RNPS No. 2143
ISSN 2072-6260
Versión Digital

CENATAV
Centro de Aplicaciones de Tecnologías de Avanzada

REPORTE TÉCNICO
Minería de Datos

SERIE GRIS

Estado del arte sobre aplicaciones del middleware ROS

Alexis Fernández Rivas, Osvaldo Andrés Pérez García y José Hernández Palancar

RT_033 **febrero 2016**

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

RNPS No. 2143
ISSN 2072-6260
Versión Digital

SERIE GRIS

REPORTE TÉCNICO
Minería de Datos

Estado del arte sobre aplicaciones del middleware ROS

Alexis Fernández Rivas, Osvaldo Andrés Pérez García y José Hernández Palancar

RT_033 **febrero 2016**

CENATAV
Centro de Aplicaciones de Tecnologías de Avanzada

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

Estado del arte sobre aplicaciones del middleware ROS

Alexis Fernández Rivas[1], Osvaldo Andrés Pérez García[1] y José Hernández Palancar[2]

[1]Equipo de Servicios de Infraestructura Tecnológica para la Investigación (CENATAV - DATYS), La Habana, Cuba
{afernandez, osvaldo.perez}@cenatav.co.cu
[2]Equipo de Investigaciones de Biometría (CENATAV - DATYS), La Habana, Cuba
jpalancar@cenatav.co.cu

RT_033, Serie Gris, CENATAV- DATYS
Aceptado: 29 de diciembre de 2015

Resumen. Un sistema distribuido está compuesto por varios elementos de hardware y software de diversas características, los cuales deben integrarse y cooperar, con independencia de su heterogeneidad, y con el fin de solucionar un problema determinado. Dentro de un sistema distribuido, el middleware es el elemento que facilita el manejo de la complejidad de dichos entornos. En el presente reporte se aborda un middleware para entornos robóticos denominado Sistema Operativo Robótico y se hace un estudio del estado del arte de la aplicabilidad de este en entornos distribuidos fuera de la robótica. Se incluye, además, una implementación del middleware a modo de prueba de concepto, que permite corroborar los resultados del estudio realizado, y se realizan propuestas de investigación y trabajos futuros sobre la temática.

Palabras clave: sistemas distribuidos, middleware, sistema operativo robótico, entornos inteligentes, vigilancia.

Abstract. A distributed system is composed of several hardware and software elements of different characteristics, which must be integrated and cooperate, regardless of their heterogeneity, and in order to solve a particular problem. The element that facilitates the management of the complexity of such environments within a distributed system is the middleware. In this report, a middleware for robotic environment called Robotic Operating System is discussed. This middleware allows the developing modular systems, with hardware abstraction, and mechanisms of communication between independent processes. In addition, it was included an implementation of middleware as concept proof, which corroborated the results of the study, and several researches and future works about this topic are proposed.

Keywords: Keywords: distributed systems, middleware, robotic operating system, intelligent environments, and surveillance.

1 Introducción

Hoy día los problemas tratados por las ciencias informáticas y de la computación son cada vez más complejos. En temas tales como el análisis de datos para la seguridad, ejemplo: en la gestión de eventos y de la información de seguridad (SIEM, por sus siglas en inglés), en la video vigilancia avanzada (Advanced CCTV Surveillance en inglés), en la conducción asistida (Assisted Navigation en inglés), en

los Entornos Inteligentes (IE, por sus siglas en inglés), en el seguimiento de personas (People Tracking en inglés) y la Internet de las cosas (IoT, por sus siglas en inglés), entre otros, se necesita trabajar con datos almacenados en disímiles formatos; se precisa ejecutar diferentes algoritmos[1]; se requieren de capacidades especiales para operar en tiempo real; así como integrar hardware de diferentes características (heterogeneidad), ejecutar gran variedad de procesos, paralelizar múltiples tareas y del trabajo con servicios web. Este nivel de complejidad tecnológica y procedural obliga a buscar soluciones integradoras que, de acuerdo con las propiedades y requerimientos de los sistemas, posibiliten manejar todos los elementos involucrados en cada una de las temáticas mencionadas. Esa es una de las razones por la cual el paradigma de los sistemas distribuidos ha tomado auge en estos tiempos.

A un sistema de procesamiento de información que contiene un conjunto de computadoras independientes cooperando entre ellas a través de una red de comunicación, con el fin de lograr un propósito u objetivo específico, se le denomina sistema distribuido[1]. Este concepto no es solamente aplicable a las computadoras, es decir, no solo tiene una dimensión física sino también lógica, y desde ese punto de vista se puede definir un sistema distribuido como un conjunto de procesos o aplicaciones independientes que cooperan entre sí. En este caso, la comunicación entre dichos procesos puede no depender de la red física dado que es posible que todos esos procesos residan en un mismo ordenador [1].

El empleo de este paradigma trae consigo diversas ventajas respecto a la idea de los sistemas centralizados, en los cuales una sola computadora se encarga de realizar todas las tareas del sistema sin interactuar con otras. En la Tabla 1, se resumen algunas de ellas.

Si bien las ventajas que ofrece el empleo de los sistemas distribuidos son notables respecto a la mayoría de los parámetros, exceptuando el costo económico y la seguridad, estos también requieren de un mecanismo que permita desarrollar y ejecutar aplicaciones de forma distribuida y, al mismo tiempo, debe ser capaz de manejar la heterogeneidad de hardware presente en dichos sistemas, ocultar los detalles de la distribución y proveer un conjunto común de servicios específicos. Tal mecanismo es conocido como middleware [2].

Tabla 1. Ventajas y desventajas de los sistemas centralizados vs. los distribuidos (Tomada de [1]).

Criterio	Sistema centralizado	Sistema distribuido
Económico	bajo	alto
Disponibilidad	baja	alta
Manejo de la complejidad	bajo	alto
Escalabilidad	pobre	buena
Tipo de tecnología	homogénea	heterogéneo
Seguridad	alta	baja

[1] Por ejemplo, para controlar métricas, correlacionar datos, obtener nuevos conocimientos e incluso, pronosticar eventos, entre otras posibilidades.

Los middlewares son el corazón de los sistemas distribuidos y, por tal motivo, son requeridos cada vez que se desee desarrollar sistemas de este tipo. Existen una gran variedad de middlewares en la actualidad lo cual se debe, entre otras razones, a la diversidad de temas en los que pueden ser aplicados. Sin embargo, esto no impide que los middlewares desarrollados inicialmente para un determinado campo de la ciencia (por ejemplo, la robótica) no puedan ser empleados en otras áreas del conocimiento. Dentro de este grupo se encuentra el middleware "*Robotic Operating System"* (ROS) el cual fue concebido inicialmente como *framework* para el desarrollo de sistemas robóticos. También, dada la semejanza de dichos sistemas con otras esferas tecnológicas (en especial con aquellas que requieren de combinar técnicas de procesamiento de imágenes y audio, de reconocimiento de rostros, objetos, personas, entre otros), ROS ha sido empleado como middleware para el desarrollo de otros sistemas distribuidos, contando entre sus ventajas el hecho de que está producido bajo licencia de código abierto y es gratuito, lo cual facilita la aplicación de esta tecnología en múltiples entornos. En este trabajo se presenta un estudio del middleware ROS, sus características y posibilidades fuera del marco de la robótica en sistemas donde se apliquen técnicas de reconocimiento de patrones.

Este trabajo está estructurado de la siguiente manera: en la sección 2 se exponen las características y propiedades de los middlewares mientras que en la 3 se define qué es ROS, la filosofía de diseño que siguieron sus creadores y su principio de funcionamiento. En la sección 4 se expone un estudio del estado del arte de la aplicación de ROS fuera del marco de la robótica, mientras que en la sección 5 se presenta una prueba de concepto realizada con este middleware en el campo de la videovigilancia. En 6 se concluye y se exponen posibles líneas de trabajo futuro.

2 Middleware

En la actualidad existe una amplia penetración a nivel global de las tecnologías de la información y las telecomunicaciones lo cual ha posibilitado la proliferación de sistemas distribuidos o, como también pudiera denominarse, aplicaciones distribuidas [1]. Estas aplicaciones tienen por característica fundamental que las partes que la componen realizan sus funciones en ubicaciones físicas diversas, permitiendo erradicar limitaciones geográficas, incrementar la tolerancia a fallos de los sistemas y mejorar el desempeño mediante el paralelismo de tareas; por solo mencionar algunas de sus posibilidades. Los sistemas distribuidos enfrentan, hoy día, una problemática fundamental y es la heterogeneidad en los elementos que lo componen: diferentes plataformas de hardware, tecnologías de red, sistemas operativos y lenguajes de programación. Lo anterior representa todo un desafío a la hora desarrollar dichos sistemas.

Tal y como se mencionó en la sección introductoria, el middleware es la infraestructura que permite el desarrollo y la ejecución de aplicaciones distribuidas. Este provee un conjunto de componentes básicos de alto nivel que simplifican la construcción del sistema y permiten, a desarrolladores e investigadores, abstraerse de los detalles de la implementación a bajo nivel, tales como control de concurrencia, gestión de transacciones y la comunicación a través de la red, entre otros, y a su vez, le permiten enfocarse en los requerimientos de aplicación [3].

La Figura 1 muestra un ejemplo de un sistema distribuido, comprendido por una red de computadoras y tres aplicaciones (A, B y C), en donde la aplicación B está distribuida entre los ordenadores 2 y 3. A cada aplicación se le ofrece una interfaz común (el middleware) la cual les permite comunicarse entre sí a la vez que abstrae, a dichas aplicaciones, de la naturaleza del hardware y de los sistemas operativos presentes (Figura 1, aplicación B).

El middleware como servicio se considera de propósito general [4], y se posiciona entre plataformas y aplicaciones con el fin de solucionar los problemas de heterogeneidad y distribución entre ambos. Entiéndase por plataforma como el conjunto de servicios de bajo nivel y los elementos de procesamiento, corriendo sobre algún tipo de arquitectura y un sistema operativo (ver Figura 2).

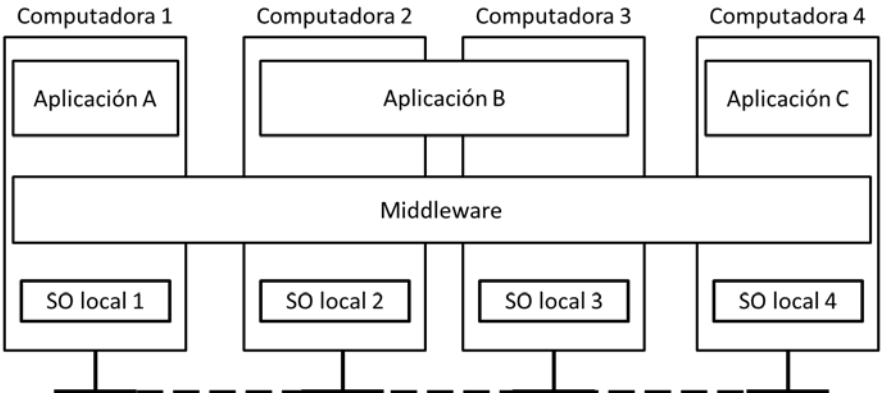

Fig. 1. Ejemplo de sistema distribuido.

El middleware también se identifica [5] como el software que asiste a una determinada aplicación en la interacción o comunicación con otras aplicaciones, redes, hardware y/o sistemas operativos y que se encarga de lidiar con la complejidad de las conexiones presentes en los sistemas distribuidos, a la vez que provee herramientas para mejorar la calidad del servicio (QoS, por sus siglas en inglés), la seguridad, el intercambio de información, entre otros elementos que pueden ser imperceptibles por el usuario.

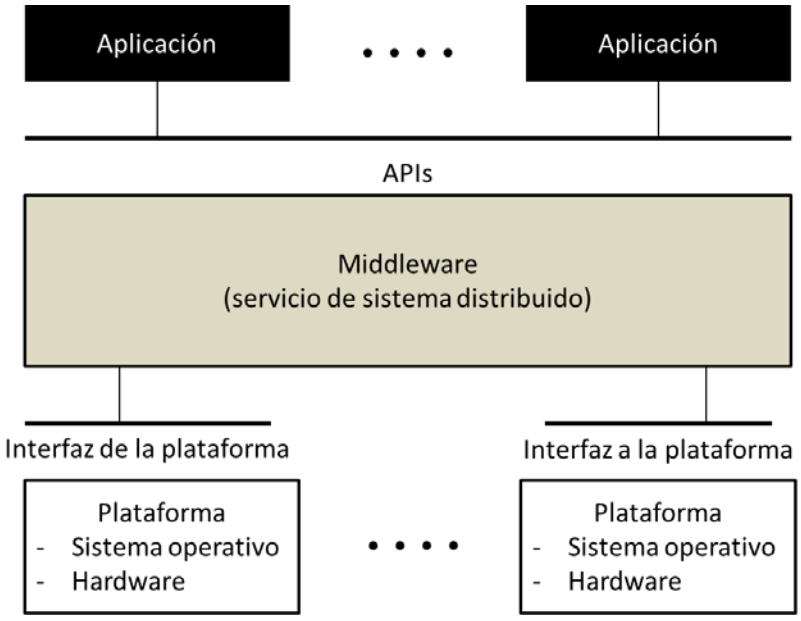

Fig. 2. Concepto de plataforma en un sistema distribuido.

En resumen, el middleware es el SO de los entornos distribuidos ya que provee abstracción de alto nivel para los recursos de la red así como facilita el acceso a los recursos distribuidos.

2.1 Propiedades de los middleware

Las propiedades generales que caracterizan a un middleware son las siguientes [4]:

- **Multiplataforma**: el middleware debe poseer implementaciones que le permitan ejecutarse en varias plataformas (independencia del hardware y del sistema operativo). Esta propiedad amplía la

cobertura de plataformas para aquellas aplicaciones que dependan de esta y, si el servicio es distribuido, también mejora la interoperabilidad del sistema dado que las aplicaciones de diferentes plataformas se pueden comunicar entre sí e intercambiar información.

