XXXVIII Jornadas de Automática
https://doi.org/10.17979/spudc.9788497497749.0921

DDS en el desarrollo de sistemas distribuidos heterogéneos con soporte para criticidad mixta

Héctor Pérez Tijero y J. Javier Gutiérrez

Grupo de Ingeniería Software y Tiempo Real, Universidad de Cantabria, 39005-Santander, SPAIN
{perezh, gutierjj}@unican.es

Resumen

El uso de middleware de distribución facilita la programación de sistemas distribuidos heterogéneos, y por extensión también puede facilitar la generación automática de código como parte de una estrategia de desarrollo basada en modelos. Sin embargo, este middleware presenta una complejidad añadida que dificulta su uso en sistemas con ciertos requisitos de criticidad o de tiempo real. En este trabajo se hace una revisión de algunos estudios previos en los que se muestra la posibilidad de utilizar un middleware de distribución centrado en los datos (DDS, Data Distribution Service) para la integración de aplicaciones con criticidad mixta en sistemas distribuidos.

Palabras clave: Tiempo real, DDS, sistemas particionados, hipervisor, flujo e2e

1. Introducción

Durante las últimas décadas, los sistemas industriales de tiempo real han pasado de ejecutar aplicaciones cerradas en sistemas empotrados y de propósito muy específico a formar parte de entornos de ejecución potentes, abiertos e interconectados. Sin embargo, el incremento de potencia ha dejado de estar asociado a la velocidad de los procesadores, y en la actualidad está ligado a la presencia de un mayor número de *cores* o núcleos. A medida que los sistemas multicore van penetrando en la industria, se busca integrar cada vez más funcionalidad en un único sistema. Así, resulta habitual encontrarse con sistemas que deben satisfacer simultáneamente una combinación de requisitos no funcionales, relacionados por ejemplo con la seguridad, la integridad, los tiempos de respuesta o incluso el consumo energético. La aplicación de técnicas de virtualización resulta fundamental en este contexto, ya que habilitan la ejecución de aplicaciones heterogéneas en una misma plataforma computacional.

Dentro de las técnicas de virtualización, el particionamiento del sistema proporciona un entorno de ejecución integrado donde las aplicaciones están aisladas temporal y espacialmente, permitiendo la coexistencia de aplicaciones con distintos niveles de criticidad. Estas técnicas son habituales en sistemas que tienen que pasar por un proceso de certificación para comprobar que cumplen las garantías necesarias establecidas por los estándares de certificación de cada sector industrial. Un ejemplo representativo de particionado en el campo de la aviónica es el propuesto por el estándar ARINC 653 [1].

Aunque el particionado puede ser implementado de diferentes formas [2], las técnicas de particionado basadas en un hipervisor son especialmente relevantes ya que permiten la ejecución de varios sistemas operativos independientes, lo que abre la posibilidad de integrar software empaquetado (COTS) en sistemas particionados.

En general, el particionamiento es utilizado principalmente en los sistemas críticos como pueden ser los de aviónica, aunque se está empezando a aplicar a otros sectores como el energético [3], la automoción [4] o el control industrial [5]. En este contexto, el uso de middleware basado en estándares de distribución puede resultar especialmente atractivo por sus características de interoperabilidad, abstracción de servicios de red o gestión transparente de las comunicaciones.

En la actualidad, uno de los estándares de distribución que más difusión está teniendo es DDS (*Data Distribution Service for Real-Time Systems*) [6], el cual está diseñado explícitamente para el desarrollo de aplicaciones de tiempo real. Uno de los objetivos principales de DDS es facilitar la integración de aplicaciones con independencia del lenguaje de programación o del sistema operativo. Por otro lado, el estándar también define un protocolo denominado DDSI-RTPS [7] para garantizar la interoperabilidad entre implementaciones.

Las características multiplataforma y multilenguaje de DDS, junto con su protocolo de interoperabilidad, resultan especialmente relevantes en sistemas heterogéneos como los que se encuentran habitualmente en los sistemas particionados. Así, en este artículo se muestra el papel que puede jugar la integración de DDS en los sistemas particionados para posibilitar el desarrollo de sistemas distribuidos en los que se pueden combinar aplicaciones críticas o con requisitos de tiempo real, con otras que no tienen restricciones de ejecución o éstas son más laxas. Para ello, se realiza una compilación de las ideas y resultados más relevantes mostrados en diferentes trabajos previos [8][9][10][11][12].

El documento está organizado de la siguiente manera. El apartado 2 está dedicado a presentar las características básicas del middleware de distribución DDS y de los sis-

temas particionados. En el apartado 3 se explora el modelado de DDS para el análisis de tiempo real. La propuesta de una arquitectura para sistemas particionados y distribuidos con DDS se realiza en el apartado 4, mientras que el apartado 5 presenta el desarrollo de una plataforma basada en la arquitectura propuesta. Finalmente, el apartado 6 plantea las conclusiones y el trabajo futuro.

