https://doi.org/10.17979/spudc.9788497498081.0884

MAS-RECON. MIDDLEWARE RECONFIGURABLE BASADO EN MULTIAGENTES

U. Gangoiti*, A. Armentia*, R. Priego*, E. Estévez**, M. Marcos*
*Dept. Ingeniería de Sistemas y Automática, UPV/EHU, España
e-mail: {unai.gangoiti, aintzane.armentia, rafael.priego, marga.marcos}@ehu.eus
**Dept. Ingeniería Electrónica y Automática EPS de Jaén, España
e-mail: eestevez@ujaen.es

Resumen

Los avances tecnológicos en el ámbito del control y la automatización han permitido la puesta en marcha de las llamadas aplicaciones sensibles al contexto. Estas aplicaciones demandan flexibilidad para evolucionar a medida que lo haga el contexto (adaptabilidad), así como para evitar interrupciones del servicio en el caso de fallos en nodos (disponibilidad). La solución propuesta en este trabajo hace frente, entre otros, a estos requisitos de flexibilidad mediante un middleware basado en multiagentes que además de cumplir con los requisitos de adaptabilidad asegura la disponibilidad del sistema incluso para aplicaciones con estado.

Palabras Clave: adaptabilidad, disponibilidad, middleware, sistema multiagente.

1 INTRODUCCIÓN

Las aplicaciones sensibles al contexto no solo monitorizan su entorno físico para procesar la información capturada, sino que también deben evolucionar para adaptarse a cambios en él y/o incluso intervenir en su comportamiento. Se ejecutan comúnmente en entornos distribuidos y heterogéneos, y requieren de mecanismos para la gestión de dispositivos de diferentes capacidades (desde dispositivos móviles con recursos limitados hasta equipos con altas prestaciones). A medida que supervisan el entorno, su respuesta debe ser eficiente con el fin de reaccionar lo más rápidamente posible a un cambio, por lo que es necesario un sistema de gestión de los recursos adecuados, no sólo por ser esencial para hacer frente a las limitaciones de los sistemas empotrados sino por asegurar la eficacia. También es necesario evitar las interrupciones del servicio con el fin de evitar pérdidas de información. En consecuencia, se debe garantizar la disponibilidad en caso de fallo en los nodos de procesamiento o dispositivos sensores. Finalmente, otros aspectos críticos son la privacidad, la confidencialidad y la integridad de los datos (seguridad).

Por lo tanto, este tipo de sistemas plantean varios retos para los desarrolladores que tienen que ser tenidos en cuenta en las fases de análisis y diseño, y tienen que garantizarse en tiempo de ejecución [3]. Hay varios trabajos, que tienen que ver con cuestiones de seguridad y privacidad. Las soluciones más habituales son el cifrado de mensajes utilizando una infraestructura de clave pública (PKI) y Secure Socket Layer (SSL) [7], mecanismos de autorización y autenticación [4], y el desarrollo de marcos de seguridad [12] o de políticas de seguridad [1].

Por otro lado, también hay trabajos basados en middlewares de apoyo a las aplicaciones en su interacción o comunicación con otras aplicaciones. Estas soluciones se construyen comúnmente sobre plataformas que resuelven los retos de ubicuidad: *Open Services Gateway Initiative* (OSGi) [11], *Remote Procedure Call* (RPC) [5], *Object Request Broker* (ORB) [10], Reflex [6] o la *Foundation for Intelligent Physical Agents* (FIPA) [8].

Los sistemas adaptativos se definen comúnmente en la literatura como los que son capaces de modificarse automáticamente en respuesta a los cambios en su entorno [9], es decir, el sistema debería ser consciente de su estado y de su contexto. Sin embargo, la mayoría son soluciones ad-hoc que asumen aplicaciones sin estado de ejecución.

Este trabajo propone un middleware basado en agentes para la gestión de la ejecución de este tipo de aplicaciones, cuya idea preliminar fue presentada en [2]. En este trabajo se extiende el middleware con la gestión de eventos para adaptar las aplicaciones ante cambios en el contexto. Además, la disponibilidad de las aplicaciones queda asegurada mediante el aprovechamiento de la naturaleza móvil de los agentes, añadiéndose mecanismos para la gestión de su estado de ejecución. A diferencia de otros trabajos, se trata de un middleware genérico, no ligado a ningún campo de aplicación concreto, que ofrece plantillas de código con las que implementar los agentes.