- **Distribuido**: se refiere a la capacidad de un middleware para ser accedido de forma remota o permitir que servicios y/o aplicaciones sean ejecutados de la misma forma.
- **Estándar**: capacidad para soportar protocolos, APIs y lenguajes de programación estándares.
- **Escalable**: dada las características de los sistemas distribuidos (integración de gran cantidad de plataformas) el middleware debe ser capaz de incorporar, en tiempo real, nuevos elementos al sistema sin disminuir el desempeño de este.

2.2 Requerimientos de los middlewares

W. Emmerich presentó en [3] los requerimientos en cuanto a comunicación, sincronismo, confiabilidad, escalabilidad y heterogeneidad, que deben cumplir los productos middleware para que funcionen como una capa intermedia que satisfaga las necesidades de un sistema distribuido determinado.

2.2.1 Comunicación

Los sistemas distribuidos poseen la particularidad de que sus aplicaciones o componentes pueden residir en diferentes computadoras y, por consiguiente, estos necesitan comunicarse entre sí. En la mayoría de los casos dicha comunicación solo es posible a través de una red de datos y haciendo uso de los protocolos de intercambio de información de dicha red (por ejemplo, TCP o UDP). Por tal motivo, es usual que los sistemas distribuidos sean compilados sobre la capa de transporte de una red de datos (por ejemplo redes Ethernet), ya que a este nivel se establecen las sesiones de comunicación entre dos terminales para, posteriormente, intercambiar información [6].

Cada vez que un elemento de un sistema distribuido desee acceder y/o consumir algún servicio paramétrico que esté ofreciendo cualquiera de los restantes elementos o aplicaciones de dicho sistema, este requiere de alguna implementación (interfaz) que le permita iniciar una sesión de comunicación y acomodar los parámetros (estructuras de datos) a la forma de la carga útil de, por ejemplo, un datagrama TCP o UDP (secuencia de bytes). Realizar dicha implementación en cada elemento del sistema es costoso, propenso a generar errores y consume tiempo [3]. Por tal razón, el middleware es el encargado de gestionar la comunicación entre dichos elementos liberándolos de iniciar sesiones y de presentar la información. Los componentes ahora solo tienen que ocuparse de enviar los parámetros correctos de solicitud del servicio y de esperar por la respuesta a dicha solicitud, dejándole al middleware la tarea de acomodar dichas solicitudes y respuestas a la estructura de datos adecuada. Este proceso es conocido como “acomodamiento”.

2.2.2 Sincronismo

Los sistemas distribuidos requieren de mecanismos de sincronización que gestionen la comunicación entre los componentes una vez que estos deseen intercambiar información. Esto se debe a que, por ejemplo, los componentes que residen en un mismo ordenador se ejecutan de forma concurrente, es decir, son independientes entre sí y pueden estar operando a la vez, por lo que se necesita de algún método que les permita coordinar enlaces (comunicación) entre ellos en caso de que lo requieran en algún punto de su ejecución. En [3] plantean que la sincronización debe ser implementada como parte de la capa de sesión del middleware.

Los métodos de sincronización descritos en [3] son los siguientes: síncrono, síncrono diferido y asíncrono. El método síncrono se pone de manifiesto cuando la ejecución de un componente es bloqueada o detenida tras este haber realizado una solicitud de servicio a otro componente. La ejecución del componente llamador se reanuda una vez que recibe la respuesta (ver Fig. 3).

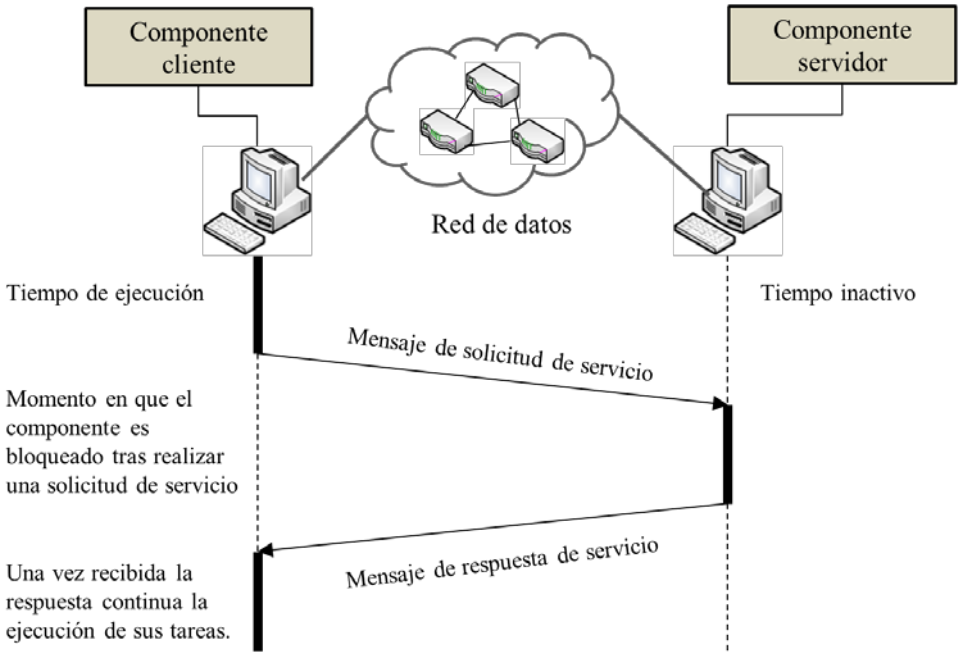

Fig. 3 Mecanismo de comunicación síncrono.

A diferencia del método síncrono, el síncrono diferido permite que el componente siga ejecutando tareas tras realizar la solicitud del servicio. En este caso, el elemento llamador se sincroniza en algún momento posterior con el proveedor del servicio y es su responsabilidad iniciar la sincronización (esto lo hace mediante un sondeo a intervalos de tiempo al servidor).

Por último, el método asíncrono sigue el mismo principio que el del síncrono diferido, es decir, el componente que solicita un servicio a otro componente no detiene la ejecución de sus tareas en espera de la respuesta, sin embargo, este no realiza un sondeo para verificar si el servidor terminó de procesar la solicitud sino que espera a que este último le responda, o sea, que inicie la sincronización (ver Fig. 4). Este tipo de método requiere de algún mecanismo intermedio que permita almacenar y ordenar las solicitudes emitidas, dicho mecanismo es conocido como cola de mensajes [7].

Cuando más de dos componentes desean hacer una solicitud de servicio a otro componente en específico, se ejecuta una solicitud de grupo; para ello se pueden hacer uso de los protocolos de redes multidifusión, aunque se requiere de un soporte adicional por parte del middleware que garantice la confiabilidad de la entrega y el acomodamiento de los parámetros de las solicitudes [3].

2.2.3 Confiabilidad

Algunos de los protocolos de red empleados por los sistemas distribuidos para comunicarse no garantizan la entrega de cada paquete que el transmisor envía al receptor, así como tampoco garantizan mantener el orden en que se envían dichos paquetes. Por tal motivo, los sistemas distribuidos requieren de mecanismos capaces de detectar y corregir errores durante la transferencia de la información. Tales mecanismos pueden disminuir el desempeño de los componentes [3]. Debido a esto, es necesario implementar técnicas de confiabilidad que permitan lograr un compromiso entre ambas métricas (desempeño y confiabilidad) en dependencia de las exigencias o prioridades definidas, los cuales pueden estar orientados a garantizar la entrega de una o varias solicitudes de un servicio determinado entre dos componentes o para un conjunto de componentes [3]. En la Tabla 2 se muestran algunos de los mecanismos de confiabilidad más utilizados.

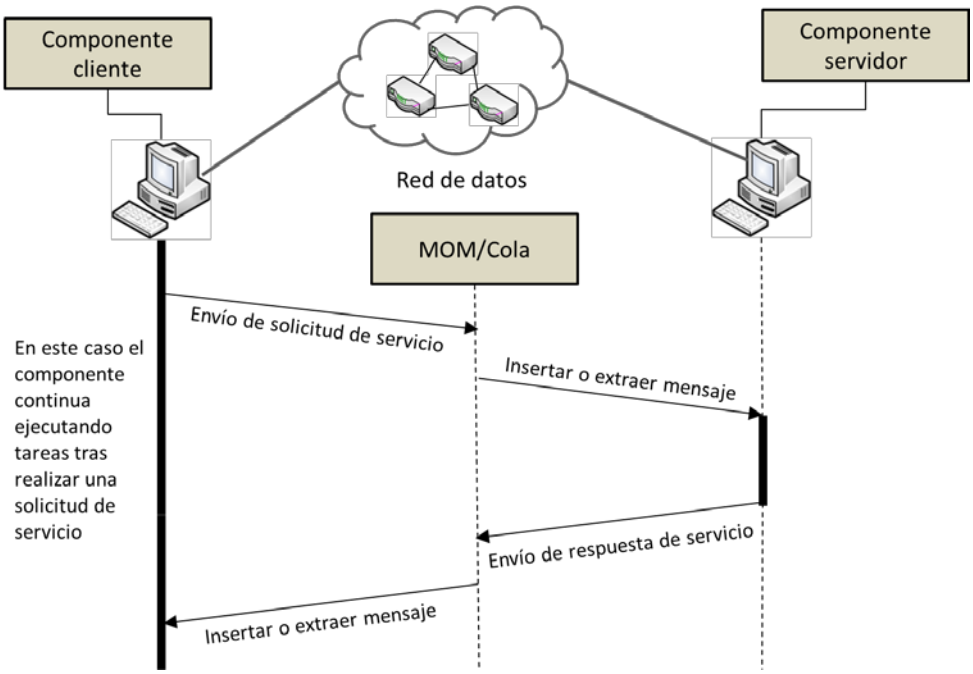

Fig. 4 Mecanismo de comunicación asíncrono.

Tabla 2. Mecanismos de confiabilidad.

Solicitud de un servicio	Mecanismo de confiabilidad
Entre dos componentes	Mejor esfuerzo
	Como máximo una vez
	Al menos una vez
	Exactamente una vez
Entre grupos de componentes	k-confiabilidad
	Time-outs
	Solicitudes totalmente ordenadas

Otra forma de aumentar la confiabilidad es replicando componentes [3], lo cual significa que varias copias de un mismo componente deben residir en diferentes dispositivos (redundancia), de manera tal que si el componente servidor principal falla, alguna de sus réplicas va a continuar operando, evitándose así la interrupción del servicio.

2.2.4 Escalabilidad

Una tendencia común en los sistemas distribuidos es la de crecer y hacerse más complejos con el transcurso del tiempo. Debido a esto, los middlewares deben ser capaces de asimilar nuevos componentes y plataformas (hardware), sin que esto implique una disminución en el desempeño del sistema.

Otro de los retos que afronta el diseño de un sistema distribuido escalable es el de soportar cambios en la distribución de los componentes por cada computadora del Sistema, sin que se modifique la arquitectura de dicho sistema o el diseño del código de cualquier componente. En [3] se plantea que la forma de lograr lo anterior es siguiendo el enfoque estructural para la construcción de sistemas distribuidos: transparencias de distribución del Modelo de Referencia de Procesamiento Distribuido Abierto (RM-ODP, por sus siglas en inglés) [8] de la Organización Internacional de Estándares (ISO, por sus siglas en inglés). Dicho enfoque plantea que una infraestructura provee transparencia de

distribución cuando es capaz de ocultar sus funcionalidades y problemas a la aplicación o sistema de aplicaciones que soporta.

El RM-ODP es un estándar que define el "cómo hacer" para desarrollar aplicaciones distribuidas abiertas en donde la premisa principal es: la de abstracción de la lógica de negocios de la capa subyacente o infraestructura tecnológica sobre la cual esta es desplegada.

Un ejemplo de transparencia de distribución es la transparencia en el acceso, cuyo enfoque está orientado a que el acceso a un servicio de un componente por parte de otro, sea independiente a la representación de los datos o los métodos de invocación del servicio, sea local o remoto. Otro ejemplo es la transparencia de la ubicación, la cual demanda que los componentes no tienen que conocer la ubicación física de otro componente para poder interactuar con este. De los ejemplos anteriores se puede concluir que un sistema distribuido, en donde sus componentes pueden acceder a otros componentes sin la necesidad de conocer la ubicación física de estos y sin cambiar la forma en que realizan las solicitudes a los servicios que exponen, debe ser capaz de soportar mecanismos de balanceo de carga que permitan distribuir el grado de explotación de los ordenadores [3]. El proceso anterior es totalmente transparente para el usuario, es decir, este no sería consciente de los cambios que se están realizando cuando está consumiendo determinado servicio y se necesita realizar, por ejemplo, el balanceo de carga para garantizar la calidad de ese servicio.

2.2.5 Heterogeneidad

La heterogeneidad es una característica propia de los sistemas distribuidos debido a la cantidad de aplicaciones y elementos de hardware legados y nuevos que lo componen. Lo anterior tiene como implicación directa que el middleware debe de ser capaz de lidiar con la diversidad de hardware, sistemas operativos, lenguajes de programación inclusive hasta con otros middleware dentro del mismo sistema.

En ocasiones un solo middleware no es capaz de satisfacer todas las necesidades del sistema [3]; por lo que se requiere de la combinación de un conjunto de estos. Por tal motivo, los middlewares deben poseer la característica de ser interoperables con otras implementaciones del propio middleware o con diferentes tipos de middlewares.

2.3 Clasificación de los middlewares

Los middlewares pueden ser clasificados, por ejemplo, teniendo en cuenta la forma de comunicación o el modo de operación; sin embargo en la literatura la mayoría de los enfoques coinciden en agruparlos según las herramientas básicas o primitivas que estos proveen para desempeñar la interacción entre los componentes distribuidos. De acuerdo a este criterio, las categorías son las siguientes:

- Middlewares transaccionales.
- Middlewares orientados a mensajes (MOM, por sus siglas en inglés).
- Middlewares de procedimiento.
- Middlewares de objetos o componentes.

Por otro lado, en [5] propusieron una taxonomía en donde los middlewares son agrupados en dos categorías principales: los de integración y los de aplicación. En la categoría de integración concentran aquellos middlewares que poseen una forma específica para ser integrados a un sistema heterogéneo determinado. Mientras que, en la categoría aplicación, incluyen aquellos middlewares que se ajustan a un tipo específico de función de una aplicación determinada, en otras palabras, son middlewares de servicios locales los cuales se posicionan como una capa adicional, por ejemplo, de un sistema operativo, y lo complementan con una serie extra de servicios [5]. En la primera categoría están contemplados todos los tipos de middlewares vistos anteriormente, mientras que la segunda categoría agrupa el siguiente conjunto de middlewares:

- Middleware de acceso de datos.
- Middleware escritorio.