2. DDS y sistemas particionados

2.1. Data Distribution Service

DDS es un estándar de distribución desarrollado por la OMG [6]. Está basado en un paradigma de comunicación denominado editor-suscriptor en el que el middleware se centra en la obtención de datos, trasladando a un segundo plano cuál es el origen de éstos. Las aplicaciones que siguen este modelo forman un sistema distribuido en el que los nodos procesadores se comunican entre sí enviando (los editores) y recibiendo (los suscriptores) datos de forma anónima, tal y como se ilustra en la Figura 1. DDS introduce el concepto de espacio global de datos virtual, en el que el flujo de información puede originarse en uno o varios editores y tener como destino uno o varios suscriptores.

La comunicación entre editores y suscriptores se lleva a cabo a través de la definición del tipo de dato a compartir (*topic*) por parte de los editores. Por su parte, los suscriptores registrarán en el sistema su interés por recibir determinados *topics*, siendo el propio middleware el encargado de hacer posible la comunicación de forma automática y transparente. El modelo propuesto se caracteriza por estar débilmente acoplado, pues tanto editores como suscriptores están desacoplados respecto al tiempo (los datos producidos pueden ser almacenados para un uso posterior) y al espacio (editores y suscriptores no se conocen mutuamente).

Por defecto, el servicio de descubrimiento o localización de entidades definido por DDS es dinámico, esto es, los participantes pueden incorporarse o abandonar el sistema de forma transparente al usuario. Con este fin, el middleware intercambia internamente información sobre la presencia y características de todas las entidades del sistema distribuido, generando un tráfico de red adicional que se conoce como metatráfico.

Además de definir el modelo de distribución, la especificación del DDS también contempla la configuración de distintos aspectos no funcionales del sistema distribuido a través de los parámetros de calidad de servicio (QoS). Estos parámetros permiten configurar, por ejemplo, la disponibilidad o el plazo temporal de recepción de datos.

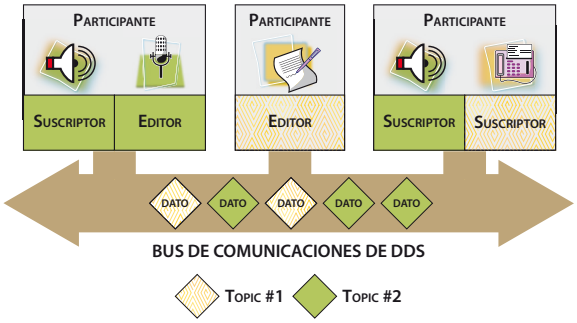

Figura 1 Distribución con DDS

Otro aspecto relevante de DDS para los sistemas de tiempo real son sus mecanismos de tolerancia a fallos. Éstos se basan en la redundancia de editores y/o suscriptores que son gestionados de forma transparente al usuario por parte del middleware.

2.2. Particionado de sistemas

Una partición representa un entorno de ejecución de aplicaciones protegido mediante técnicas de aislamiento y contención de fallos. Por un lado, las técnicas de aislamiento espacial buscan proteger los espacios de memoria asignados exclusivamente a cada partición. Por otro, el aislamiento temporal tiene por objeto garantizar intervalos de ejecución libres de interferencias para cada partición. Habitualmente, este tipo de aislamiento se implementa a través de una planificación estática que asigna a cada partición uno o varias ventanas temporales para su ejecución. Cuando el particionado se realiza a través de un hipervisor, cada partición puede ejecutar un sistema operativo independiente y por tanto existe un segundo nivel de planificación para los hilos o threads de las particiones.

Una de las tareas fundamentales del hipervisor consiste en virtualizar los recursos del sistema (por ejemplo, relojes, temporizadores, interrupciones, etc), así como los núcleos disponibles cuando se trata de un sistema multicore. Aunque existen técnicas de virtualización que no requieren modificar el sistema operativo, su elevada sobrecarga dificulta su uso en sistemas de tiempo real. En este contexto, las técnicas de paravirtualización son más habituales y requieren modificar el sistema operativo para que acceda a los recursos del sistema a través de los servicios proporcionados por el hipervisor.

En los sistemas particionados con ARINC 653, la comunicación entre particiones se lleva a cabo mediante el intercambio de mensajes a través de canales de comunicación. Un canal conecta un único puerto de envío con uno o varios puertos de recepción. Dado que las comunicaciones de un sistema particionado deben estar supervisadas y autorizadas por el hipervisor, la configuración de todos sus parámetros (canales, puertos y sus correspondientes atributos) deben realizarse estáticamente.

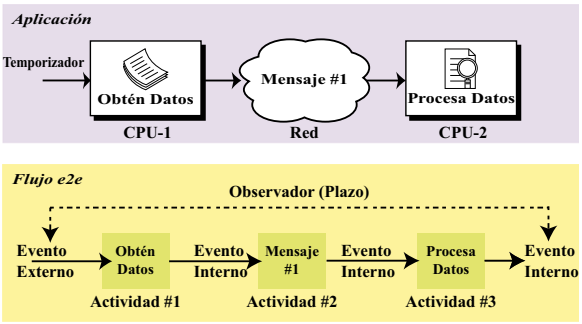

Figura 2 Modelo de tiempo real

Por último, la asignación de los recursos a particiones se realiza de forma estática, normalmente a través de un fichero de configuración. La definición de partición en la configuración se caracteriza por sus áreas de memoria, puertos de comunicaciones y requisitos temporales, así como cualquier otro recurso necesario para su ejecución. Esta asignación estática de recursos es el mecanismo básico para garantizar el determinismo y la seguridad en un sistema particionado.

