las aplicaciones invocar los servicios de MidArte.
- Capa de intermediación o "middleware". Tiene un único componente, el Local Server Process (LSP), que se compone de diversos módulos: Local Sever (LS), Real-Time Channel based Reactive Memory (RT-CRM), Selective Channels (SC), Scheduler, Fault Tolerance y Local Connection Admission Control (LCAC).

MidArt se caracteriza por la simplificación y el reparto de las tareas de la comunicación de forma organizada y bien definida en módulos, componentes y capas. Las librerías de desarrollo están orientadas a objetos y los componentes que implementan las capas son objetos instanciados a partir de las clases que proporciona la librería. Por ello puede considerarse que el sistema es orientado a objetos, aunque distribuya mensajes entre ellos.

3.2.13 Jini

Jini no son unas iniciales, aunque al estar relacionado con JAVA suele encontrarse alguna interpretación de las siglas. Jini es una extensión de JAVA cuyo objetivo es facilitar la integración de sistemas distribuidos. Jini se basa en el principio del ofrecimiento y uso de servicios en un sistema distribuido, entendiéndose como servicio desde un hardware hasta una aplicación concreta, pasando por el concepto clásico de servicio. Los servicios se agrupan en entidades llamadas "federaciones", que realmente son federaciones de máquinas virtuales de JAVA.

Los orígenes de JINI son similares a los de JAVA, a partir del año 1994 comienzan a ser necesarias las transacciones entre procesos con localizaciones separadas en la red. Los conceptos básicos y componentes que se encargan de proveer dichas características son los siguientes.

- Servicios. Es el concepto básico de JINI, prácticamente todo se considera un servicio: dispositivos, datos, almacenamiento, procesamiento de datos como filtrado o los cálculos realizados con los datos. En definitiva, todo aquello que pueda ser útil para un usuario u otros servicios.
- Servicio Lookup. Método por el que se localizan los servicios disponibles en una comunidad JINI. Es el punto de inicio y de interacción entre un servicio y un usuario de dicho servicio.

- RMI: Remote Method Invocation. RMI es el RPC de JAVA que trabaja a nivel de objetos.
- Seguridad. Se basa en un modelo de lista de control que valida, permite accesos y supervisa las acciones de los servicios.
- Asignaciones de tiempos, o métodos por los que se asigna el tiempo a los recursos de los servicios y a los servicios. En el modelo de JINI se permite la exclusividad en el uso de un recurso durante un periodo de tiempo.
- Transacciones Son protocolos para el envío de mensajes entre servicios. Las operaciones entre uno varios servicios se pueden aunar en una sola transacción.
- Eventos Los eventos, no podían faltar en un sistema así. En este caso además, el sistema debe ser capaz de soportar eventos distribuidos. Si un nuevo dispositivo se une a la federación Jini, si un servicio desea notificar algo a un usuario,.. en muchas circunstancias los la notificación de eventos remotos resulta indispensable.

El hecho de trabajar con servicios, implica que los principales componentes de un sistema basado en JINI son el servidor, el cliente y las interacciones entre ellos. Uno de los puntos fuertes en los que se basa ésta tecnología es en la ligereza del código que implementa el núcleo de JINI, es del orden de 40KB en la versión actual. Emplear un núcleo muy sencillo tiene la ventaja de poder ser empotrado en casi cualquier dispositivo, desde grandes computadores a pequeños procesadores con capacidad muy limitada.

3.2.14 DCE

Distributed Computing Environment (DCE), OSF DCE es un conjunto de servicios que puede utilizarse separadamente o en combinación, en ambientes de desarrollo de computación distribuida. Los servicios son los siguientes:

- Servicio de llamada a procedimiento remoto o DCE Remote Procedure Call (DCE RPC).
- Servicio de Directorios o DCE Directory Service, que permite a usuarios y aplicaciones almacenar, retener y manipular información.
- Servicio de seguridad o DCE Security Service
- Servicio de manejo de hilos o DCE Threads.
- Servicio de tiempo o DCE Distributed Time.
- Sistema de archivos distribuido o DCE Distributed File System (DCE DFS).
- Servicio de soporte para computadoras sin disco local o DCE Diskless Support Service.
- Servicio de integración a computadoras personales.

DCE RPC es utilizado como único mecanismo de comunicación entre el cliente y el servidor en otros servicios que componen DCE que no pertenezcan al modelo. Actualmente DCE es distribuida bajo la licencia LGPL por la organización The Open Group. También existen una serie de distribuciones comerciales de DCE de IBM o HP.