- Middleware basado en web.
- Middleware de tiempo real.
- Middleware especializado.

En el presente trabajo nos enfocaremos sobre los middlewares de integración, en particular los middlewares orientados a mensajes, subcategoría que incluye a ROS. Para mayor conocimiento acerca de los middlewares de aplicación y de los restantes tipos de middlewares de integración, consultar [3, 5].

2.4 Middlewares orientados a mensajes

Un middleware orientado a mensajes funciona como una capa intermedia que garantiza la recepción y entrega de mensajes entre aplicaciones clientes y aplicaciones servidores, a través de una red de datos. Su confiabilidad se basa en la implementación de mecanismos de almacenamiento persistente o temporal de mensajes y de reenvió de información. Ambos mecanismos evitan las pérdidas de mensajes debido a posibles fallas en el sistema y/o a las pérdidas naturales de paquetes que ocurren en las redes de datos [7]. Todo lo anterior se logra gracias a las facilidades que brinda la cola de mensajes dentro del MOM: ordenamiento y almacenamiento de la información. De dicha cola depende la confiabilidad en la entrega de la información por parte del middleware tanto a los clientes como a los servidores.

Los middlewares orientados a mensajes proveen un modo de comunicación asíncrono entre los componentes de un sistema distribuido. Los elementos del sistema pueden realizar llamadas a servicios sin la necesidad de que estos detengan la ejecución de sus tareas en espera de la respuesta (ver sección 2.2.2). Por otro lado, también posibilita la entrega de mensajes independientemente del estado tanto del componente llamador como el del llamado, ya sea porque estos no se encuentren activos o disponibles, para responder en tiempo de ejecución.

En [7] plantean que un MOM posibilita independencia de las características de desempeño de cada uno de los componentes del sistema distribuido, es decir, que cualquier cambio que se produzca en alguno de los elementos del sistema o en los subsistemas no va a afectar el desempeño general de este. También existe la posibilidad de elegir qué elemento atiende una determinada llamada de servicio o no, lo cual permite realizar balanceo de carga dentro del sistema ya que se puede, por ejemplo, aliviar la carga de un determinado elemento y redirigir el tráfico de solicitudes hacia otro que esté siendo escasamente explotado. Por otro lado, MOM presenta algunas limitaciones en cuanto a la transparencia de acceso (ver sección 2.2.4). Esto se debe a que carece de sentido que los componentes del sistema empleen colas de mensajes para comunicarse con componentes locales. Lo anterior implica que en un sistema distribuido basado en MOM la comunicación entre los componentes en pos de solicitar algún servicio no es independiente de si el componente servidor es local o remoto y constituye una limitación a la hora de escalar el sistema con nuevos componentes ya que estos tienen que tener en cuenta dicha condición [5].

2.4.1 Esquemas de intercambio de mensajes

Un MOM cuenta con dos esquemas de intercambio de mensajes: punto a punto y publicador/ subscriptor, ambos esquemas están basados en el intercambio de mensajes a través de una cola de mensajes. Las características y tipos de colas de mensajes se salen del alcance del presente trabajo. Para conocer más sobre ellas, consultar [9].

Punto a punto

Este esquema de intercambio de mensajes provee una forma asíncrona de intercambio de datos en un sentido estricto. En este caso, los mensajes que son emitidos por los componentes llamadores (clientes) hacen uso de una cola de mensajes para llegar a sus destinatarios, como se había mencionado antes. Un tipo de cola muy empleada en este modelo es la Primero en Entrar Primero en Salir (FIFO, por sus siglas en inglés), en la cual los mensajes son consumidos en el mismo orden en que estos llegan, es decir, el primer dato recibido será el primer dato en ser consumido por algún elemento del sistema.

La principal característica de este modelo es que no hay restricción alguna en el número de clientes que pueden publicar en una cola, pero solo un elemento consumidor o servidor puede consumir los mensajes emitidos [7]. La Figura 5 muestra una representación de este modelo de intercambio y de una cola de mensajes FIFO.

Publicador/subscriptor
El modelo de intercambio de mensajes publicador/subscriptor brinda facilidades para difundir la información entre elementos consumidores y emisores de mensajes anónimos. Este mecanismo de distribución de mensajes uno-a-muchos y muchos-a-muchos permite que un solo elemento publicador pueda enviar a sus mensajes a cientos de elementos consumidores. Por otro lado, ambos tipos de elementos no tienen que preocuparse por las características de la aplicación objetivo ya que este esquema cuenta con un componente central que se encarga de gestionar el intercambio de información. Este esquema será abordado en mayor detalle en la sección 3.3.1 ya que ROS hace uso de este.

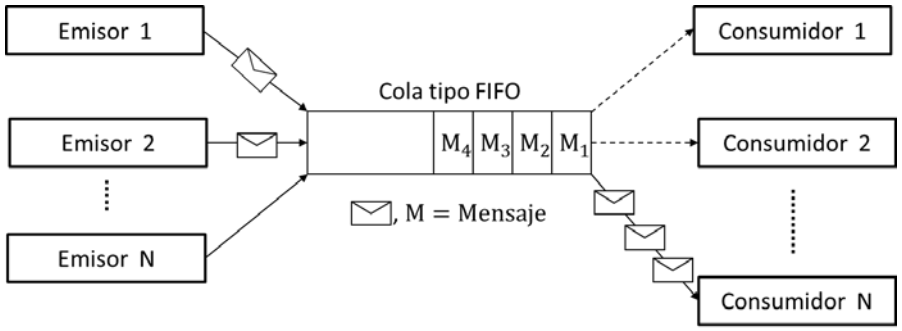

Fig. 5 Esquema de intercambio de mensajes punto-a-punto.

3 Sistema operativo robótico (ROS)

A pesar de su nombre, ROS no es un sistema operativo en el sentido tradicional [10], su objetivo es proveer una capa de comunicación estructurada por encima del sistema operativo instalado en un ordenador, el cual puede formar parte de un clúster heterogéneo de computadoras [11].

3.1 Motivación para el desarrollo del middleware ROS

En [12] se exponen algunas de las problemáticas que presenta el desarrollo de software para robots:

1. *Necesidad de realizar computación distribuida:* La mayoría de los sistemas robóticos modernos dependen de la operación conjunta de muchos y diferentes procesos que se ejecutan a través de varias computadoras. Por ejemplo, un robot puede estar conformado por varias computadoras donde cada una se encarga de gestionar algunas de las partes del robot. Por otro lado, es una buena práctica el tener dividida la aplicación que opera el robot, en pequeños procesos que trabajen de forma conjunta para lograr un objetivo determinado.
2. *Necesidad de un mecanismo de comunicación:* Los procesos antes mencionados necesitan de un mecanismo que les permita comunicarse entre ellos, estén estos o no residiendo en el mismo ordenador.
3. *Necesidad de reutilización de software:* La evolución progresiva en la investigación de la robótica trae consigo el desarrollo de múltiples algoritmos que permiten realizar tareas específicas de este campo tales como navegación, planificación de movimiento, reconocimiento de objetos, reconocimiento de

rostro, entre otros; lo mismo ocurre con los controladores que permiten manejar el hardware que conforma a los autómatas. Por otra parte, la mayor utilidad de cada una de esas aplicaciones radica en la posibilidad de poder reutilizarlas en otros sistemas.

Basado en lo anterior un equipo, de investigadores y desarrolladores tanto del Departamento de Ciencias de la Computación de la Universidad de Stanford como de la empresa Willow Garage[2] liderados por Morgan Quigley, se dieron a la tarea de diseñar y desarrollar un middleware para entornos robóticos que les permitiera acometer un conjunto de tareas específicas durante el desarrollo de robots de servicio para al proyecto STAIR[3] de la Universidad de Stanford y el Programa Personal de Robots de Willow Garage. El middleware creado resultó ir más allá de las metas trazadas en un principio por el equipo de M. Quigley, quienes determinaron que este podía ser generalizado como middleware para el desarrollo robótico [11].

3.2 Filosofía de diseño de ROS

En [11] se exponen los criterios de diseño que siguieron los desarrolladores de ROS, los cuales, afirmaron que hasta ese momento (año 2009) no tenían noticias de algún middleware para la robótica que cumpliera con dichos requerimientos, los cuales se pueden resumir de la siguiente manera:

1. Topología de comunicación punto a punto.
2. Basado en herramientas.
3. Multi-lenguaje.
4. Ligero.
5. Gratis y de código abierto.

3.2.1 Topología de comunicación punto a punto

Un sistema construido a partir del uso de ROS como middleware, básicamente consiste en un conjunto de procesos que se ejecutan en diferentes computadoras, los cuales se interconectan en tiempo real siguiendo una topología de comunicación punto a punto (peer to peer en inglés[4]) [11].

La Figura 6 muestra un ejemplo simple de cómo pudiera ser un sistema basado en ROS y un posible esquema de red para dicho sistema. Se puede apreciar como el módulo ROBOT está constituido por varias computadoras interconectadas vía Ethernet[5] las cuales se encargan de realizar tareas poco complejas tales como mover el robot o gestionar sus sensores. Por otra parte, el segmento de red del módulo ROBOT se conecta a través de un canal inalámbrico (por ejemplo WiFi[6]) con un clúster de procesamiento en donde se ejecutan tareas de mayor demanda de recursos, tales como reconocimiento de voz o de rostro [11].

[2] https://www.willowgarage.com/
[3] http://stair.stanford.edu/
[4] En este tipo de topología los host de la red son capaces de comportarse como clientes y servidores a la vez [13].
[5] Estándar de redes área local para computadoras.
[6] Estándar de comunicación inalámbrico.

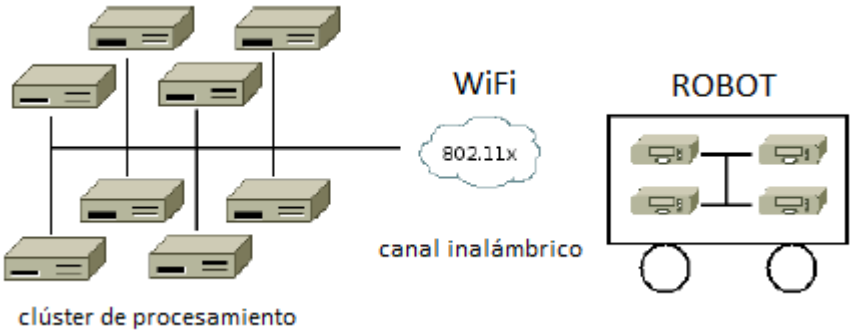

Fig. 6. Ejemplo de un sistema basado en ROS. Tomado de [11].

La idea de esta filosofía es que si el robot debe realizar una tarea específica, por ejemplo, dirigirse hacia un objeto determinado; entonces solo se activen y comuniquen aquellos procesos necesarios para lograr dicho objetivo. Este principio ayuda a que redes con un ancho de banda limitado, como es el caso de las redes inalámbricas, no se saturen, así como también permite crear sistemas altamente modulares.

Es válido resaltar que este tipo de topología requiere de un mecanismo de búsqueda que le permita a los procesos encontrarse entre ellos en tiempo real. Este mecanismo es parecido a un servicio de dominio de nombres (DNS, por sus siglas en inglés [6]) [11].

3.2.2 Basado en herramientas

Para gestionar la complejidad de ROS, M. Quigley y colaboradores[11] decidieron implementar un gran número de pequeñas herramientas que permiten compilar y ejecutar varios componentes de ROS. Estas herramientas pueden realizar tareas tales como navegación, obtener y establecer parámetros de configuración, establecer la comunicación entre procesos, entre otras.

Según [11], esta idea de modularidad limita la eficiencia; sin embargo, al mismo tiempo afirman que esta pérdida es compensada por la ganancia en cuanto a estabilidad y manejo de la complejidad de ROS.

3.2.3 Multi-lenguaje

ROS fue diseñado sobre la idea de que los desarrolladores e investigadores puedan trabajar con este independientemente del lenguaje de programación que usen. Por tal motivo, ROS soporta de forma nativa cuatro diferentes lenguajes de programación: C++, Python, Octave y LISP; en el caso de Octave; este se implementa través de una “envoltura” (wrapper, en inglés) que provee una biblioteca de C++ para este.

Por otra parte, las negociaciones de comunicación y configuración entre los procesos ocurren según el mecanismo XML-RPC (XML acrónimo de Lenguaje de Marcas Extensible y RPC acrónimo de Llamada de Procedimiento Remoto) cuya implementación existe en varios lenguajes de programación.

3.2.4 Ligero

Una de las metas que persigue ROS es la de brindar a la comunidad de investigadores la posibilidad de reutilizar los códigos y controladores generados durante el desarrollo de sus proyectos. Para lograr esto M. Quigley y colaboradores [11] defendieron la idea de que el desarrollo de algoritmos y controladores, para un determinado sistema robótico, ocurriera dentro de bibliotecas independientes que no contengan ninguna dependencia en ROS, lográndose con esto ubicar de forma virtual toda la complejidad, en cuanto a código se refiere, en dichas bibliotecas. De esta forma solo se requiere de pequeños ejecutables que expongan las funcionalidades de dichas bibliotecas a ROS, lo cual, a su vez, facilita la extracción y reutilización del código.

Otra ventaja directa de hacer uso de bibliotecas independientes es que se facilita el proceso de prueba y depuración del código creado.

3.2.5 Gratis y de código abierto

Todo el código fuente de ROS esta público y disponible para cualquier usuario. También está distribuido bajo los términos de la licencia BSD (BSD acrónimo de Distribución de Software Berkeley) lo cual posibilita desarrollar proyectos y productos comerciales y no comerciales, con este.