3. Modelado de DDS para el análisis de tiempo real

En este apartado se analiza el estándar DDS para extraer el modelo que permite la aplicación de técnicas de cálculo de tiempos de respuesta.

3.1. Modelo de tiempo real

Un sistema de tiempo real se puede modelar como un conjunto de transacciones o flujos e2e (*end-to-end flows*) que representan un conjunto de tareas y mensajes en los que existen relaciones de precedencia en su ejecución, tal y como se ilustra en la Figura 2. Cada flujo e2e se activa por la llegada de uno o más eventos externos que generan la ejecución de actividades (por ejemplo, la ejecución de un trozo de código en el procesador o el envío de un mensaje por la red en la Figura 2). Estas actividades a su vez generan eventos que son internos al flujo e2e y que pueden activar a otras actividades del mismo, pudiendo tener cada uno de estos eventos requisitos temporales asociados.

Este modelo se ha utilizado tradicionalmente en el cálculo de los tiempos de respuesta de peor caso tanto en sistemas monoprocesadores como en sistemas distribuidos. Actualmente, este modelo forma parte del estándar de modelado MARTE [13] y puede ser utilizado en procesos de desarrollo dirigido por modelos (MDE) [14].

3.2. Modelado de las características de tiempo real

Aunque DDS está orientado al desarrollo de aplicaciones distribuidas de tiempo real, existen ciertos aspectos que el estándar no aborda y que podrían comprometer los tiempos de respuesta de la aplicación. Por ejemplo, la configuración de los parámetros de planificación de todas las entidades planificables en el sistema (esto es, threads y mensajes), o cotas en la influencia del metatráfico sobre el resto del sistema.

De acuerdo al trabajo presentado en [8], el modelo de distribución propuesto por DDS es adecuado para aplicar las técnicas de análisis de planificabilidad tradicionales sobre sistemas gobernados por eventos. De hecho, el modelo propuesto por DDS facilita la transición entre el modelo teórico de tiempo real definido en MARTE y el sistema físico. Esto se debe principalmente a que las características de desacoplo de DDS proporcionan flexibilidad a la hora de implementar un sistema distribuido, independientemente de la naturaleza síncrona o asíncrona de sus flujos e2e.

3.3. Modelado de los parámetros de QoS

Otra de las características relevantes de DDS es la configuración de los aspectos no funcionales a través de los parámetros de QoS. El uso de estos parámetros puede influir en el comportamiento temporal de la aplicación [15], por lo que resulta necesario explorar si las políticas de QoS definidas en el estándar pueden representarse mediante el modelo de tiempo real [9]. En general, la mayoría de los parámetros de QoS puede modelarse utilizando las relaciones de precedencia de los flujos e2e (lineales o no lineales). En otros casos, el modelado depende de la implementación a utilizar.

Por otro lado, algunos de los parámetros de QoS restringen las relaciones de precedencia que componen el flujo e2e. Por ejemplo, algunos parámetros de QoS pueden provocar el descarte de ciertas muestras de datos, con lo que los flujos e2e que utilicen estas políticas no podrían construirse siguiendo el ejemplo de la Figura 2. En ese caso, habría que utilizar un modelo desacoplado formado por varios flujos e2e independientes, y calcular el tiempo de respuesta de peor caso de principio-a-fin como la suma de los tiempos de respuesta de cada flujo e2e. Esta circunstancia puede influir en las técnicas de análisis de planificabilidad que pueden aplicarse en cada caso particular.

Además de verificar los tiempos de respuesta de peor caso del sistema distribuido, es importante resaltar que el uso de técnicas de modelado facilita también la integración del middleware de distribución en los procesos de desarrollo dirigidos por modelos.

4. Arquitectura de sistema distribuido: DDS sobre particiones ARINC 653

Aunque un hipervisor proporciona servicios propios para la comunicación entre particiones, el uso de un estándar de distribución facilitaría la interoperabilidad entre sistemas heterogéneos, así como el desarrollo de aplicaciones independientemente del lenguaje de programación, las redes subyacentes o la ubicación física. Si nos centramos en el paradigma de distribu-