3.2.15El caso Microsoft: DDE, OLE, COM, DCOM y .NET

Un ejemplo de la evolución de los sistemas de comunicaciones dentro de la misma empresa es el caso de Microsoft, donde se ha ido evolucionando desde un modelo sin soporte a la distribución basado en el paso de mensajes hasta un modelo de distribución de componentes. En la figura 12, se puede ver cómo cada modelo ha ido progresando, en paralelo con los intentos de estandarización en los que Microsoft participaba.

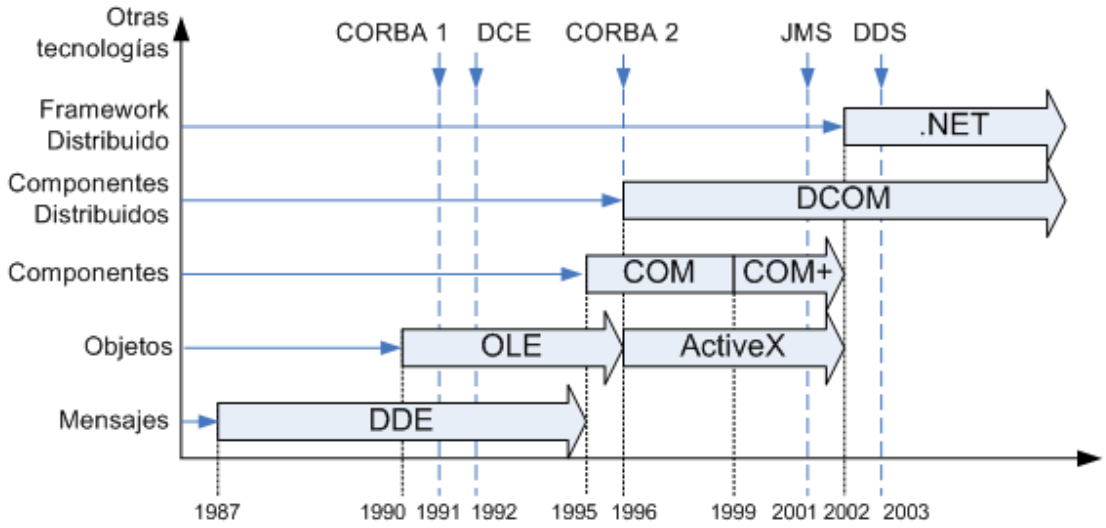

Figura 12. Evolución comparativa de las arquitecturas de Microsoft.

El sistema inicial de comunicaciones del que parten los sistemas operativos de Microsoft es el DDE (Dynamic Data Exchange) consistente en el paso de mensajes entre aplicaciones en forma de mensajes, lo que implica el mantenimiento de un servidor DDE. Algunas aplicaciones continúan empleándolo, aunque normalmente ya no se emplea ya que cualquiera de las tecnologías posteriores proporciona mejores prestaciones. Para el soporte de sistemas distribuidos hay una versión llamada NetDDE que no es más que el uso de comandos DDE sobre las tecnologías posteriores

El siguiente paso en la comunicación de procesos es la tecnología OLE (Object Linking and Embedding), los cambios que incluye con respecto a DDE son el uso de la tecnología de objetos, la posibilidad de mantener dinámicamente los enlaces entre aplicaciones y el uso del concepto de "incrustar" objetos dentro de otra aplicación con la posibilidad de intercambiar información con el origen de forma dinámica. En 1996 la versión 2 de OLE pasa a llamarse ActiveX orientando la tecnología al ámbito concreto de la Web.

La evolución de OLE, aunque paralela a ActiveX, pasó a llamarse COM (Component Object Model) en la que se ofrece una interfaz independiente del lenguaje de programación para la comunicación de diferentes objetos. La independencia del modelo se logra por medio de una interfaz común y de un conjunto de criterios comunes a la hora de programar componentes COM. La extensión de COM a un sistema distribuido, de forma oficial, aunque oficiosamente ya existía con ActiveX, se llamó DCOM (Distributed Component Object Model). DCOM añade características de DCE al modelo COM en cuanto a la serialización de información y recolección de basura en el sistema distribuido. Cabe destacar que DCOM no es en sí mismo una tecnología de Microsoft, sino un modelo que también han desarrollado otras compañías, por ejemplo Open Group tiene una implementación llamada COMsource; Microsoft cumple con el

modelo sin cambiarle el nombre, lo que hace que formalmente sea recomendable llamarlo Microsoft DCOM.