3.3 Definición de ROS y principio de funcionamiento

En [14] se hace una definición formal de ROS como un cuasi sistema operativo para robots de código abierto. Este provee los servicios que se esperan de un sistema operativo tales como abstracción de hardware, control de dispositivos a bajo nivel, implementación de funcionalidades bien conocidas, intercambio de mensajes entre procesos y gestión de paquetes. Este también provee un conjunto de herramientas que permiten obtener, compilar y ejecutar código a través de múltiples computadoras. Esta definición de ROS resume en alguna medida lo expuesto en la sección anterior.

En ROS se aplican cuatro conceptos fundamentales que permiten describir cómo funciona este: nodos, mensajes, tópicos y servicios [11].

Un nodo es un proceso que realiza una determinada funcionalidad [14]. Este elemento es el que permite realizar el diseño de un sistema modular haciendo uso de ROS debido a que es posible ejecutar simultáneamente, en un mismo host o de manera distribuida en diferentes hosts, múltiples nodos, donde cada uno realiza una tarea específica [15]. Por otro lado, es importante señalar que estos nodos necesitan de al menos un mecanismo de comunicación para poder operar coordinadamente. En ROS los nodos se comunican haciendo uso de los siguientes métodos: mensajes y servicios. El primer método ocurre de forma asincrónica mientras que el segundo puede ocurrir tanto de forma síncrona como asíncrona.

3.3.1 Mensajes

Un mensaje en ROS no es más que una estructura de datos basadas en modelos estrictamente tipados [11]. De manera general ROS soporta tipos de datos estándares (enteros, de punto flotante, booleanas, entre otros) así como arreglos de estos.

Los mensajes proveen un canal de comunicación unidireccional entre los nodos [16]. Un mensaje puede estar compuesto por varios mensajes o por arreglos de varios mensajes. Un nodo envía un mensaje a través del paradigma publicador/subscriptor [11] en donde, si un nodo desea enviar un mensaje a otro, este publica la información en un tópico (esto no es más que una etiqueta que permite diferenciar una información de otra); y si otro nodo está interesado en adquirir dicha información entonces se subscribe al tópico publicado (ver Fig. 7). Deben destacarse tres características relacionadas con este método [11]:

1. Un tópico puede tener múltiples subscritores y publicadores mientras que un nodo puede subscribirse y/o publicar múltiples tópicos.
2. Los nodos publicadores o subscriptores ignoran la existencia de otros nodos publicadores/subscriptores. Esto se explica más adelante en la sección 3.3.3.
3. El proceso de subscripción a un tópico y la posterior transferencia de la información puede ocurrir en cualquier momento, es decir, es un mecanismo de comunicación asincrónico.

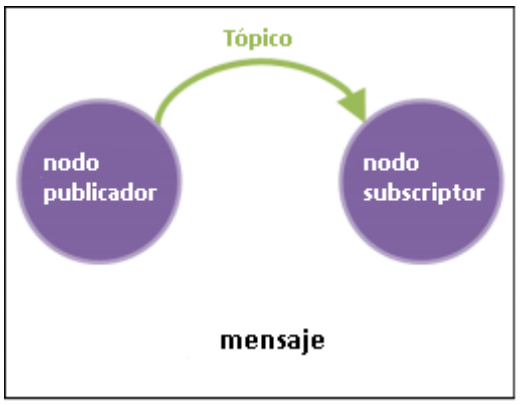

Fig. 7. Mecanismo de transferencia de mensajes: publicador/subscriptor. Tomado de [16].

3.3.2 Servicios

Un servicio permite la comunicación bidireccional entre dos nodos. Este puede ser entendido como un llamado a una función para que realice un proceso computacional determinado (acción de "respuesta") para lo cual solo se necesita proveer de los parámetros necesarios a dicha función (acción de "solicitud") (ver Fig. 8). Este mecanismo es análogo al de los servicios web los cuales están definidos mediante URIs (siglas en inglés de Identificador de Recursos Uniformes) y poseen documentos de solicitudes y respuestas bien definidos [11].

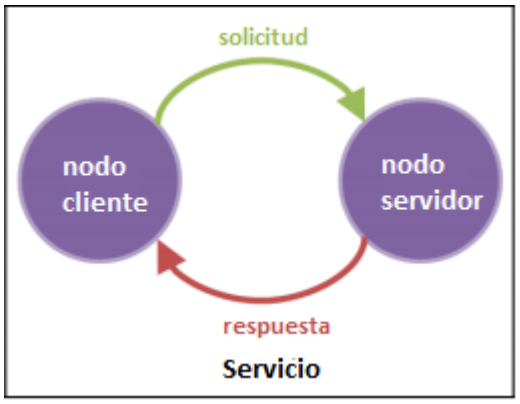

Fig. 8. Mecanismo de comunicación mediante llamadas a servicios. Tomado de [16].

En ROS, uno o varios servicios únicos pueden ser implementados y ejecutados dentro de un solo nodo, denominado nodo servidor, mientras que las solicitudes a dichos servicios pueden originarse a partir de uno o varios nodos referenciados como nodos clientes. Este tipo de comunicación ocurre de la forma punto a punto, es decir, un nodo realiza una llamada un servicio específico y la respuesta es retornada a ese mismo nodo [12, 16].

Finalmente, señalar que las llamadas a los servicios pueden ser efectuadas tanto de forma síncrona como asíncrona [16].

3.3.3 Nodo máster

En la sección 3.3.1 se hizo referencia a que un nodo no tiene conocimiento de los tópicos publicados por otros nodos y viceversa. Lo anterior implica que los nodos no son capaces por sí mismos de iniciar una comunicación entre ellos. Por tal motivo, ROS implementa una solución que es capaz de asistir el establecimiento de las comunicaciones entre uno o varios nodos, así como de advertir los tópicos y servicios existentes en la red, dicho elemento se denomina nodo máster.

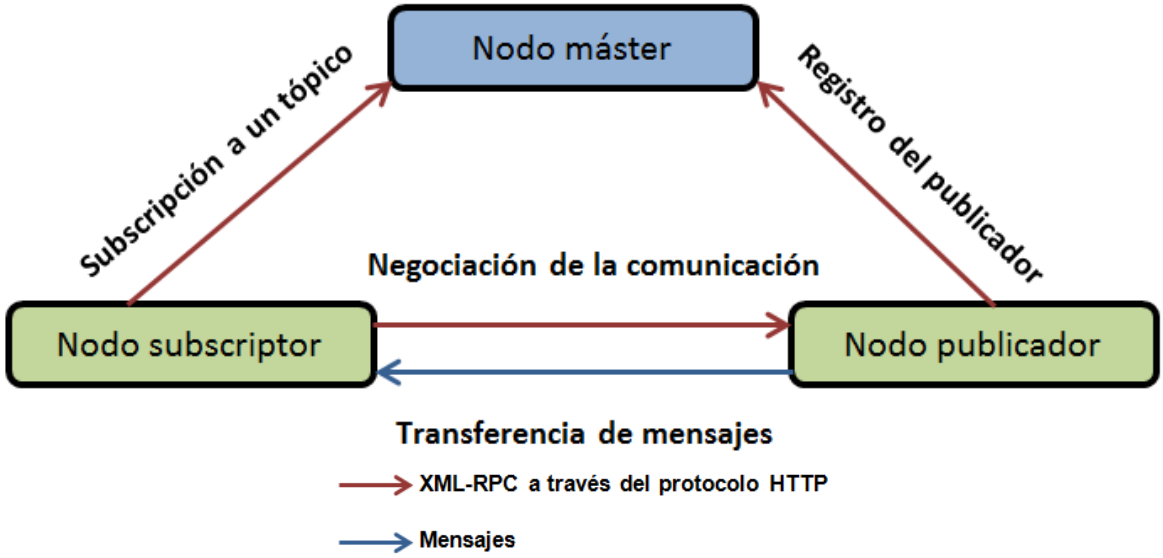

Fig. 9. Intercambio directo de información entre los nodos, sin la intervención del máster. Tomado de [15].

El nodo máster expone su interfaz haciendo uso del estándar XML-RPC lo cual le permite a los nodos realizar llamadas a este para registrar un nuevo tópico, obtener una lista de los tópicos publicados, obtener una lista de los nodos que estén publicando un tópico específico, registrar un nuevo servicio, obtener una lista de servicios y obtener información acerca de un servicio específico [15]. Por otro lado, los nodos solo cuentan con el máster como medio para poder establecer comunicaciones por lo que si este falla entonces no se pueden establecer nuevas conexiones ni, por consiguiente, realizar nuevas transferencias de información.

Una vez que los nodos negocian con el máster los parámetros de la comunicación y la comunicación en sí, estos quedan directamente conectados (ver Fig. 9). La comunicación establecida no fluye a través del máster y, por consiguiente, es independiente a cualquier falla o cambio de configuración que presente este.

4 Aplicaciones distribuidas basadas en el middleware ROS

A pesar de que ROS está entendido para ser un middleware orientado al desarrollo de la robótica, su aplicación, hoy día, no se ha detenido ahí y ha llamado la atención de investigadores y desarrolladores de otras áreas donde se requiere de un middleware que permita diseñar sistemas modulares, robustos y con heterogeneidad de software y hardware. Tras realizar un estudio de la literatura se pudo concluir que hay presencia de ROS fuera del marco de la robótica, concretamente, como parte del desarrollo de tecnologías para Entornos Inteligentes y para la Vigilancia. Ambas esferas del desarrollo tecnológico son abordadas con mayor detalle en las subsecciones posteriores a este epígrafe. También como resultado de dicho estudio se comprobó que ROS está por encima, en cuanto a soporte de hardware y repositorio de componentes, de otras tecnologías middlewares orientadas al desarrollo de la robótica que existen en la actualidad [17, 18].

En la Fig. 10 se muestra una taxonomía de los ámbitos en los que ha sido empleado ROS como middleware.

Las aplicaciones distribuidas desarrolladas para la robótica haciendo uso de ROS como middleware se salen del alcance del presente trabajo, sin embargo, vale la pena mencionar algunas reportadas en la literatura: En [19] proponen dos métodos que permiten mejorar la integración entre los robots y las personas basándose en mejorar la capacidad de los robots de adquirir información acerca del entorno y

de los objetos que lo conforman. En [20] presentan un software basado en ROS que provee un conjunto de herramientas que le permiten al robot aprender y detectar en tiempo real sonidos del mundo real. En [21] se prueban una serie de algoritmos que permiten manipular un brazo mecánico a partir del movimiento de los dedos. Dicho movimiento es captado por un guante equipado con sensores. Para esto emplearon a ROS como middleware lo cual les permitió integrar el hardware (guante con sensores) con la simulación de dicho brazo mecánico y comunicar de forma flexible los diversos procesos que conformaron la aplicación (ver Fig. 11).

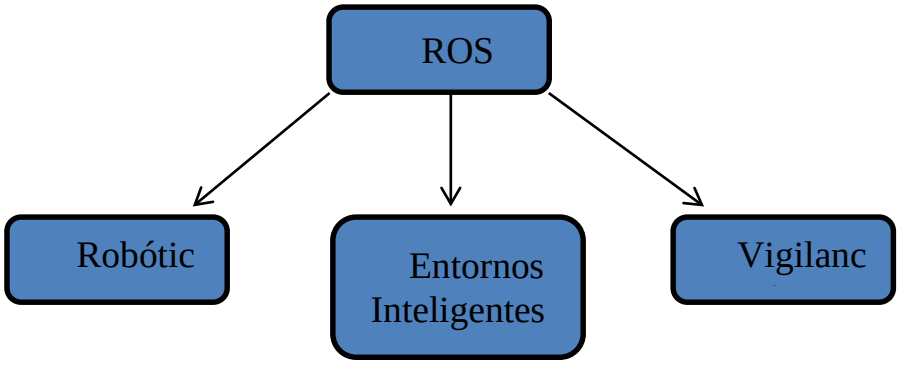

Fig. 10. Taxonomía de los ámbitos de aplicación de ROS.

4.1 Entornos inteligentes

En [15] definen a los Entorno Inteligente como entornos ordinarios equipados con dispositivos computacionales que permiten mejorar la experiencia de las personas ubicadas dentro de estos. Otra definición presente en la literatura es: “un mundo pequeño, donde todo tipo de dispositivos inteligentes están trabajando continuamente para hacer la vida de sus habitantes más confortable” [22].

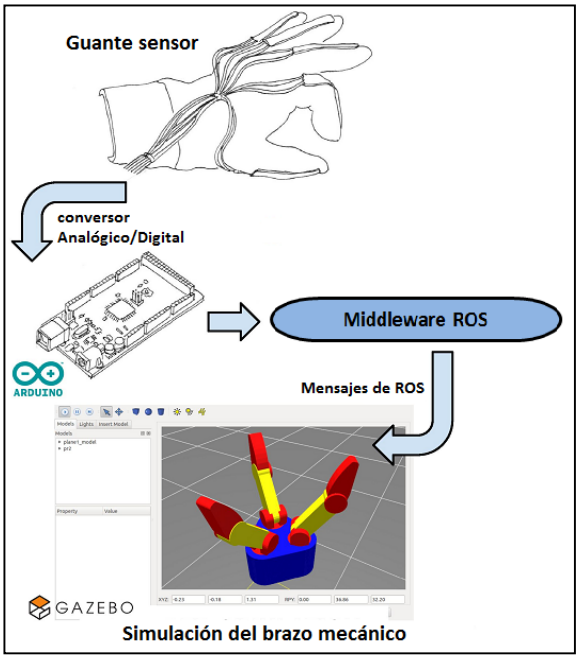

Fig. 11. Estructura del sistema propuesto en [21].

En sentido general, desde una visión más tecnológica, un Entorno Inteligente está compuesto por varios sensores, dispositivos personales, cámaras de video, y por una infraestructura computacional y de red [15], a través de los cuales se genera y transporta la información captada por los sensores, acerca del usuario y de su entorno, la que luego es procesada para finalizar el ciclo proveyendo el servicio al usuario. Por otro lado, es muy común encontrar la filosofía de Entornos Inteligentes ligada al Internet de las Cosas, que no es más que la idea de estar rodeados de objetos o dispositivos capaces de interactuar entre ellos y de cooperar con otros vecinos a través de un esquema común de comunicación (por ejemplo, a través de los protocolos de comunicación de Internet), con el fin de acometer una tarea específica [15, 23].