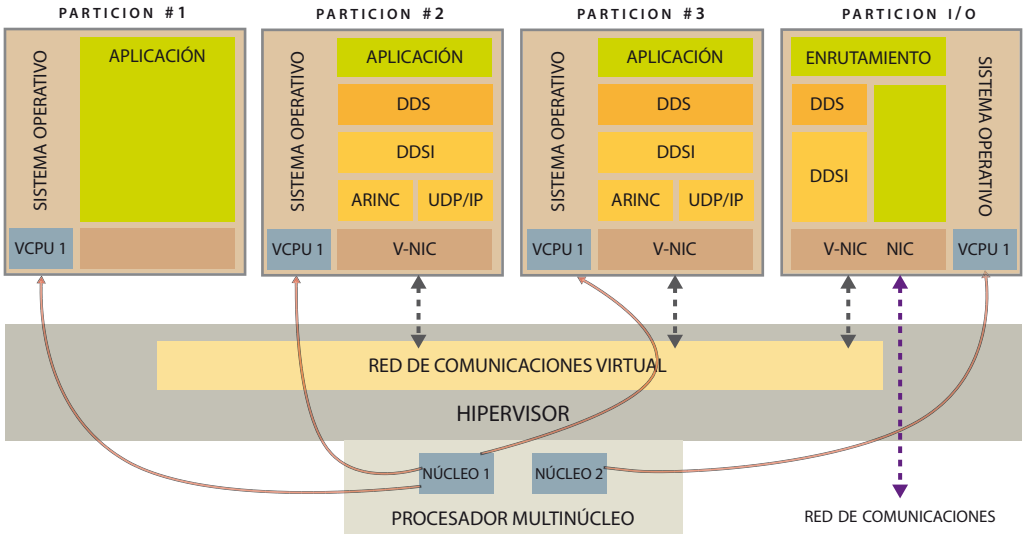

Figura 3. Arquitectura del sistema: nodo procesador multicore

ción, tanto DDS como ARINC 653 proporcionan servicios similares basados en comunicaciones débilmente acopladas y esencialmente asíncronas.

La Figura 3 muestra un ejemplo de la arquitectura del sistema que incluye un hipervisor sobre un procesador multicore con cuatro particiones con sistemas operativos independientes. En la arquitectura propuesta, las comunicaciones se realizan a través de DDS y, por tanto, cada partición que requiera comunicarse ejecutará una instancia del middleware (por ejemplo, las particiones 2 y 3 de la Figura 3). Para establecer con éxito esta comunicación, la plataforma virtualizada debe proporcionar unos recursos adicionales a modo de infraestructura de comunicaciones virtual. Esta infraestructura está compuesta por: (1) una interfaz de red virtual por cada partición, (2) una red virtual de comunicaciones para interconectar las particiones y (3) un mecanismo de acceso seguro a la interfaz de red física.

4.1. Infraestructura de comunicaciones

La interfaz de red virtual o V-NIC (ver Figura 3) proporciona una interfaz de red homogénea a las particiones, independientemente de si éstas tienen o no acceso al dispositivo de red. Esta abstracción se implementa en el nivel del sistema operativo y proporciona las funcionalidades básicas de una interfaz de red como son el envío y la recepción de mensajes. Además, es responsable de crear los puertos de comunicaciones que sirven de punto de acceso a los servicios de comunicaciones ofrecidos por el hipervisor. Por tanto, la V-NIC constituye la entidad fundamental para que el middleware de distribución pueda ejecutarse sin modificaciones en su arquitectura.

La transmisión de mensajes entre particiones se realiza a través de la red virtual de comunicaciones. Esta entidad representa una red lógica que interconecta las particiones locales a un nodo, tal y como se muestra en la Figura 3. La red virtual se implementa en el nivel del hipervisor, el cual gestiona qué comunicaciones están permitidas y cuáles son sus características (por ejemplo, el tamaño máximo de los mensajes intercambiados).

Dependiendo de la criticidad de las particiones, la red virtual puede implementarse utilizando diferentes mecanismos de comunicación, como áreas de memoria compartida entre particiones o canales de comunicación gestionados internamente por el hipervisor. Asimismo, la red virtual del hipervisor debe proporcionar soporte para la notificación de los eventos relacionados con la comunicación (por ejemplo, la llegada de un nuevo mensaje).

Cuando las comunicaciones son entre particiones pertenecientes al mismo nodo, los mecanismos descritos hasta ahora son suficientes para el intercambio de información. Sin embargo, cuando las particiones se encuentran en diferentes nodos es necesario controlar además el acceso al dispositivo de red, ya que varias particiones pueden intentar acceder simultáneamente a él. La implementación de un acceso seguro puede realizarse a nivel del hipervisor o a nivel de partición. En el primer caso, se incrementa la complejidad y el tamaño del hipervisor, lo que hace que su código sea más difícil de verificar y certificar. En el segundo caso, el acceso al recurso de red compartido se controla a través de una partición especial denominada partición de entrada/salida o partición de I/O, tal y como se muestra en la Figura 3. Esta partición tiene acceso exclusivo a la interfaz de red y actúa como intermediario entre el origen del mensaje y su destino.

4.2. Middleware de distribución

Como ya se ha comentado, las similitudes entre los modelos de distribución propuestos por las especificaciones DDS y ARINC 653 facilitan la integración de ambos. Esta integración puede realizarse tanto a nivel

de hipervisor como a nivel de partición [11]. En el primer caso, DDS implementaría el servicio de comunicaciones del hipervisor y por tanto, se utilizaría como un bus de comunicaciones y protocolo de interoperabilidad. Sin embargo, integrar DDS a nivel de partición es una opción más flexible ya que abre la posibilidad de utilizar además las políticas de QoS definidas por el estándar. La Figura 3 muestra esta segunda opción en la que una instancia del middleware DDS se ejecuta en cada partición que requiera comunicaciones. Al mismo tiempo, el middleware puede integrarse en la partición de dos formas distintas: (1) mediante una implementación estándar de DDS que se comunica a través de la pila de protocolos UDP/IP [10]; y (2) a través de una implementación certificable de DDS, en la que parte de la funcionalidad de DDS puede estar restringida y los protocolos de comunicaciones pueden estar basados directamente en los servicios de comunicaciones del hipervisor [12] (por ejemplo, las comunicaciones ARINC 653).