Actualmente Microsoft tiene como modelo de sistema distribuido el .NET. Esto ha añadido algo de confusión ya que también denomina por ese nombre el "Framework" para el funcionamiento de la mayor parte de las aplicaciones desarrolladas para Windows XP y posteriores. El impulso que se le da al lenguaje C# ha hecho que también se llame .NET a dicho lenguaje. En cuanto a modelos de comunicación distribuida, .NET no es más que la evolución de DCOM y ActiveX uniendo a todo el sistema y añadiendo los servicios Web al modelo. Además emplea XML como norma en el formato de los mensajes, lo que supone un gran avance hacia los sistemas abiertos con semántica autocontenida. Formalmente no se puede considerar .NET como un modelo de comunicaciones ya que es algo mucho más extenso.

3.2.16 AMPQ

Actualmente la tendencia es ofrecer un middleware orientado a transacciones de mensajes entre servicios, a éste tipo de middleware también se les conoce como Message Oriented Middleware (MOM). Cabe destacar un estándar de facto llamado Advanced Message Queuing Protocol (AMQP), iniciado en 2004 y empleado en diversos sistemas como OpenAMQ, Rabbit MQ Apache Qpid o Red Hat Enterprise MRG.

El funcionamiento de AMPQ, a grandes rasgos, puede verse en la figura 13. Se puede apreciar cómo no existe gestión de publicación-subscripción sino gestión de mensajes entre aplicaciones las aplicaciones que los publican y las que los reciben.

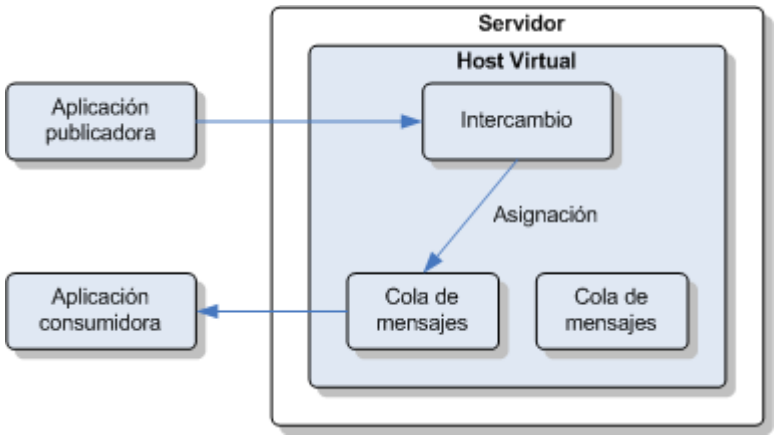

Figura 13. Esquema básico de AMPQ.

AMPQ se diferencia de los sistemas de publicación y subscripción en la ubicación de las colas de mensajes, ya que emplea elementos específicos para la gestión de dichas colas. Hay una excelente comparación de AMPQ con DDS en [Pardo-Castellote, 2007].

4 Conclusiones

4.1 Una visión global

Todos los paradigmas anteriores pertenecen a un conjunto cuyos componentes son no excluyentes ya que en muchos casos se emplean de manera conjunta en una arquitectura. Dependiendo del sistema que se deba implementar, un modelo podrá ser válido para lograr unos objetivos y un cambio de los objetivos puede hacer válido otro modelo. En todos los paradigmas existe un objetivo común de ocultar, o facilitar, al usuario los detalles de las comunicaciones. Sin embargo, a medida que los sistemas se hacen más complejos, además de ocultarse los detalles del mismo, se adaptan a los requerimientos de programación del sistema.

Muchas aplicaciones, especialmente las aplicaciones de tiempo real están centradas en los datos, o lo que es lo mismo, precisan recibir datos de diferentes fuentes, o distribuirlos a diferentes destinos. Por tanto las arquitecturas de comunicaciones de los sistemas de control emplearán arquitecturas orientadas a distribuir datos, no invocar servicios remotamente como cliente/servidor o llamadas de funciones como RPC.

De entre los modelos de comunicaciones, uno de los métodos en los que se basan las comunicaciones, de manera más frecuente, es el uso de mensajes, también conocidos en ocasiones como MOM (Message Oriented Middleware). Los servicios de mensajería de capacitan una comunicación asíncrona entre componentes de un sistema. Los servicios de mensajería ocultan el origen del mensaje, por lo que sirven tanto para intercambiar mensajes entre componentes de un mismo servidor como los mensajes que provienen de otros servidores en un sistema distribuido.

Para que un servicio de mensajería pueda emplearse en un sistema distribuido, los nodos que se comunican deben estar programados para reconocer un formato de mensaje común. El contenido del mensaje puede ser variable; existen mensajes orientados a campos, tanto de longitud fija, como variable o indexado, mapas de información con pares de nombre valor, o incluso texto.