4.1.1 ROS como middleware para los entornos inteligentes

Partiendo de lo mencionado en el epígrafe anterior se puede concluir que los Entornos Inteligentes pueden ser tratados como aplicaciones distribuidas que requieren de un middleware que se encargue de gestionar y operar la cantidad de dispositivos y servicios que los conforman. En [24] y [25] se aborda el tema de los middleware en los EI y en el IoT respectivamente.

En [23] G. Fortino y colaboradores, realizaron una revisión y comparación de varios middlewares para EI, partiendo de una serie de requerimientos generales que deben cumplir estos para ser aplicados en dichos entornos. Entre los middlewares analizados en ese trabajo se encuentran iRoom [26], Aura [27], Context Toolkit [28], JCAF [29], Ambient Agoras [30], GAIA [31] y ROS.

Los requerimientos levantados para esa investigación fueron:

- *Abstracción sobre entradas y salidas heterogéneas de diferentes dispositivos de hardware (Req1, ver Tabla 2):* capacidad de ofrecer abstracción entre las diversas entradas y salidas de los dispositivos permitiéndoles interactuar con independencia del hardware.
- *Abstracción a través de interfaces de hardware y software (Req2):* permitir la interacción entre el hardware y el software con independencia de sus características (estandarización).
- *Abstracción de flujos de datos y tipos de datos (Req3):* formalizar los datos generados por los diferentes elementos que componen un EI mediante la abstracción de estos. Esto se puede hacer con un lenguaje como XML.
- *Abstracción sobre ubicación (Req4*): permitir abstracción para capturar y proveer información de ubicación y contexto de los objetos pertenecientes al entorno debido a que estos pueden estar ubicados tanto estática como dinámicamente así como estar referidos a uno o varios contextos [32].
- *Abstracción sobre el proceso de desarrollo (Req5):* Proveer herramientas y métodos que permitan modelar de manera eficaz los EI.

En la Tabla 2 se muestra de manera resumida el resultado de la evaluación realizada en dicho trabajo. Partiendo de estos resultados los autores concluyen que ROS es el middleware que mejor se adecua a sus necesidades, aunque destacan a iRoom y Aura por los resultados obtenidos en la comparación.

Tabla 2. Comparación entre los middlewares revisados en [23].

	Req1	**Req2**	**Req3**	**Req4**	**Req5**
ROS-player/stage	si	si	si	si	Si
iRoom	si	si	no	si	Si
Aura	si	si	no	no	Si
Context toolkit	si	si	no	no	Si
JCAF	si	si	no	no	Si
Gaia	si	si	no	no	Si
Ambient agoras	-	-	-	-	-

En [33] L. Roalter y colaboradores, evaluaron un conjunto de middlewares para realizar el diseño de un ejemplo de Escenario Inteligente. Entre estos se encontraban GAIA (ya mencionado), MundoCore [34] y ROS. Estos plantean que tanto GAIA como MundoCore habían sido propuestos anteriormente

como middlewares para la computación ubicua[7]; sin embargo estos presentan limitaciones en cuanto a la posibilidad de reutilización de código y la falta de una comunidad que les brinde soporte y los enriquezca con nuevos algoritmos y controladores. Teniendo en cuenta esto, proponen el uso de ROS y de la plataforma Player/Stage, este último antecesor de ROS para el entorno de la robótica.

Los elementos tenidos en cuenta por L. Roalter y colaboradores, para emplear un middleware del campo de la robótica para desarrollar un EI, fueron los siguientes:

- Los desafíos respecto a la heterogeneidad en cuanto a los dispositivos y las interfaces de software en el campo de la robótica son bastante parecidos a aquellos presentes en el contexto de los EI.
- El grado de madurez alcanzado por Player/Stage como middleware, el cual cuenta por más de diez años con una amplia comunidad de soporte y desarrollo.
- Conceptualmente un EI es muy similar a un robot estático e inamovible denominado Immobot [35], es decir, un EI puede ser tratado como un Immobot durante el desarrollo con ROS o Player/Stage.

De manera similar a [23], en este caso, L. Roalter y colaboradores terminaron decantándose por ROS debido a que es el sucesor natural de Player/Stage y porque también cuenta con una amplia comunidad de soporte.

- Ambos enfoques [23, 33] permiten justificar la elección de ROS como middleware para el desarrollo de EI, a pesar de la existencia en el estado del arte de otros middleware aptos para el desarrollo de dichos entornos.

4.1.2 Ejemplos de aplicación de ROS en entornos inteligentes

En [33] se expone el empleo de ROS como middleware para integrar sensores, accionadores y servicios de usuario como parte de un EI. El escenario de prueba consistió en dos oficinas equipadas con varios sensores tales como cámaras web, sensores infrarrojos para detectar movimiento, sensores de radiofrecuencia, de luz y de temperatura entre otros; los cuales se integran a un grupo de servicios web tales como RSS[8] y Twitter. También implementaron un conjunto de servicios contextuales [32] entre los que se encuentran fecha y hora, información meteorológica, entre otros. Notar que el escenario diseñado es de una alta complejidad y heterogeneidad. La construcción del escenario ubicado en la segunda oficina fue sobre la base de la posibilidad que brinda ROS para interconectar varios servidores máster por lo que la información de los servidores ubicados en ambas oficinas está disponible para cualquier consumidor dentro del entorno. Otro elemento que facilita la entrega de la información a los diversos consumidores presentes en dicho entorno es el esquema de intercambio de mensajes publicador/subscriptor de ROS (descrito en la sección 3.3.1), por otra parte, dicho esquema también les permitió integrar en tiempo real, es decir durante la implantación del escenario de prueba, los diversos servicios y sensores antes mencionados.

Entre las impresiones reportadas por L. Roalter y colaboradores en [33] sobresalen las positivas por encima de las negativas, respecto a estas últimas solo señalan algunos problemas con la documentación y algunos de los tutoriales disponibles en el sitio oficial de ROS. En cuanto a las impresiones positivas destacan la posibilidad que brinda el middleware de interconectar varios servidores máster lo cual facilita el diseño y desarrollo de escenarios más complejos dentro de los Entornos Inteligentes, también resaltan como las herramientas de simulación y visualización de dicho middleware permiten acelerar el proceso de desarrollo. Otro beneficio mencionado en [33] es la capacidad multi-lenguaje que ofrece ROS posibilitando desarrollar algoritmos en lenguajes eficientes como C, así como algoritmos de prueba y experimentación rápida haciendo uso de Python.

T. Misu y colaboradores propusieron en [36] un asistente de conducción inteligente denominado Townsurfer el cual está orientado a mejorar la experiencia de viaje en el interior de un automóvil. Dicha interacción está basada tanto en la situación contextual del conductor y su vehículo, tales como la ubicación del automóvil, la dirección de la mirada del conductor y la interacción natural entre el conductor

[7] Otra forma de referirse a los entornos inteligentes.
[8] http://www.rss.nom.es/

con el propio asistente. De forma general, el sistema propuesto consta de tres elementos fundamentales: 1) *reconocimiento automático del habla* con comprensión de lenguaje natural con el propósito de extraer la información semántica de las solicitudes del conductor; 2) *procesamiento de imágenes* para determinar hacia donde mira el conductor sobre la base de la dirección de la cabeza y; 3) *sistema de geo-localización* para determinar tanto la ubicación del vehículo en un mapa así como la trayectoria del viaje. Según [36] la combinación de los elementos antes mencionados garantiza que el sistema pueda responder de forma acertada las preguntas realizadas por el conductor respecto a los lugares que lo rodea.

En la Figura 12, se muestra el esquema del sistema distribuido propuesto en [36] el cual fue construido sobre la base del middleware ROS. Se puede apreciar que el sistema consta de tres entradas de datos o sensores principales: habla (micrófono), geo-ubicación (GPS (acrónimo de Sistema de Posicionamiento Global) y una Unidad de Medición de Inercia (IMU[9], por sus siglas en inglés)) y orientación de la mirada (sensor de profundidad[10]). Cada sensor está conectado a un nodo de ROS que se encarga de procesar la información captada.

En primer lugar, se tiene el nodo de reconocimiento del habla el cual es el encargado de iniciar la interacción entre los restantes nodos del sistema. En este se realiza el reconocimiento del habla y, en combinación con un intérprete de lenguaje natural, se extraen parejas de conceptos/valores [36] para determinar la semántica de cada expresión del usuario. El reconocimiento del habla es desempeñado a partir del Motor Julius de Reconocimiento del Habla [37] en combinación con dos modelos componentes generados a partir del HRItk [38]: un modelo de reconocimiento de la gramática y un diccionario de pronunciación. Por otro lado, el nodo de geo-ubicación es el encargado obtener información acerca de los puntos de interés (POI, por sus siglas en inglés) que rodean al usuario mientras que, el nodo de determinación de la dirección de la mirada (*gaze*, en inglés) realiza dicha función partiendo de una estimación de la orientación del rostro a través de una implementación para detectar rostros basada en la biblioteca PCL [39]. Por último, el nodo de detección de la mirada en combinación con el nodo de geo-ubicación le permiten al sistema generar una lista de POI partiendo de que la solicitud del usuario tenga expresiones de referencias, por ejemplo "¿Cuál es ese lugar ubicado a la derecha?", en este caso se generan todos los posibles candidatos ubicados a la derecha.

Otro ejemplo de Entorno Inteligente, en este caso, orientado a la agricultura, se expone en [40] mientras que en [41] se propone un juego de herramientas que permiten simplificar la investigación en los EI y facilitar la creación de estos. Este conjunto de herramientas está conformado por elementos de simulación, visualización y de generación de prototipos y de un middleware que lo complementaba, en este caso, ROS. En [42] presentan un proyecto denominado RUBICON con el cual se pretende mejorar la coordinación y adaptabilidad de una red de robots heterogéneos que cooperan entre sí con el objetivo de realizar tareas complejas que mejoren la experiencia del usuario en diversos entornos (ej. casas, hospitales). Para ello proponen la combinación a través de la plataforma ROS de métodos de robótica cognitiva, aprendizaje de máquinas, planificación, control basado en agentes y redes de sensores inalámbricos.

[9] Es un dispositivo electrónico que mide e informa acerca de la velocidad, orientación y fuerzas gravitacionales.
[10] Dispositivo que permite capturar imágenes en 3D.

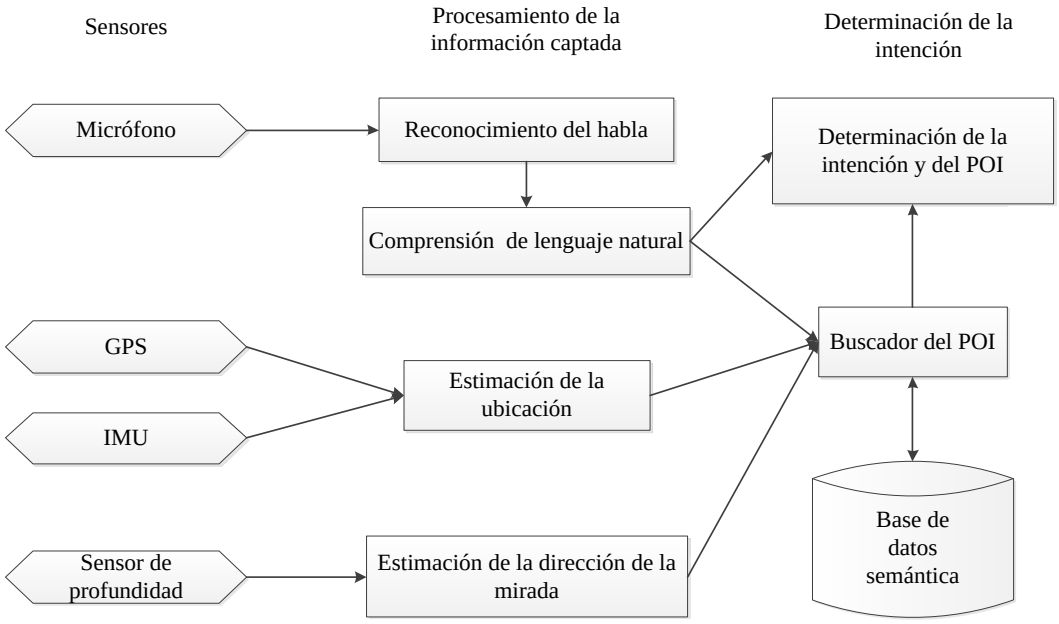

Fig. 12. Esquema de la aplicación Townsurfer.

4.2 Vigilancia

La vigilancia es el proceso de monitoreo enfocado, sistemático y rutinario del comportamiento, actividades o cambio de la información relativa a las personas, con el propósito de influenciar, gestionar, proteger o detectarlas [43]. Pudiera agregarse a esta definición, teniendo en cuenta el contexto actual, la posibilidad de monitorear la actividad o comportamiento de procesos y otras tecnologías que interactúan entre ellas (IoT, por ejemplo) o con los usuarios (en un EI), además deben considerarse los entornos que rodean a las personas independientemente de la presencia de estas en el lugar (por ejemplo, para la detección de explosivos).

En la actualidad, la actividad de los sistemas de vigilancia está altamente automatizada debido al desarrollo vertiginoso de las tecnologías vinculadas. Por ejemplo, para desempeñar la actividad de monitoreo se emplean redes de sensores, cámaras de video, vehículos aéreos no tripulados (UAV por sus siglas en inglés), dispositivos de radiofrecuencia, sensores móviles, entre otros; los cuales, generalmente, están integrados a procesos automatizados que se encargan de procesar y analizar la información generada por estos con el fin, por ejemplo, de detectar eventos asociados a la comisión de delitos informáticos.

Los procesos antes mencionados pudieran ser módulos donde se ejecutan algoritmos de reconocimiento de rostro, detección de objetos o seguimiento de personas, por citar algunos ejemplos. El tipo de proceso va a depender del propósito de la vigilancia.

Una práctica muy común durante el diseño de un sistema de vigilancia es combinar distintas técnicas de monitoreo y tipos de procesos con el fin de obtener sistemas más efectivos y robustos. Teniendo esto en cuenta, un sistema de vigilancia puede ser entendido como un sistema distribuido en donde se integren todos los elementos antes mencionados (hardware y procesos) con el fin de realizar alguna de las tareas de la vigilancia. .