En general, la partición de I/O no ejecutará ninguna instancia del middleware de distribución ya que su función se reduce a redireccionar los mensajes recibidos. Sin embargo, en sistemas no críticos también puede ser una opción ejecutar una instancia del middleware en la partición de I/O [10].

5. Claves de la integración DDS/ARINC 653

Aunque como ya se ha dicho, el modelo de distribución de DDS es similar al propuesto por el servicio de comunicaciones de los sistemas particionados con ARINC 653, existen diferencias entre ambos estándares que pueden comprometer la integración de ambas tecnologías, tal y como se describe a continuación.

5.1. Retos en la integración de tecnologías

El modelo de distribución definido por DDS soporta las interacciones uno-a-uno, uno-a-muchos y muchos-a-uno. Sin embargo, la interacción muchos-a-uno suele estar restringida en sistemas particionados para minimizar el riesgo de pérdida de mensajes [16].

En general, la interacción muchos-a-uno es habitual en los sistemas distribuidos tradicionales. Por ejemplo, el middleware suele abrir un puerto de comunicación pre-configurado y público donde puedan contactar los nuevos participantes. Además, en el caso de DDS, la interacción muchos-a-uno es un elemento fundamental para su servicio de tolerancia a fallos basado en la redundancia de editores. Para solventar este problema de integración, se propone soportar este tipo de interacción en el nivel del protocolo DDSI-RTPS, tal y como se ilustra en la Figura 4.

El tamaño de los mensajes intercambiados entre las particiones puede ser variable, pero debe ser acotado para mantener el determinismo del sistema. De

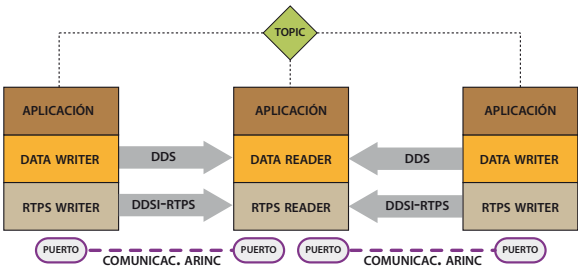

Figura 4 Interacción muchos-a-uno en sistemas particionados con DDS

acuerdo al estándar, el tamaño de los mensajes de DDS no está acotado ya que pueden incluir tanto datos de usuario como internos al protocolo (metatráfico). Por tanto, las implementaciones de DDS para sistemas particionados deberían permitir la configuración del tamaño máximo de los mensajes.

5.2. Características de las comunicaciones

La duración y asignación de ventanas temporales a particiones representa un aspecto clave en el diseño de sistemas particionados. En concreto, resulta especialmente relevante la configuración elegida para la partición de I/O, ya que actúa como intermediaria en todas las comunicaciones con el exterior. Por tanto, esta partición debería ejecutarse con regularidad suficiente para que las particiones puedan recibir/enviar sus mensajes de acuerdo a sus plazos temporales. Al mismo tiempo, la ejecución regular de esta partición limita la ejecución del resto de particiones y puede llegar a penalizar ampliamente su rendimiento [10]. Una opción para mitigar este problema viene dada por el uso de un procesador multicore en el que uno de los núcleos sea asignado a la partición de I/O [11].

Como contrapartida, el uso de procesadores multicore incrementa la complejidad en la gestión de las interrupciones. En un procesador con un solo núcleo, las particiones procesan las interrupciones al comienzo de su ejecución. Así, una interrupción de red se genera en la recepción de un mensaje, pero no se procesa hasta que entre en ejecución la partición de I/O. Una vez en ejecución, la partición de I/O redireccionará el mensaje y la correspondiente interrupción a la partición de destino. Por el contrario, un procesador multicore permite que la partición de I/O y la partición de destino del mensaje puedan estar en ejecución al mismo tiempo y, por tanto, que puedan interferirse durante su ventana de ejecución. Desde la perspectiva del rendimiento del sistema, resulta interesante que el mensaje pueda ser entregado tan pronto como se haya recibido. Para ello, es necesario incorporar mecanismos que permitan notificar la llegada de nuevos mensajes entre particiones (por ejemplo, interrupciones entre núcleos). Sin embargo, desde la perspectiva de la seguridad del sistema, es necesario que el hipervisor

también proporcione mecanismos de protección ante ráfagas incontroladas de interrupciones.