Un paso más de complejidad, pero también de potencia supone el empleo de mensajes cuyo contenido es un objeto serializado. Actualmente se están desarrollando sistemas donde el contenido de los mensajes es código, con lo que los programas que se ejecutan en los servidores varían, no solo en la información con la que trabajan, sino en los algoritmos y conocimientos con los que procesan dicha información.

Las aplicaciones que emplean los servicios de mensajería para intercambiar información no precisan un acoplamiento sólido, es decir el emisor y el receptor pueden cambiarse o sustituirse sin que ello afecte a las aplicaciones. Cuanto más se aíslen las aplicaciones de la aplicación del servicio de mensajes, mayor capacidad de acoplamiento tendrán, ya que no impondrán restricciones a las aplicaciones, que potencialmente puedan usar el servicio.

A diferencia del modelo de comunicación de intercambio de mensajes, los sistemas de mensajería no bloquean al emisor cuando éste envía el mensaje, y se queda a la espera de una respuesta. Además, no es necesario que el receptor permanezca desocupado cuando espera la llegada del próximo mensaje. El servicio de mensajes se encarga de organizar y gestionar los mensajes en las colas hasta que se puedan aceptar.

La gestión de la entrega y de la coherencia del mensaje también es responsabilidad del servicio de mensajería, lo que descarga de muchas responsabilidades a las aplicaciones que se comunican a través de dicho servicio y permiten que éstas puedan simplificarse.

4.2 Contextualización temporal de los modelos de comunicación

En la figura 14, se pueden observar las arquitecturas anteriores en función del año de aparición de la tecnología y del modelo principal de comunicaciones que emplean. De los sistemas anteriores se puede observar cómo la evolución puede ser dentro de un mismo paradigma, como hacen NML, IPC y RTC que se basan todos ellos en TCX [Gowdy, 2000], o cambiar la tecnología aunque la funcionalidad sea similar, como el caso de Microsoft con DDE, OLE, COM y DCOM. En otras tecnologías se ha preferido no cambiarla y mantener la funcionalidad, es el caso de CORBA y DDS, ambos propuestos por la OMG.

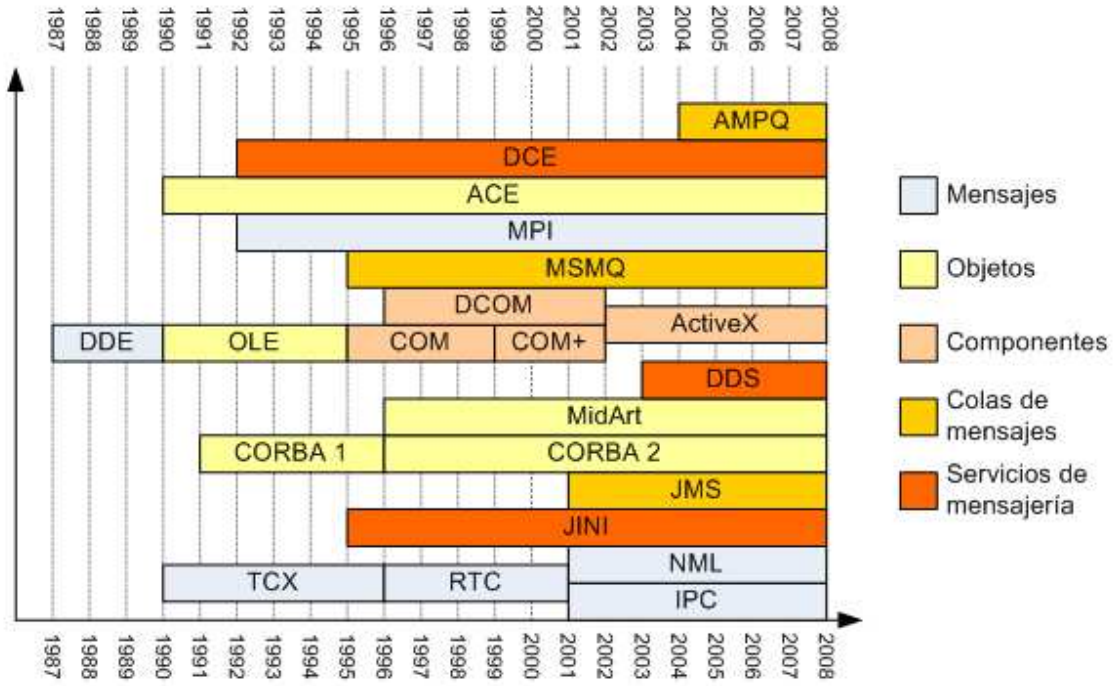

Figura 14. Ubicación diferentes tecnologías en función del tipo de comunicación que implementan.