En la siguiente sección se presentan algunos ejemplos de sistemas para la vigilancia desarrollados a partir de uso de ROS como middleware.

4.2.1 Ejemplos de aplicación de ROS en sistemas para la vigilancia

Lovell y colaboradores proponen en [44] una plataforma de vigilancia para realizar la detección y rastreo de personas en aeropuertos combinando técnicas biométricas y la infraestructura tecnológica de estos,

específicamente, la red de cámaras de video vigilancia de alta definición (HD, por sus siglas en inglés) las cuales brindan imágenes de elevada resolución que, a su vez, permiten realizar una mejor detección del sujeto en cuestión.

La idea principal expuesta fue la de diseñar un sistema distribuido que fuese capaz de soportar exigencias tales como realizar la detección de personas en un entorno multitudinario y en tiempo real, capacidad para manejar las características del flujo de video, de manejar fuentes simultaneas de video y la naturaleza propia de la vigilancia en un aeropuerto (24/7). Teniendo en cuenta dichas exigencias Lovell y colaboradores desarrollaron un sistema denominado Motor de Búsqueda de Rostros (FSE, por sus siglas en inglés) el cual está compuesto por varios componentes tales como gestores de cámaras IP, algoritmos de detección de rostro (técnica biométrica escogida para realizar la detección de personas), bases de datos e integración con plataformas móviles. ROS fue el middleware escogido para implementar y hacer interactuar los componentes debido a las facilidades de comunicación uno-a-muchos y muchos-a-muchos que brinda el esquema de comunicación publicador/subscriptor. De forma general, ROS permitió realizar búsquedas de rostros en tiempo real a través de la red de cámaras. En la Figura 13 se muestran los nodos que conforman el FSE.

Entre los elementos que caracterizan el sistema se encuentran la tecnología de reconocimiento de rostro desarrollada por el Grupo de Video Vigilancia Avanzada del NICTA[11] la cual, está basada en la técnica Histogramas de Multiregión (MRH, por sus siglas en inglés) [45], técnicas de aceleración de GPU empleando CUDA y bases de datos distribuidas y mapeadas en memoria con el fin de aumentar la velocidad del proceso de reconocimiento de rostros.

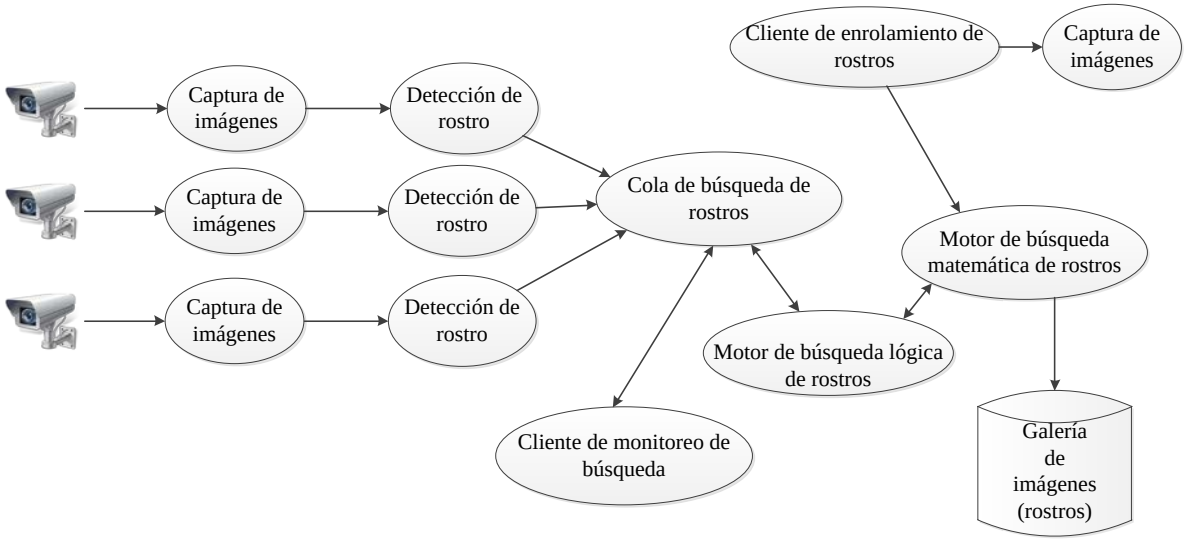

Fig. 13. Diagrama de nodos del sistema propuesto en [44].

Lovell y colaboradores lograron validar su propuesta mediante una prueba realizada en un aeropuerto, la cual consistió en identificar un total de 15 personas de interés durante una ventana de tiempo de 120 minutos. No se incluyeron apreciaciones respecto al desempeño del middleware ROS en los resultados reportados, sin embargo, plantean trabajar en un futuro nuevamente con dicho middleware. En [46] se presenta un sistema similar (ver Fig. 14) al desarrollado por el mismo grupo de especialistas, cuya arquitectura y objetivos son similares a los del experimento presentado en [44].

[11] http://www.nicta.com.au/

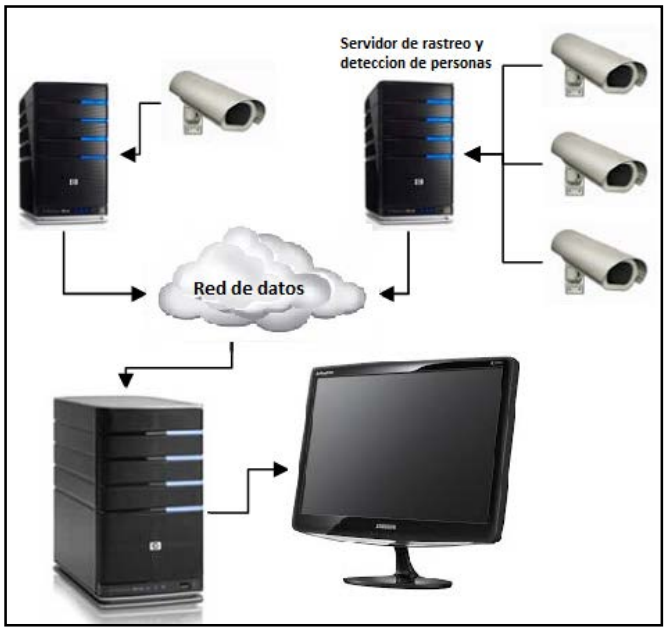

Fig. 14. Configuración de hardware del sistema propuesto en [46].

En [47], L. Iocchi y colaboradores, presentan un estudio para mejorar la vigilancia en entornos públicos y multitudinarios. Para ello, los autores proponen un sistema complejo de vigilancia conformado por tres componentes fundamentales: personal de seguridad portando sensores (por ejemplo, PDAs[12] o teléfonos móviles inteligentes), robots móviles portando sensores y una red de sensores fijos tales como cámaras de video. La idea general es la de procesar toda la información captada por la red de sensores con el propósito de detectar e identificar objetos y/o personas de interés en escenarios complejos de vigilancia.

Como detalle interesante, debe destacarse que, al enfocar el diseño de la arquitectura de vigilancia los desarrolladores priorizaron los requerimientos de modularidad, basándose en una concepción jerárquica. De esta forma, el sistema distribuido resultante propuesto en [47] consta de cuatro módulos fundamentales: 1) *red de sensores* la cual está conformada por nodos sensores interconectados a través de una red inalámbrica y cada nodo consta de un grupo de sensores; 2) *intérprete del escenario* conformado por algoritmos y técnicas que permiten realizar un análisis de escenarios complejos y obtener información de interés; 3) *sistema de control multi-robot* para controlar los robots o plataformas móviles equipadas con sensores; y 4) un *sistema de ubicación del personal de seguridad* encargado de localizar y alertar a todos los elementos móviles de seguridad presentes en el sistema.

De forma general cada módulo está representado dentro ROS por un conjunto de nodos y servicios encargados gestionar cada sensor de la red y de manejar la información generada por cada uno de estos. Dicha información es entregada a un módulo de análisis de video que realiza la detección de objetos o personas además de dar seguimiento a objetivos de interés. En la Figura 15 se muestra un ejemplo de los nodos encargados de manejar una cámara dentro de ROS. En el módulo de análisis de video se ejecutan los algoritmos para realizar la detección y rastreo de objetos o personas de interés.

[12] Asistente personal digital, dispositivo electrónico digital que puede ser una agenda o un organizador personal.

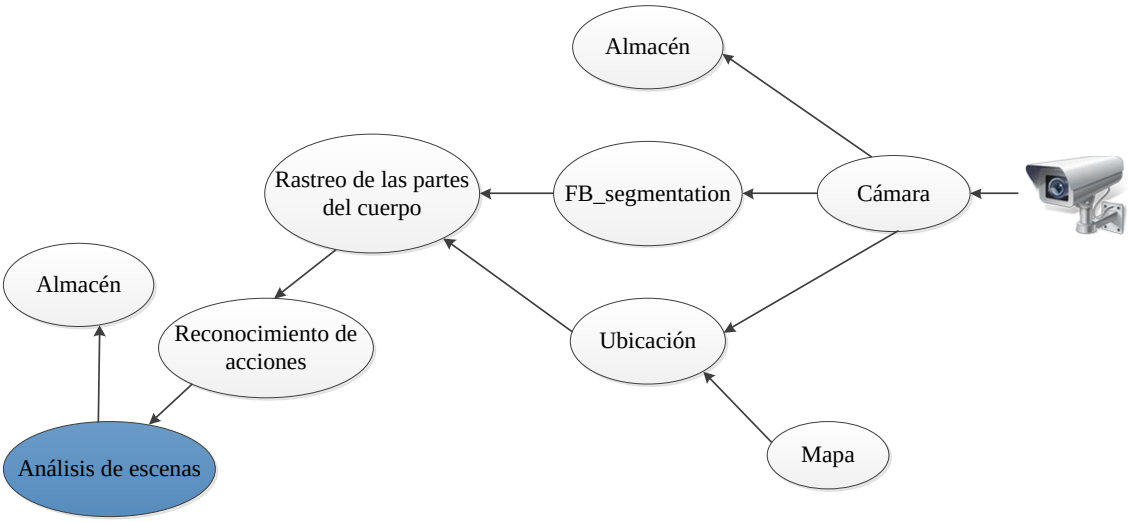

Fig. 15. Diagrama de nodos que operan una cámara de video.

Para validar el sistema, L. Iocchi y colaboradores desplegaron una red de sensores en la zona de equipajes de un aeropuerto con el fin de aprovechar la gran cantidad de pasajeros que circulan por esa zona. Emplearon dos robots para patrullar los cuales se comunican con una cámara fija y con un operador que es el encargado de procesar todo el flujo de video. La cámara fija le indica a los robots la zona de interés hacia la cual estos se deben desplazar. También hicieron uso de una cámara fija que opera en una ubicación remota la cual también entrega información al operador antes mencionado. Una vez que es procesada toda la información resultante del monitoreo, el resultado se le envía a un operador móvil a través de un PDA. En la Figura 16 se puede ver una representación esquemática de dicha prueba. Una idea similar a la anterior se expone en [48] y [49]. Es válido precisar que esta prueba fue realizada haciendo uso únicamente del flujo de video generado por dos cámaras fijas quedándose fuera del alcance de la misma el manejo e integración de la información generada por otro grupo de sensores que forman parte del diseño original de la arquitectura. Tampoco el número de sensores fue lo suficientemente grande como para valorar la respuesta o desempeño del middleware operando en escenarios con una alta carga de elementos.

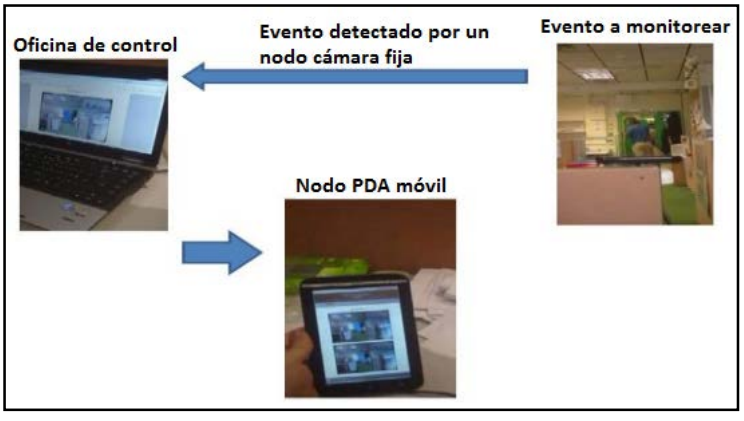

Fig. 16. Esquema de funcionamiento del sistema de video vigilancia presentado en [47].

E. Pereira y colaboradores presentaron en [50] una arquitectura para monitorear y detectar derrames de petróleo en el mar (ver Fig. 17). Para esto se apoyaron en varios elementos físicos tales como sensores móviles (UAV con cámaras en este caso), estaciones de control terrestres, *drifters* (tipo de boyas

equipadas con sensores) equipados con un Sistema Automático de Identificación [51] (AIS, por sus siglas en inglés) y embarcaciones.

Otra característica del sistema mencionado es la variedad de mecanismos de comunicación inalámbrica presentes (WiFi o canal AIS). Por otro lado, implementaron algoritmos capaces de procesar la información proveniente de los elementos físicos y, posteriormente, de tomar decisiones. Para hacer interactuar el mundo físico con el lógico hicieron uso de ROS como middleware. Otros ejemplos del empleo de ROS en el campo de la vigilancia haciendo uso de UAV se pueden encontrar en [52, 53].