Además de la comunicación basada en paso de mensajes, algunos sistemas particionados permiten el uso de memoria compartida para comunicar particiones que requieran un mayor ancho de banda [4][10]. En este caso, el concepto de aislamiento espacial debe interpretarse de una forma más laxa, esto es, desde el punto de vista de la configuración estática del sistema, el hipervisor supervisa qué áreas de memoria son compartidas y sólo permite el acceso a las particiones autorizadas previamente.

5.3. Análisis de planificabilidad

La arquitectura propuesta en el apartado 4 puede beneficiarse del uso de técnicas de análisis de tiempos de respuesta de peor caso para verificar los plazos temporales del sistema distribuido particionado. Para ello, es necesario que estas técnicas tengan en cuenta las características de los sistemas distribuidos con DDS [8] [9], y también las de los sistemas particionados con ARINC 653. Para estos últimos, se pueden usar técnicas compatibles con sistemas gobernados por eventos tanto en los procesadores [17] como en las redes de comunicaciones [18].

6. Desarrollo de una plataforma distribuida

Este apartado describe la implementación de una plataforma distribuida de tiempo real que responde a la arquitectura presentada. Esta plataforma está basada en la implementación de DDS denominada RTI Connext Micro [19] que se ejecuta sobre el sistema operativo de tiempo real MaRTE OS [20], y que a su vez se ejecuta sobre el hipervisor XtratuM [21]. El desarrollo de esta plataforma se ha centrado en validar la arquitectura propuesta en el apartado 4. En concreto, la plataforma proporciona soporte para las siguientes características descritas en la Figura 3:

- La red de comunicaciones virtual está basada en el servicio de comunicaciones definido en ARINC 653, que en la plataforma está proporcionado por el hipervisor XtratuM.
- La interfaz de red virtual se ha implementado como un *driver* de MaRTE OS que proporciona una funcionalidad similar a los *drivers* de los dispositivos de red.
- La comunicación entre las particiones por medio de DDS se realiza por medio del protocolo de interoperabilidad DDSI-RTPS. En la plataforma, este protocolo se puede ejecutar sobre dos niveles de transporte diferentes: un transporte basado en la pila de protocolos UDP/IP, que es el definido por defecto en el estándar, y un transporte basado en el servicio de comunicaciones de ARINC 653.

La integración de estas tecnologías en la plataforma no puede realizarse directamente y requiere implementar un conjunto de extensiones que se describen brevemente a continuación.

6.1. Extensiones a la implementación de DDS

RTI Connext Micro [19] es un middleware de distribución que sigue el estándar DDS y que está especialmente orientado a sistemas empotrados y/o certificables. Este middleware está diseñado para ser multiplataforma e integrar diferentes redes de comunicaciones. Para ello, implementa dos capas de abstracción de los servicios que debe proporcionar el sistema operativo y las comunicaciones subyacentes. Aunque algunos de los servicios requeridos por el middleware de distribución se proporcionan de forma virtualizada por el hipervisor, uno de nuestros objetivos es el uso de COTS, por lo que el middleware debe acceder a estos servicios a través de una interfaz estándar del sistema operativo como POSIX. En concreto, la adaptación del middleware a la plataforma requiere implementar las siguientes funcionalidades:

- *Servicios del sistema operativo*. Se ha desarrollado una capa de adaptación de los servicios proporcionados por MaRTE OS, incluyendo gestión de memoria, concurrencia, sincronización, gestión del tiempo y registro de eventos.
- *Servicios de red*. Se han añadido los servicios generales de gestión y registro de las interfaces de red existentes en el sistema. Además, la incorporación de un nuevo transporte basado en ARINC 653 requiere implementar dos capas de software adicionales: (1) una capa para las operaciones de alto nivel requeridas por el middleware (por ejemplo, operaciones para enviar/recibir mensajes DDSI-RTPS, creación de threads para operaciones de I/O, la gestión de los recursos de red, etc.), y (2) otra capa para interactuar directamente con el servicio de comunicaciones ARINC 653 y la interfaz de red virtual proporcionada por el sistema operativo.
- *Servicios de localización de entidades*. Un nuevo mecanismo para gestionar la información relacionada con la ubicación de las particiones y la forma de contactar con ellas.

6.2. Extensiones al sistema operativo de tiempo real

El sistema operativo utilizado en la plataforma es MaRTE OS [20], un *kernel* que proporciona la funcionalidad de la norma POSIX.13 [22] para plataformas empotradas de tiempo real.

En un sistema particionado como el propuesto, el hipervisor XtratuM es el encargado de gestionar todo el hardware de la plataforma. Para facilitar el acceso de las particiones a estos recursos, XtratuM proporciona una interfaz de acceso al hardware. Por tanto, los

sistemas operativos de las particiones no pueden acceder al hardware de forma nativa, sino que deben ser modificados para utilizar esta interfaz. En el caso de MaRTE OS, el *kernel* proporciona una capa de abstracción de bajo nivel para acceder al hardware, por lo que la adaptación se ha centrado en modificar esa capa de bajo nivel a la interfaz proporcionada por el hipervisor [23]. En concreto, se han realizado las siguientes modificaciones:

- *Adaptación al modelo de interrupciones virtualizadas*. Dado que el manejo de interrupciones corresponde al hipervisor, el sistema operativo debe adaptarse con objeto de utilizar las operaciones proporcionadas por el hipervisor para acceder a las interrupciones virtualizadas equivalentes, así como al resto de interrupciones internas al hipervisor que son propias de un sistema particionado. Además, el sistema operativo debe configurarse para atender exclusivamente las interrupciones autorizadas por el hipervisor para cada partición.
- *Integración de interfaces de red virtualizadas*. Para proporcionar acceso a los servicios de comunicaciones de XtratuM, se ha implementado la interfaz de red virtual en MaRTE OS. Esta entidad se ha implementado como un *driver* del sistema operativo que proporciona a las aplicaciones una interfaz estándar POSIX para acceder al dispositivo virtual (*open*, *close*, *read*, *write* e *ioctl*). La interfaz de red virtual es responsable de crear y configurar los puertos de comunicación establecidos en la configuración de la partición, así como de proporcionar una funcionalidad similar a la de la interfaz física de red. Dado que los servicios de comunicación proporcionados por XtratuM son no bloqueantes, otro aspecto a implementar ha sido un mecanismo bloqueante para la recepción de mensajes. Este mecanismo se basa en la gestión de interrupciones internas del hipervisor para notificar la entrega de mensajes a las particiones.
- *Uso de servicios virtualizados para la gestión del tiempo.* XtratuM proporciona relojes y temporizadores para la gestión del tiempo en las particiones. Así, los relojes y temporizadores de propósito general proporcionados por el sistema operativo, junto con otros servicios avanzados (por ejemplo, los relojes de tiempo de ejecución) deben implementarse utilizando los recursos virtuales proporcionados por XtratuM.
- *Integración en el proceso de desarrollo*. Una vez que la configuración del sistema particionado ha sido validada, el proceso de desarrollo genera un contenedor software que integra el gestor de arranque, las particiones y el hipervisor. Por tanto, el entorno de compilación cruzada de MaRTE OS debe integrarse dentro de ese proceso de desarrollo.

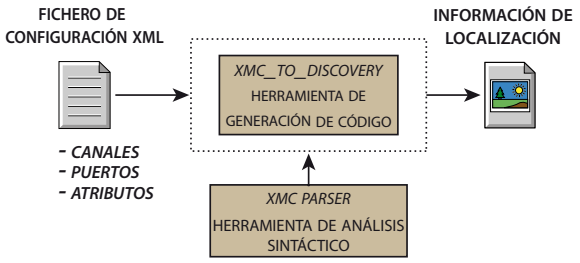

Figura 5 Generación automática de la información de localización de entidades para DDS

6.3. Extensiones al hipervisor

XtratuM es un hipervisor especialmente diseñado para sistemas empotrados de tiempo real [21]. El servicio de comunicaciones implementado por XtratuM está basado en la especificación ARINC 653, pero sólo proporciona soporte para la comunicación entre particiones ubicadas en el mismo nodo procesador, es decir, que no requieren el uso de la red de comunicaciones. Por tanto, resulta necesario extender la configuración del hipervisor para indicar si las particiones van a ejecutarse o no en el mismo nodo [12]. Además, DDS requiere acceder a esta información de configuración para ejecutar su servicio de localización de entidades. En concreto, la integración del hipervisor en la plataforma distribuida ha requerido las siguientes extensiones:

- *Identificación del nodo en las comunicaciones*. Cada instancia del hipervisor puede ser identificada mediante la incorporación de un identificador al fichero de configuración. Dado que los canales de comunicación están definidos a nivel de sistema y no de partición, este identificador está también vinculado a los puertos de comunicación.
- *Generación automática de la información de contacto de las particiones para DDS*. Tal y como se ilustra en la Figura 5, se ha desarrollado una nueva herramienta *xmc_to_discovery* que genera la información que requiere DDS a partir del fichero de configuración de XtratuM.

7. Conclusiones y trabajo futuro

En este trabajo se ha explorado el modelado de tiempo real de aplicaciones distribuidas con DDS. Aunque DDS está orientado al diseño de aplicaciones distribuidas de tiempo real, quedan ciertos aspectos relevantes por concretar en el estándar y que por tanto quedan abiertos a la implementación.

Además, se ha analizado su integración en sistemas particionados con ARINC 653. Dado que uno de los principales inconvenientes de los sistemas distribuidos particionados es el cuello de botella en el uso de las comunicaciones, se ha propuesto una arquitectura distribuida utilizando un nodo multiprocesador, en el que

uno de los núcleos está dedicado a gestionar la interfaz de red. Por último, se ha presentado un desarrollo concreto de la plataforma distribuida de acuerdo con la arquitectura propuesta.

Con el incremento de la conectividad de los sistemas, uno de los aspectos que más preocupa en la actualidad es la seguridad de los sistemas. Por ello, resultaría interesante explorar la seguridad de la plataforma propuesta desde la perspectiva del middleware y del hipervisor. Por otro lado, nuestro trabajo continua en la línea de los sistemas críticos mediante la integración de redes de comunicaciones especializadas como AFDX.

Agradecimientos

Este trabajo ha sido financiado en parte por el Gobierno de España en el proyecto TIN2014-56158-C4-2-P (M2C2).