Se puede ver cómo los modelos de comunicación están muy relacionados con los modelos de programación existentes en el momento. Hasta la generalización de la programación orientada a objetos, el modelo empleado se basaba en librerías que proporcionaban funciones de comunicación basadas en mensajería. Son los casos de TCX, DDE o MPI. Además, el modelo de cliente-servidor hacía que los servidores de mensajería también tuviesen aparición como modelos iniciales. Son los casos de DCE, y de JINI que aparece como servicio de mensajería de JAVA, prácticamente en el mismo momento de aparición del lenguaje.

Con la generalización de la programación orientada a objetos, aparecen modelos de comunicación basados en dicho paradigma, como ACE o el ampliamente utilizado CORBA y sus variaciones entre las que destaca MidArt como ejemplo de soporte a tiempo real. La programación basada en componentes tiene su sistema de comunicaciones paradigmático con COM y DCOM de Microsoft.

La combinación de los servicios de mensajería y objetos da lugar al interesante modelo DDS donde se busca simplificar la programación de las comunicaciones, ofreciendo la misma potencia que los anteriores modelos.

La evolución a un sistema middleware, especialmente como aplicaciones, da lugar a los modelos de colas de mensajes, como JMS, MSMQ o el modelo más reciente AMPQ.

4.3 Características de los modelos

Los modelos vistos anteriormente cubren prácticamente todas las características deseables en un sistema de comunicaciones que implemente un control inteligente. De entre ellas cabe destacar que algunos de ellos permiten implementar diferentes paradigmas de comunicaciones y pasar de un modelo cliente-servidor a un modelo Publicación-subscripción sin problemas.

Todos los modelos, evidentemente, se sustentan en un paso de mensajes. Sin embargo, algunos de ellos ocultan los detalles de la gestión del paso de mensajes, ésta característica es deseable, especialmente si lo que se desea es implementar un middleware o emplear las características de ocultación de la comunicación en el caso de la computación distribuida. La gestión de las colas de mensajes es otro de los aspectos más interesantes a destacar de los modelos anteriores. Todos usan colas para el envío de los mensajes, y al igual que los detalles del envío o recepción de los mensajes en las colas, es deseable que los usuarios no deban ocuparse de la gestión de éstos aspectos.

Sin embargo, en muchas ocasiones, sí que se debe actuar sobre el envío de mensajes, especialmente cuando se desea un comportamiento concreto de los aspectos temporales o de flujo de mensajes. En estos casos, lo más habitual es que se ofrezcan características concretas de control de flujo o de tiempo real.

En la figura 15, se puede observar la ubicación de los principales modelos vistos anteriormente en función del soporte que ofrecen al tiempo real. Se puede apreciar cómo los sistemas de mensajería directa, que no implementan un paradigma de forma implícita, son los que ofrecen mejor soporte al tiempo real, mientras que los sistemas basados en servicios o colas de mensajes ofrecen un control de tiempo básico.

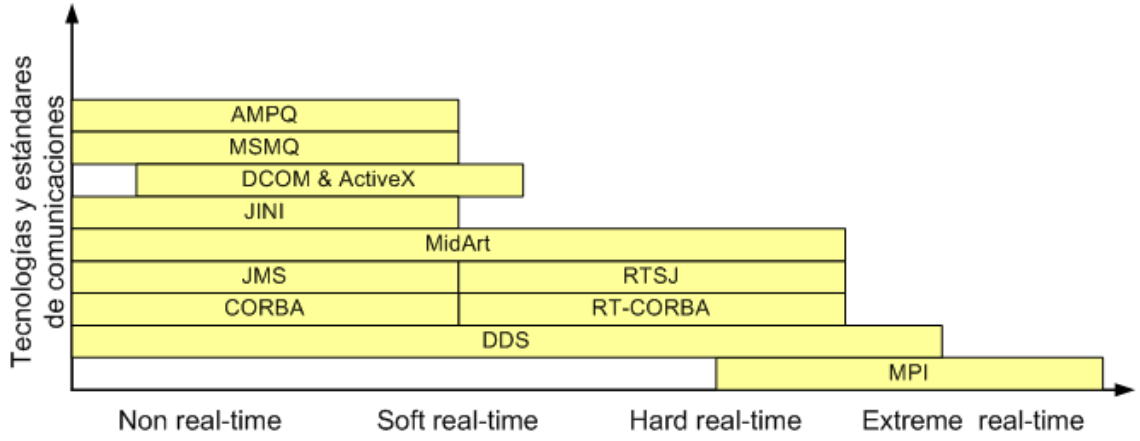

Figura 15. Ubicación diferentes tecnologías en función de los aspectos de tiempo real que cubren.