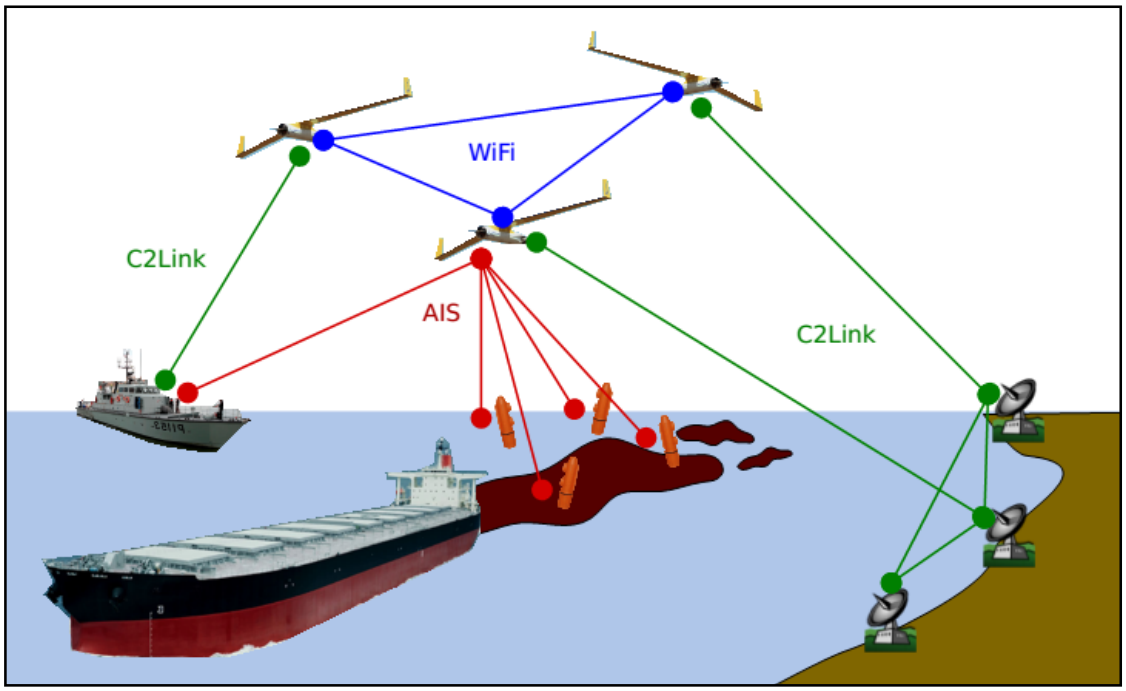

Fig. 17. Despliegue del sistema de monitoreo propuesto en [50].

En [54] presentan un sistema para ubicar personas y seguir su orientación partiendo del seguimiento de sensores portables que estos lleven consigo y de una red de cámaras 3D. Todo esto con el fin de monitorear un síntoma muy frecuente en los pacientes que padecen de la enfermedad de Parkinson. Para realizar el monitoreo de los pacientes combinaron la información transmitida por el sensor portable, que no es más que un *Smartphone*, y emplearon dos cámaras Kinect, mientras que para realizar la estimación de la posición y la orientación de los pacientes, implementaron algoritmos de desarrollo propio apoyándose en las bibliotecas OpenCV [55] y PCL. Los elementos antes mencionados fueron integrados a través del middleware ROS debido a que este ya provee las bibliotecas antes mencionadas, así como herramientas de compresión de imágenes y de grabación de datos que permiten almacenar de forma eficiente y sincronizada toda la información producida por los sensores que conforman el sistema distribuido. De la misma forma que en el ejemplo anterior las pruebas realizadas con el sistema propuesto en [54] abarcan pocos elementos generadores de información por lo cual en los resultados expuestos en dicho trabajo no se reportaron dificultades con el middleware utilizado. Otras aplicaciones orientadas a la vigilancia del estado físico o de salud de las personas y que han sido desarrolladas sobre la base del middleware ROS se pueden encontrar en [56, 57].

5 Prueba de concepto de ROS

El trabajo con los sistemas de video vigilancia haciendo uso de cámaras IP y las posibilidades para la detección de personas de interés, analizando las imágenes recibidas de las cámara mediante técnicas de reconocimiento de rostros, es de gran interés para los autores de este reporte, por lo que se implementó a modo de prueba un sistema distribuido basado en ROS con el fin de realizar la tarea de detectar personas de interés haciendo uso de una red de cámaras IP.

La técnica empleada para la detección de las personas fue Reconocimiento de Rostro (Face Recognition, en inglés), y el algoritmo utilizado para las pruebas fue tomado del paquete face_recognition[13] perteneciente al repositorio de ROS. También se hizo uso de un controlador de ROS para el manejo de las cámaras IP del fabricante AXIS[14]. La configuración del hardware se puede apreciar en la Figura 18. Las características de la PC donde se implementó ROS se muestran en la tabla siguiente.

Tabla 3. Características de la PC donde se instaló el sistema de prueba.

	CPU	**RAM**	**Sistema Operativo**
PC Intel (R) Core (TM) 2 Quad	Q9450 @ 2.66 GHz	2 GB	Ubuntu 12.04 LTS

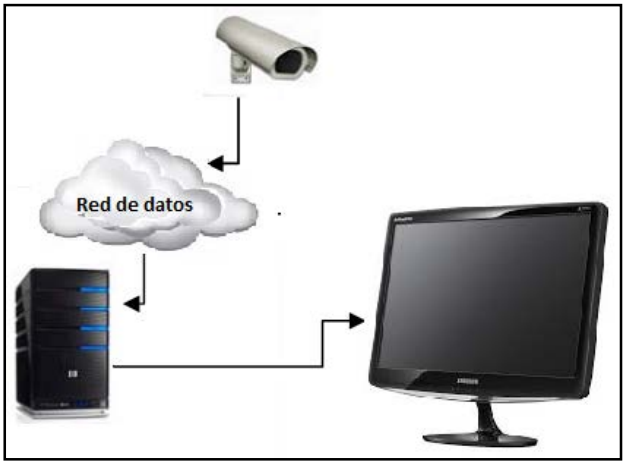

Fig. 18. Configuracion de hardware del sistema de prueba.

La arquitectura lógica del sistema quedó como se muestra en la Figura 19. En esta imagen se pueden apreciar los nodos del sistema y la interacción que existe entre ellos mediante tópicos y servicios. El paquete face_reognition está conformado por dos nodos denominados Fserver y Fclient. Fserver desempeña las funcionalidades de detección y reconocimiento de rostros a partir de un flujo de video. Para ello publica los tópicos /fr_order (por aquí recibe un mensaje que le indica que acción debe realizar) y se subscribe al tópico publicado por el controlador de la cámara ip (/camera_raw) para recibir el flujo de video.

Por otro lado, el nodo Fclient esta entendido para realizar pruebas o realizar demostraciones del funcionamiento del Fserver. Este último se comunica con el nodo Fserver a través de una llamada de servicio con retroalimentación [16]; esto es: el nodo Fclient le envía una solicitud de servicio, por ejemplo, de adquirir imágenes para generar la base de datos; al nodo Fserver y este devuelve a Fclient el estado de realización de la tarea en forma de retroalimentación. Una vez concluida la tarea Fserver

[13] http://wiki.ros.org/face_recognition
[14] http://www.axis.com/es/

devuelve el resultado al nodo Fclient y este se encarga de mostrarlo al usuario. Tal y como se había comentado en la sección 2 de este trabajo, todos los nodos deben registrarse en el nodo máster y exponerle sus tópicos y servicios a este.

La base de datos con las imágenes de las personas enroladas que se deseaban autenticar fue generada haciendo uso de las mismas cámaras IP y del nodo Fserver.

La prueba consistió de hacer uso de una cámara IP para autenticar a tres personas cuyas imágenes fueron tomadas por la misma cámara para generar la base de datos. En la Figura 20 se pude apreciar lo que muestra el nodo Fclient una vez autenticada la persona.

Esta prueba permitió probar y demostrar que ROS puede ser empleado como middleware en un escenario de vigilancia integrado por sensores fijos, en este caso, cámaras IP y por procesos que permitieron realizar la autenticación de un grupo de personas de interés.

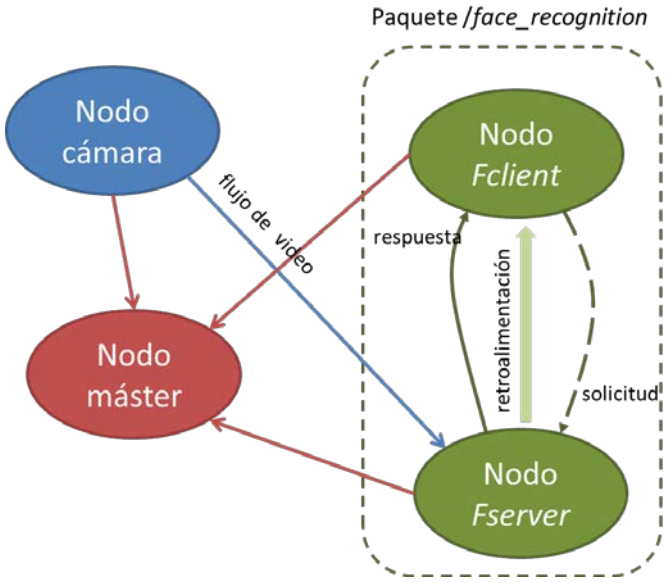

Fig. 19. Arquitectura lógica del sistema de prueba.

6 Conclusiones

La filosofía de sistemas distribuidos constituye un paradigma para el desarrollo de aplicaciones que requieran de integrar subgrupos de otras aplicaciones para acometer una tarea específica como parte del sistema global, mientras que el concepto de middleware constituye la piedra angular dentro de dicho paradigma. En este reporte se brinda una visión acerca de las posibilidades de ROS como middleware para desarrollar sistemas distribuidos tanto en el campo de la robótica como fuera de este. En función de este objetivo, se desarrolló una prueba de concepto que permitió comprobar las facilidades de implementación que brinda este middleware, así como las potencialidades de su repositorio para tareas no orientadas a entornos robóticos, aspecto que se validó con el empleo del módulo de reconocimiento de rostro y los controladores de las cámaras de video encontrados en dicha biblioteca.

De forma particular se pueden resaltar varios elementos positivos y negativos respecto a ROS partiendo de lo presentado en este trabajo. Empezando con los aspectos negativos cabe destacar que en la literatura consultada existen pocos reportes de problemáticas o dificultades presentadas con el empleo de dicho middleware así como con su desempeño. Esto se debe, en buena parte, a que las aplicaciones revisadas dentro de dicha literatura no son de una alta complejidad en cuanto a la cantidad de elementos, tanto de hardware como de software, que estas contienen. Sin embargo, en [15] plantean una serie de

limitaciones que pudiera presentar ROS cuando el sistema puede estar conformado por un alto número de elementos:

1. Esquema de funcionamiento limitado por el hecho de que un solo nodo (nodo máster) es el encargado de gestionar todas las comunicaciones y el intercambio de la información lo cual, a su vez, constituye una grave dificultad a la hora de operar en entornos que presenten gran número de piezas de hardware.
2. Baja disponibilidad ya que si el nodo máster deja de funcionar no se podrán establecer nuevas comunicaciones entre los componentes del sistema.
3. La comunicación directa que se establece entre subscriptores y publicadores puede generar una sobrecarga del ancho de banda en el lado del publicador si muchos subscriptores intentan acceder a la vez a este.

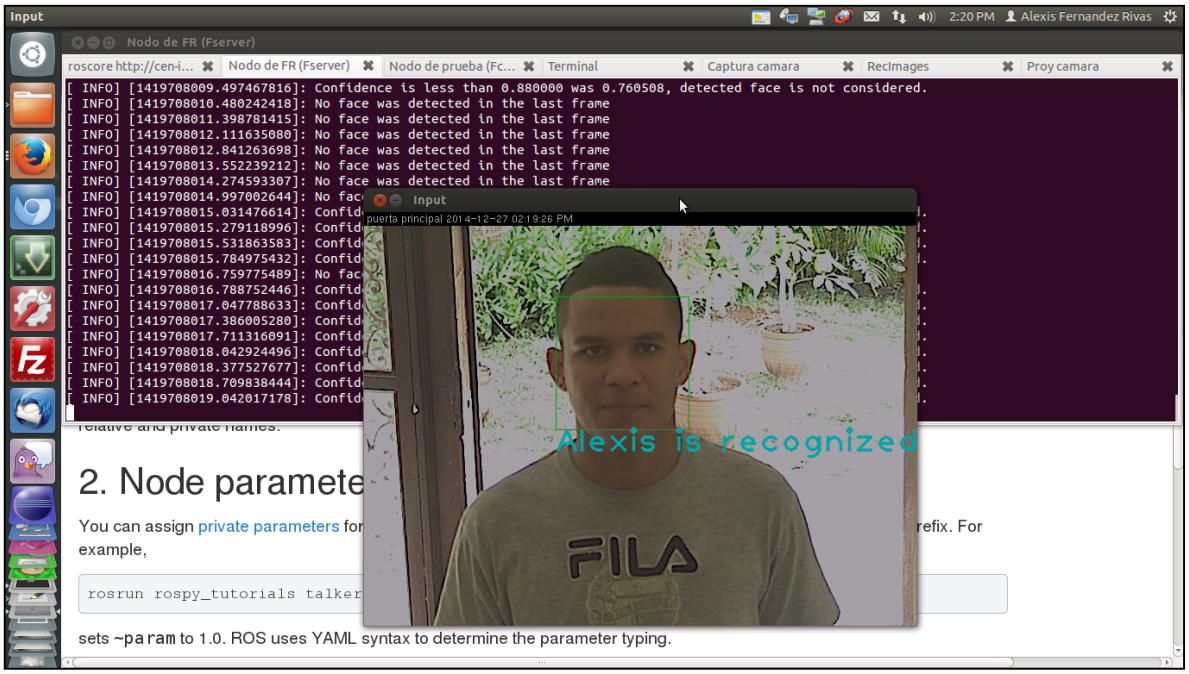

Fig. 20. Salida gráfica del nodo Fclient: autenticación de la persona.

Respecto a las limitaciones antes mencionadas, en [15] se proponen una serie de adecuaciones para ROS en función de erradicarlas.

Por otro lado, en [58] y [59] exponen algunas vulnerabilidades que presenta dicho middleware tales como: comunicación entre nodos realizada en texto plano, empleo de puertos TCP desprotegidos para la comunicación interna, vulnerable al *spoofing* de paquetes [6] así como a la inyección de paquetes ROS falsos.

ROS presenta una serie de aspectos positivos que lo hacen muy atractivo para ser empleado como middleware en ámbitos fuera del marco de la robótica: alto nivel integración de hardware, provee un repositorio de aplicaciones y un conjunto de herramientas que facilitan el proceso de desarrollo del sistema, soporte para procesamiento de imágenes mediante bibliotecas conocidas tales como PCL y OpenCV, esquema de comunicación que permite comunicar asíncronamente, tanto de modo uno-a-muchos como muchos-a-muchos, componentes dentro de un sistema distribuido, entre otras ventajas.