Referencias

[1] Airlines Electronic Engineering Committee, Aeronautical Radio INC. “Avionics Application Software Interface, required Services”. ARINC Specification 653-1. 2010.

[2] S. Han y H. Jin. “Resource partitioning for Integrated Modular Avionics: comparative study of implementation alternatives”. Software: Practice and Experience (SPE), http://dx.doi.org/10.1002/spe.2210. 2014.

[3] J. Pérez, D. González, S. Trujillo, T. Trapman y J. M. Garate. “A Safety Concept For A Wind Power Mixed-Criticality Embedded System Based On Multicore Partitioning”, Proc. of the 1st Int. Workshop on Mixed Criticality Systems, Vancouver (Canada), 2013.

[4] Open VEhiculaR SEcurE platform (OVERSEE) European Project, 7th Framework Prog., https://www.oversee-project.com, 2013.

[5] Open and cost-effective virtualization techniques and supporting separation kernel for the embedded systems industry (VOS4ES) European Project, 7th Framework Prog., http://cordis.europa.eu/project/rcn/100889_en.html, 2013.

[6] Object Management Group. Data Distribution Service for Real-time Systems. v1.4, formal/15-04-10, 2015.

[7] Object Management Group. The Real-time Publish-Subscribe Wire Protocol. DDS Interoperability Wire Protocol Specification. v2.2, formal/2014-09-01, 2014.

[8] H. Pérez y J. J. Gutiérrez, “On the schedulability of a data-centric real-time distribution middleware”, Computer Standards and Interfaces, Elsevier, Volume 34, Issue 1, págs. 203–211, 2012.

[9] H. Pérez, y J. J. Gutiérrez, “Modeling the QoS parameters of DDS for event-driven real-time applications” Journal of Systems and Software, Volume 104, págs. 126-140, 2015

[10] H. Pérez, y J. J. Gutiérrez, “Enabling data-centric distribution technology for partitioned embedded systems”, IEEE Trans. on Parallel and Distributed Systems, Vol. 27(11), págs. 3186-3198, 2016.

[11] H. Pérez, J. J. Gutiérrez, S. Peiró, y A. Crespo. “Distributed architecture for developing mixed-criticality systems in multi-core platforms”. The Journal of Systems and Software, Vol. 123, Elsevier, págs. 145-159, 2017.

[12] H. Pérez, y J. J. Gutiérrez, “Handling heterogeneous partitioned systems through ARINC-653 and DDS”. Computer Standards & Interfaces, Vol. 50, Elsevier, págs. 258-268, 2017.

[13] Object Management Group.“A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems”, v1.1.http://www.omg.org/spec/MARTE/1.1/. 2011.

[14] L.Rioux, R. Henia, N. Sordon, M. González Harbour, J.J. Gutiérrez, J.M. Rivas, C. Cuevas, J.M. Drake, y J.L. Medina, “Schedulability analysis and optimization in a model-based integrated tool-chain: Synthetic MARTE models for optimizing real-time design with MAST and TEMPO” Conf. on Forum on specification & Design Languages, Demo Night Session, Spain, 2015.

[15] J. L. Poza, J. L. Posadas, y J.E. Simó. “From the Queue to the Quality of Service Policy: A Middleware Implementation”. Proc. of the Int. Conference on Artificial Neural Networks, Berlin, 2009.

[16] Airlines Electronic Engineering Committee, Aeronautical Radio INC. “Aircraft Data Network, Part 7 - Avionics Full Duplex Switched Ethernet (AFDX) Network”. ARINC Specification 664-7. September, 2009.

[17] J.C. Palencia, M. González Harbour, J.J. Gutiérrez, y J.M. Rivas. “Response-Time Analysis in Hierarchically-Scheduled Time-Partitioned Distributed Systems”. IEEE Transactions on Parallel and Distributed Systems, 28(7), págs. 2017-2030, 2017.

[18] J.J. Gutiérrez, J.C. Palencia, y M. González Harbour. “Holistic schedulability analysis for multipacket messages in AFDX networks”. Journal of Real-Time Systems 50(2), Springer, págs. 230-269, 2014.

[19] RTI Connext DDS Micro is available at http://www.rti.com. Last access in June, 2017.

[20] M. Aldea y M. González. “MaRTE OS: An Ada Kernel for Real-Time Embedded Applications”. Proc. of the Int. Conference on Reliable Software Technologies, Ada-Europe, Leuven, Belgium, LNCS 2043. 2001.

[21] M. Masmano, I. Ripoll, A. Crespo, y J.J. Metge, “Xtratum a hypervisor for safety critical embedded systems”, Proc. of the 11th Real-Time Linux Workshop, Germany, 2009.

[22] IEEE Portable Application Standards Committee (PASC). “Standard for Information Technology-Portable Operating System Interface (POSIX) Realtime and Embedded Application Support”, Std. 1003.13. 2003.

[23] H. Pérez, M. Aldea y D. Medina, “Multiprocessor platform for partitioned real-time systems”, Software: Practice and Experience, In Press, doi: 10.1002/spe.2404, 2016.