Cabe destacar que para los modelos JMS o CORBA, existen implementaciones con control temporal y de flujo. Estos modelos están desarrollados directamente para cubrir las carencias en éstos aspectos que tienen los modelos iniciales, pero en ocasiones se considera más un enriquecimiento adicional del modelo que una característica del

mismo. Sin embargo, hay algunas evoluciones como MidArt en el modelo de objetos o DDS en el modelo de mensajería que sí implementan un control de tiempo real estricto como característica destacable.

Resumiendo las características anteriores de los sistemas, se puede decir entre las características más deseables de un sistema de comunicaciones están las siguientes.

- Implementación de un paradigma de comunicación o soporte a varios paradigmas. Para los componentes que emplean el sistema de comunicaciones, es interesante poder utilizar un modelo cliente-servidor, como un "peer to peer".
- Gestión de las comunicaciones síncronas y asíncronas. Los componentes que empleen el sistema de comunicaciones deben poder comunicarse de forma asíncrona, o lo que es lo mismo a iniciativa del componente, o de forma síncrona a iniciativa del sistema de comunicaciones.
- Agrupamiento de los componentes de comunicación o soporte a la organización de los mismos, en función de características deseables del usuario y no de las características de los componentes.
- Configuración dinámica del sistema. Es importante que el sistema de comunicaciones permita variar la mayor cantidad de características posibles, sin necesidad de detener la actividad de los componentes que lo emplean: portabilidad y escalabilidad.
- Descubrimiento del sistema. El sistema de comunicaciones debe ofrecer la posibilidad de informar de los cambios de configuración a los componentes que lo emplean, o informar de la configuración a los nuevos componentes.
- Control temporal o soporte a tiempo real. Los componentes deben conocer algunos aspectos como el tiempo que tarda un mensaje en llegar, o poder solicitar al sistema de comunicaciones la entrega en un periodo concreto.
- Sencillez en la interfaz de comunicaciones, en componentes y en protocolos empleados. En algunos modelos de comunicación, la complejidad de éstos aspectos ha provocado que su uso no sea tan extendido pese a ser unos modelos muy válidos.
- Gestión de la calidad de servicio. El aspecto del comportamiento del sistema en función de unos parámetros es importante puesto que permite sintonizar las comunicaciones con los requerimientos de los componentes.

Finalmente la gestión de dichas características puede hacerse de diversas formas: por medio de la configuración del modelo de comunicaciones empleado, por medio de características de cada componente, pero destaca especialmente la configuración de las comunicaciones por medio de la configuración del comportamiento en lugar de la definición de unas características específicas.

Para la configuración del comportamiento del sistema de comunicaciones en función de los requerimientos de los usuarios aparece el concepto de calidad de servicio, o QoS (Quality of Service) por sus siglas en inglés. A continuación se analizará el papel que la calidad de servicio toma en la configuración de las comunicaciones para poder reconocer qué características son relevantes en la definición de parámetros de comportamiento del sistema de comunicaciones.

4.4 Posibles áreas de investigación

Todos los paradigmas de comunicaciones tienen sistemas que los implementan y además tienen vigencia actual, es decir, la aparición del paradigma de servicio de mensajería no excluye la posibilidad de usar el paradigma del paso de mensajes clásico. Eso hace que la investigación en los sistemas de comunicaciones no siga una línea unidimensional hacia una tendencia, sino que abarque un amplio rango.

Otro de los aspectos en los que es posible desarrollar líneas de investigación es el centrado en el control temporal de las mismas. Eso implica tanto el desarrollo de sistemas con soporte a un control básico de retardos en mensajes como el soporte a l cumplimiento de plazos de tiempo real estricto.

Dada la importancia del tema, la calidad de servicio se ha dejado para posteriores documentos, sin embargo es posiblemente una de las áreas más interesantes en cuanto a líneas de investigación ya que implica todos los aspectos mencionados anteriormente. En posteriores estudios se desarrollarán estas cuestiones con mayor detalle.

5 Referencias

[Busch, 2007] Don Busch. Introduction to OpenDDS. Object Computing, Inc. (OCI) 2007.

[Coulouris et al., 2001] Coulouris, G., Dollimore, J., Kindberg, T. Sistemas Distribuidos, Conceptos y diseño. Tercera Edición. Addison Wesley. Madrid. 2001.