Del estudio de la bibliografía, se pudo comprobar la cantidad de aplicaciones para la vigilancia que se pueden desarrollar haciendo uso de este middleware específicamente dentro del ámbito de la video vigilancia, ya sea con sensores móviles o estáticos. A lo que se le agrega la posibilidad de usar estos sensores conjuntamente con algoritmos propios de reconocimiento de rostro o de objetos para obtener un

sistema avanzado para la vigilancia que permita, entre otras cosas, identificar, autenticar o seguir personas de interés en entornos complejos para la seguridad como pueden ser los aeropuertos.

6.1 Recomendaciones y posible trabajo futuro

De proponerse continuar el estudio sobre la aplicabilidad de ROS, por ejemplo, como base para un sistema de video vigilancia en frontera, manejando múltiples cámaras de video y aplicando algoritmos para el reconocimiento y seguimiento de personas (por la forma de caminar, el rostro, etc.), será necesario realizar nuevos experimentos distribuyendo físicamente el sistema (hardware y software), para determinar su comportamiento, en términos de eficiencia, al gestionar un número elevado de sensores y procesos. Estos experimentos permitirán determinar, además, el tipo de configuración y las arquitecturas adecuadas para enfrentar tareas similares con los recursos tecnológicos disponibles (cantidad de memoria, CPUs, ancho de banda, etc.).

Al desarrollo de un posible sistema avanzado de video vigilancia se le pueden incorporar las experiencias relativas al empleo de ROS en el campo de los EI. También sería apropiado realizar un estudio acerca de otros middlewares de código abierto que hayan sido empleados en el desarrollo de plataformas de video vigilancia.

Teniendo en cuenta el presente estudio realizado sobre ROS, los ejemplos de aplicabilidad y los intereses del centro, entre las posibles líneas de trabajo futuro pudieran estar las siguientes:

1. Estudiar la posibilidad de utilizar a ROS como middleware para el desarrollo de un sistema de análisis de datos para la ciberseguridad combinando técnicas de minería de datos.
2. Estudiar el desempeño de ROS en escenarios donde el número de sensores generando información sea superior a las cantidades reportadas en el estado del arte.
3. Estudiar el desempeño de ROS durante el trabajo con bases de datos que contengan un gran volumen de información.
4. Estudiar la posibilidad de realizar procesamiento paralelo en el ámbito del middleware ROS.
5. Identificar vulnerabilidades relacionadas con la seguridad dentro de la arquitectura ROS que no estén reportadas en el estado del arte.

7 Referencias bibliográficas

1. Puder, A., Römer, K., Pilhofer, F.: Distributed systems architecture: a middleware approach. Elsevier (2006).
2. Jingyong, L., Yong, Z., Yong, C., Lichen, Z.: Middleware-based distributed systems software process. In: Proceedings of the 2009 International Conference on Hybrid Information Technology. pp. 345–348. ACM (2009).
3. Emmerich, W.: Software engineering and middleware: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering. pp. 117–129. ACM (2000).
4. Bernstein, P.A.: Middleware: a model for distributed system services. Commun. ACM. 39, 86–98 (1996).
5. Bishop, T.A., Karne, R.K.: A Survey of Middleware. In: Computers and Their Applications. pp. 254–258. Citeseer (2003).
6. Peterson, L.L., Davie, B.S.: Computer Networks: A Systems Approach. Elsevier (2011).
7. Curry, E.: Message-oriented middleware. Middlew. Commun. 1–28 (2004).
8. Vallecillo, A., others: RM-ODP: The ISO Reference Model for Open Distributed Processing. DINTEL Ed. Softw. Eng. 3, 66–69 (2001).
9. Park, K.I.: QoS in Packet Networks. Springer Science & Business Media (2004).
10. Stallings, W.: Operating Systems: Internals and Design Principles. Prentice Hall (2009).
11. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. In: ICRA workshop on open source software. p. 5 (2009).
12. O'Kane, J.M.: A Gentle Introduction to ROS. Independently published (2013).
13. Steinmetz, R.: Peer-to-Peer Systems and Applications. Springer Science & Business Media (2005).

14. Arnal Benedicto, L.: Detección de personas con visión artificial y sensores de rango, https://riunet.upv.es/handle/10251/27697, (2013).
15. Schneider, T.: Distributed Networks Using ROS–Cross-Network Middleware Communication using IPv6. (2012).
16. Mead, R., Zeltner, S., Matarić, M.J.: Integrating ROS into Educational Robotics: Bridging the Gap between Grade School and Grad School. (2013).
17. Elkady, A., Sobh, T.: Robotics middleware: A comprehensive literature survey and attribute-based bibliography. J. Robot. 2012, (2012).
18. Magyar, G., Sinčák, P., Krizsán, Z.: Comparison Study of Robotic Middleware for Robotic Applications. In: Emergent Trends in Robotics and Intelligent Systems. pp. 121–128. Springer (2015).
19. Astua, C., Barber, R., Crespo, J., Jardon, A.: Object detection techniques applied on mobile robot semantic navigation. Sensors. 14, 6734–6757 (2014).
20. Romano, J.M., Brindza, J.P., Kuchenbecker, K.J.: ROS open-source audio recognizer: ROAR environmental sound detection tools for robot programming. Auton. Robots. 34, 207–215 (2013).
21. Zubrycki, I., Granosik, G.: Test setup for multi-finger gripper control based on robot operating system (ROS). In: Robot Motion and Control (RoMoCo), 2013 9th Workshop on. pp. 135–140. IEEE (2013).
22. Cook, D.J., Das, S.K.: How smart are our environments? An updated look at the state of the art. Pervasive Mob. Comput. 3, 53–73 (2007).
23. Fortino, G., Guerrieri, A., Russo, W., Savaglio, C.: Middlewares for smart objects and smart environments: Overview and comparison. In: Internet of Things Based on Smart Objects. pp. 1–27. Springer (2014).
24. Raychoudhury, V., Cao, J., Kumar, M., Zhang, D.: Middleware for pervasive computing: A survey. Pervasive Mob. Comput. 9, 177–200 (2013).
25. Chaqfeh, M., Mohamed, N., others: Challenges in middleware solutions for the internet of things. In: Collaboration Technologies and Systems (CTS), 2012 International Conference on. pp. 21–26. IEEE (2012).
26. Brooks, R.A.: The Intelligent Room project. In: , Second International Conference on Cognitive Technology, 1997. Humanizing the Information Age. Proceedings. pp. 271–278 (1997).
27. Sousa, J.P., Garlan, D.: Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments. In: Bosch, J., Gentleman, M., Hofmeister, C., and Kuusela, J. (eds.) Software Architecture. pp. 29–43. Springer US (2002).
28. Dey, A.K., Abowd, G.D., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Hum.-Comput. Interact. 16, 97–166 (2001).
29. Bardram, J.E.: The Java Context Awareness Framework (JCAF) – A Service Infrastructure and Programming Framework for Context-Aware Applications. In: Gellersen, H.-W., Want, R., and Schmidt, A. (eds.) Pervasive Computing. pp. 98–115. Springer Berlin Heidelberg (2005).
30. Streitz, N., Prante, T., Röcker, C., Van Alphen, D., Magerkurth, C., Stenzel, R., Plewe, D.A.: Ambient displays and mobile devices for the creation of social architectural spaces. In: Public and Situated Displays. pp. 387–409. Springer (2003).
31. Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H., Nahrstedt, K.: A middleware infrastructure for active spaces. IEEE Pervasive Comput. 1, 74–83 (2002).
32. Schmidt, A., Beigl, M., Gellersen, H.-W.: There is more to context than location. Comput. Graph. 23, 893–901 (1999).
33. Roalter, L., Kranz, M., Möller, A.: A middleware for intelligent environments and the internet of things. In: Ubiquitous Intelligence and Computing. pp. 267–281. Springer (2010).
34. Aitenbichler, E., Kangasharju, J., Mühlhäuser, M.: MundoCore: A light-weight infrastructure for pervasive computing. Pervasive Mob. Comput. 3, 332–361 (2007).
35. Goldman, R.P., Baral, C.: Robots, softbots, immobots: The 1997 AAAI Workshop on Theories of Action, Planning and Control. Knowl. Eng. Rev. 13, 179–184 (1998).
36. Misu, T., Raux, A., Lane, I., Devassy, J., Gupta, R.: Situated multi-modal dialog system in vehicles. In: Proceedings of the 6th workshop on Eye gaze in intelligent human machine interaction: gaze in multimodal interaction. pp. 25–28. ACM (2013).
37. Open-Source Large Vocabulary CSR Engine Julius, http://julius.osdn.jp/en_index.php. Accedido: 29 de Octubre del 2015.
38. Lane, I., Prasad, V., Sinha, G., Umuhoza, A., Luo, S., Chandrashekaran, A., Raux, A.: HRItk: the human-robot interaction ToolKit rapid development of speech-centric interactive systems in ROS. In: NAACL-HLT Workshop on Future Directions and Needs in the Spoken Dialog Community: Tools and Data. pp. 41–44. Association for Computational Linguistics (2012).

39. PCL - Point Cloud Library (PCL), http://pointclouds.org/. Accedido: 29 de Octubre del 2015.
40. Correll, N., Arechiga, N., Bolger, A., Bollini, M., Charrow, B., Clayton, A., Dominguez, F., Donahue, K., Dyar, S., Johnson, L., others: Indoor robot gardening: design and implementation. Intell. Serv. Robot. 3, 219–232 (2010).
41. Roalter, L., Möller, A., Diewald, S., Kranz, M.: Developing intelligent environments: A development tool chain for creation, testing and simulation of smart and intelligent environments. In: Intelligent Environments (IE), 2011 7th International Conference on. pp. 214–221. IEEE (2011).
42. Amato, G., Bacciu, D., Broxvall, M., Chessa, S., Coleman, S., Di Rocco, M., Dragone, M., Gallicchio, C., Gennaro, C., Lozano, H., others: Robotic ubiquitous cognitive ecology for smart homes. J. Intell. Robot. Syst. 1–25 (2015).
43. Lyon, D.: Surveillance Studies: An Overview. Polity (2007).
44. Lovell, B.C., Bigdeli, A., Mau, S.: Invited paper: Embedded face and biometric technologies for national and border security. In: 2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). pp. 117–122 (2011).
45. Sanderson, C., Lovell, B.C.: Multi-region probabilistic histograms for robust and scalable identity inference. In: Advances in Biometrics. pp. 199–208. Springer (2009).
46. Dadgostar, F., Bigdeli, A., Mau, S., Smith, T., Lovell, B.: A Framework for Lab-based Real-time Video Analysis on Distributed Camera Networks. In: Proceedings of the Fourth ACM/IEEE International Conference on Distributed Smart Cameras. pp. 238–242. ACM, New York, NY, USA (2010).
47. Iocchi, L., Monekosso, N.D., Nardi, D., Nicolescu, M., Remagnino, P., Valera, M.: Smart Monitoring of Complex Public Scenes. In: AAAI Fall Symposium: Robot-Human Teamwork in Dynamic Adverse Environment (2011).
48. Tian, S., Saitov, D., Lee, S.G.: Cloud robot with real-time face recognition ability. Adv Sci Technol Lett. 51, 77–80 (2014).
49. Munaro, M., Basso, F., Michieletto, S., Pagello, E., Menegatti, E.: A software architecture for RGB-D people tracking based on ros framework for a mobile robot. In: Frontiers of Intelligent Autonomous Systems. pp. 53–68. Springer (2013).
50. Pereira, E., da Silva, P.M., Krainer, C., Kirsch, C.M., Morgado, J., Sengupta, R.: A Networked Robotic System and its Use in an Oil Spill Monitoring Exercise. (2013).
51. Automatic Identification Systems (AIS), http://www.imo.org/en/OurWork/Safety/Navigation/Pages/AIS.aspx. Accedido: 27 de Junio del 2015.
52. Capitan, J., Merino, L., Ollero, A.: Cooperative Decision-Making Under Uncertainties for Multi-Target Surveillance with Multiples UAVs. J. Intell. Robot. Syst. 1–16 (2015).
53. Navarro González, J.: People exact-tracking using a Parrot AR. Drone 2.0. (2015).
54. Takač, B., Català, A., Martín, D.R., Van Der Aa, N., Chen, W., Rauterberg, M.: Position and orientation tracking in a ubiquitous monitoring system for parkinson disease patients with freezing of gait symptom. JMIR MHealth UHealth. 1, (2013).
55. ABOUT | OpenCV, http://opencv.org/about.html. Accedido: 29 de Octubre del 2015.
56. Takac, B., Catala, A., Rauterberg, M., Chen, W.: People identification for domestic non-overlapping RGB-D camera networks. In: Multi-Conference on Systems, Signals & Devices (SSD), 2014 11th International. pp. 1–6. IEEE (2014).
57. OLIVER CHIVA, E.: Desarrollo de un sistema de monitorización de parámetros biomédicos basado ROS sobre una plataforma empotrada, https://riunet.upv.es/handle/10251/53639, (2015).
58. McClean, J., Stull, C., Farrar, C., Mascareñas, D.: A preliminary cyber-physical security assessment of the Robot Operating System (ROS). In: SPIE Defense, Security, and Sensing. pp. 874110–874110. International Society for Optics and Photonics (2013).
59. Alemzadeh, H., Chen, D., Cao, P.M., Lewis, A., Kalbarczyk, Z.T., Iyer, R.K.: Targeted Attacks on Control System of A Teleoperated Surgical Robot. (2015).

RT_033, febrero 2016
Aprobado por el Consejo Científico CENATAV

Editor: Lic. Lucía González Bayona
Diseño de Portada: Di. Alejandro Pérez Abraham
RNPS No. 2143
ISSN 2072-6260
Indicaciones para los Autores:
Seguir la plantilla que aparece en www.cenatav.co.cu
C E N A T A V
7ma. A No. 21406 e/214 y 216, Rpto. Siboney, Playa;
La Habana. Cuba. C.P. 12200
Impreso en Cuba