[Fedor, 1994] Fedor, C. TCX: an interprocess communication system for building robotic architectures. Technical report, Robotics Institute, Carnegie Mellon University, 1994.

[FIPA, 1997] Foundation for Intelligent Physical Agents: FIPA 97 Specfication. Part 2, Agent Communication Language", (1997).

[Fong et al., 2001] Fong, T.W. C. Thorpe, and C. Baur. A safeguarded teleoperation controller. In IEEE International Conference on Advanced Robotics 2001, Budapest, Hungary, August 2001

[Fong, 1998] Fong, T.W. Fourth planet communicator. Fourth Planet Inc., 1998.

[Gaddah and Kunz, 2003] Gaddah A., and Kunz, T. A survey of middleware paradigms for mobile computing. Technical Report SCE-03-16. Carleton University Systems and Computing Engineering. (2003)

[Gamma et al, 1994] Gamma, e., Helm, R., Johnson, R., Vlissides, J. Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley, 1994

[Gowdy, 1996] Gowdy, J. IPT: An Object Oriented Toolkit for Interprocess Communication. Technical report CMU-RI-TR-96-07, Robotics Institute, Carnegie Mellon University, March, 1996.

[Gowdy, 2000] Gowdy, J. A Qualitative Comparison of Interprocess Communications Toolkits for Robotics. Tech. report CMU-RI-TR-00-16, Robotics Institute, Carnegie Mellon University, June, 2000

[Hapner et al., 2002] M. Hapner, R. Sharma, J. Fialli, and K. Stout, JMS specification, Sun Microsystems Inc., 4150 Network Circle, Santa Clara, CA 95054 USA, 1.1 edition, April 2002, http://java.sun.com/products/jms/docs.html

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

[Iona, 1997] IONA. Orbix 2.2 Programming guide. IONA Technologies Ltd. Mar. 1997.

[Lewis, 1999] Lewis, R.: Advanced Messaging Applications with MSMQ and MQ Series. Que Publishing, 1999.

[Liu, 2004] Liu, M.L., Distributed Computing: Principles and Applications. Addison-Wesley, 2004.

[MilSOFT, 2006] MilSOFT, (2006), "MilSOFT DDS Middleware", http://dds.milsoft.com.tr

[Mizunuma et al., 1996] Mizunuma, I., Shen, C., Takegaki, M. 1996. 'Middleware for distributed industrial real-time systems on ATM networks', 17th IEEE Real-Time Systems Symposium.

[OMG, 1991] Object Management Group. The Common Object Request Broker: Architecture and Specification. 1.1 ed., December 1991.

[OMG, 1995] Object Management Group, The Common Object Request Broker: Architecture and Specification. 2.0 ed., July 1995.

[Pardo-Castellote, 2007] Pardo-Castellote, G. Analysis of the Advanced Message Queuing Protocol (AMQP) and comparison with the Real-Time Publish Subscribe Protocol (DDS-RTPS Interoperability Protocol). Real-Time Innovations, Inc. 2007.

[Pedersen, 1998] Pedersen, J.D. Robust communications for high bandwidth real-time systems. Technical Report CMU-RI-TR-98-13, Carnegie Mellon University, Robotics Institute, May 1998.

[Poza et al., 2009] J.L. Poza, J.L. Posadas y J.E. Simó. Arquitecturas de control distribuido. Technical Report. Universidad Politécnica de Valencia. 2009.

[PrismTech, 2007] PrismTech Ltd. (2007) www.prismtech.com

[RTI, 1999] Real-Time Innovations. NDDS: The Real-Time Publish-Subscribe Middleware (1999). www.rti.com

[Schmidt et al., 1998] D. Schmidt, D. Levine, and S. Mungee. The Design and Performance of Real-time Object Request Brokers. Computer Communications, 21(4):294–324, April 1998.

[Schmidt, 1993] Schmidt, Douglas C. The ADAPTIVE Communication Environment An Object-Oriented Network Programming Toolkit for Developing Communication Software. 1993. 11thth and 12th th th Sun user group conferences. California

[Shackleford et al., 2000] W. Shackleford J. Michaloski, F. Proctor. The neutral message language: A model and method for message passing in heterogeneous environments. In Proceedings of the World Automation Conference, Maui, Hawaii, June 2000.

[Simmons and James, 2001] R. Simmons y D. James. Inter Process Communication: A Reference Manual, February 2001.

[Visigenic, 1998] Visigenic. Visibroker for C++ 3.2 Programmer's guide. Visigenic Software Inc. Mar. 1998.

6 ANEXO: Sistemas de comunicaciones analizados

Sistema	Significado	Paradigma (Modelos)	Compañía	Año	Web
.NET	N/S	Distributed Components	Microsoft	2002	http://msdn.microsoft.com/netframework/
ACE	ADAPTIVE Communication Environmet	Distributed Objects	Object Computing Inc	1993	http://www.cs.wustl.edu/~schmidt/ACE.html
ActiveMQ	Active Message Ques	Message Oriented Middleware	Apache	2006	www.activemq.org
Borland Visibroker	N/S	Distributed Objects	Borland	2004	www.borland.com
CICS	Customer Information Control System	Transaction Oriented Middleware	IBM	1980	www-306.ibm.com/software/htp/cics/
COM	Component Object Model	No es distribuido	Microsoft	1988	www.microsoft.com/com
CORBA	Common Object Request Broker Architecture	Distributed Objects	OMG	1995	www.corba.org
DCDT	Device Communities Development Toolkit	Publish/Subscribe	Univ. Brescia	2001	No localizada
DCE	Distributed Computing Environment	Client/Server	OSF	1993	http://www.opengroup.org/dce/
DCOM	Distributed Component Object Model	Distributed Components	Microsoft	1995	www.microsoft.com/com
DDS	Data Distribution Service	Publish/Subscribe	OMG	2004	portals.omg.org/dds
DSOM	Distributed System Object Model	Distributed Objects	IBM	1990	www.ibm.com
ETHNOS	Expert Tribe in a Hybrid Network Operating System	Publish/Subscribe	Univ. Genova	1999	No localizada
IPC	Inter Process Communication	Publish/Subscribe	Univ. Carnegie Mellon	1991	www.cs.cmu.edu/~IPC
IPT	Inter Process Toolkit	Publish/Subscribe	Univ. Carnegie Mellon	1996	No localizada
Jacorb	Java and CORBA	Publish/Subscribe	Varios (free)	1997	www.jacorb.org
JINI	N/S	Servicios	SUN Microsystems	1999	www.jini.org
JMS	Java Message Service	Message Oriented Middleware	SUN Microsystems	2001	java.sun.com/products/jms/
MidArt	Middleware and Network Architecture for Distributed Real Time Systems	Distributed Objects	Mitshubishi y otros		
MPI	Message Passing Interface	Paso de mensajes	MPI Forum	1994	www.mpi-forum.org
MSMQ	Message Queue	Paso de mensajes	Microsoft	1997	www.microsoft.com/windowsserver2003/technologies/msmq
NDDS	RTI Data Distribution Service	Publish/Subscribe	RTI	2001	www.rti.com
NDDS	Network Data Delivery Service	Publish/Subscribe	Univ. Stanford	1994	arl.stanford.edu/users/pardo/ndds.html
NML	Neutral Messaging Language	Message passing	NIST	2000	No localizada
OpenCore	N/S	Message passing	OpenSubsystems	2004	www.opensubsystems.org/core/
OpenDDS	N/S	Publish/Subscribe	Object Computing Inc	2006	www.opendds.org
OpenFusion	N/S	Distributed Objects	PrismTech	2000	www.prismtechnologies.com

OpenSplice	N/S	Publish/Subscribe	PrismTech	2006	www.prismtechnologies.com
Orbix		Distributed Objects			
ORiN	Open Resource interface for the Network/Open Robot interface for the Network	Message passing	ORIN Forum	1998	www.orin.jp
RTC	Real-Time Communications	Publish/Subscribe	Univ. Carnegie Mellon	1998	No localizada
SJSMQ	Sun Java System Message Queue	Message Oriented Middleware	SUN Microsystems	2005	www.sun.com/software/products/message_queue/
SOM	System Object Model	No es distribuido	IBM	1990	www.ibm.com
SPLICE	Subscription Paradigm for the Logical Interconnection of Concurrent Engines	Publish/Subscribe	US Navy	1993	No localizada
TAO	The ACE ORB	Publish/Subscribe	Object Computing Inc	2000	http://www.theaceorb.com/
TCX	N/S	Publish/Subscribe	Univ. Carnegie Mellon	1990	No localizada
TelRip	TeleRobotics Interconnection Protocol	Publish/Subscribe	Univ. Rice	1992	No localizada
Tibco	N/S	Service Oriented Middleware	TIBCO	1997	www.tibco.com
Tuxedo	Transactions for Unix, Extended for Distributed Operations	Transaction Oriented Middleware	BEA Systems	1984	www.beasys.es
UPnP	Universal Plug&Play	Message passing	Microsoft	1999	www.upnp.org
WebSphere MQ	N/S	Message passing	IBM	1992	www.ibm.com/webspheremq/