El resto de este trabajo está estructurado de la siguiente manera: la sección 2 define los requisitos de las aplicaciones; la sección 3 identifica la solución propuesta que consiste en el middleware MAS-RECON que proporciona plantillas de agentes para la implementación de aplicaciones, así como los mecanismos para gestionar su ejecución. La sección 4 está dedicada a la evaluación de la solución propuesta mediante un demostrador y algunas pruebas experimentales. Por último, en la sección 5 se resumen las conclusiones más importantes y el trabajo futuro.

2 REQUISITOS DE LAS APLICACIONES

Las aplicaciones sensibles al contexto presentan tres objetivos principales: la monitorización, el reconocimiento temprano y la reacción rápida y adecuada. Para cumplir estos objetivos se supervisa la información de contexto por medio de sensores, teniendo en cuenta que cada medición se debe realizar a la frecuencia correcta: requisitos R1, R2, R3 en Tabla 1.

Tabla 1: Requisitos de las aplicaciones.

Identificador	Descripción del requisito
R1	Soporte a diferentes sensores y procesamiento personalizado
R2	Ejecución en plataformas distribuidas y heterogéneas
R3	Activación temporizada o esporádica
R4	Adaptabilidad a cambios en el contexto
R5	Disponibilidad con estado de ejecución

El procesamiento de los datos capturados permite detectar situaciones relevantes en las que es necesario actuar, siendo posible que la aplicación tenga que evolucionar en respuesta a dichos cambios (R4). Además, la monitorización continua implica asegurar la disponibilidad de las aplicaciones, incluso en caso de fallo de un nodo. Por último, la recuperación del servicio tiene que ser independiente de la aplicación, es decir, el diseño de la aplicación no ha de ser alterado (R5). A modo de resumen los principales requisitos exigidos por las aplicaciones objeto de este trabajo se recogen en la Tabla 1.

3 MIDDLEWARE MAS-RECON

Esta sección presenta la arquitectura general del sistema y del middleware MAS-RECON, un middleware basado en multiagentes y que proporciona los medios para implementar la funcionalidad de las aplicaciones (cumpliendo los requisitos R1, R2 y R3), gestionando la ejecución (temporizada o esporádica) y la comunicación entre los componentes de aplicación. Proporciona también los mecanismos de flexibilidad para posibilitar la adaptabilidad en tiempo de ejecución (requisito R4). Además, para asegurar la disponibilidad independiente de la aplicación en el caso de caída de nodos (requisito R5) se proponen mecanismos de negociación entre los nodos disponibles para reubicar el componente en fallo con su estado de ejecución.

Para ello, se propone una arquitectura de sistema que permite definir las aplicaciones como un conjunto de componentes que pueden ser ejecutados en dispositivos distribuidos y heterogéneos, y que tienen que interconectarse para lograr los objetivos de la aplicación.

3.1 ARQUITECTURA DEL SISTEMA

La arquitectura del sistema, representada en la Figura 2, se divide en tres ramas principales. En la primera se ubican los nodos físicos donde corren las aplicaciones, en segundo lugar los escenarios como agrupadores lógicos de aplicaciones y finalmente los eventos como elementos de interacción con las aplicaciones.

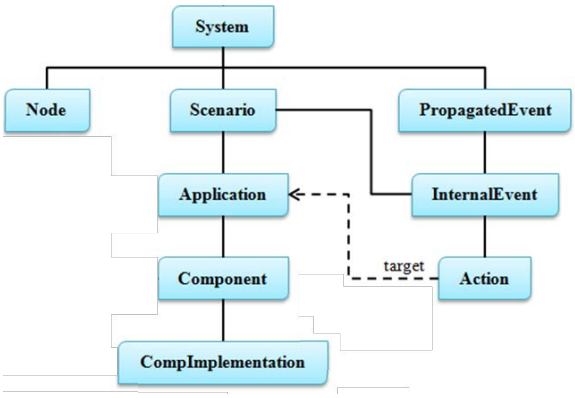

Figura 1: Arquitectura del sistema

El concepto de escenario es un agrupador de aplicaciones que tienen algo en común. Por ejemplo, en el caso de actividades de vigilancia, un escenario podría agrupar las aplicaciones relacionadas con un mismo edificio. Cada aplicación está compuesta por un conjunto de procesos (componentes) que pueden ser ejecutados en nodos distribuidos para conseguir el objetivo final de la aplicación. Los componentes colaboran intercambiando la información necesaria para proporcionar su servicio.

Finalmente, el concepto de evento permite actuar frente a cambios relevantes del contexto que requieren intervención. La ejecución de un evento interno (*internalEvent*) origina la activación de un conjunto de acciones sobre la aplicación que lanza el evento o sobre cualquier otra aplicación en el mismo escenario.

Es posible propagar un evento entre escenarios mediante el elemento *propagatedEvent*.

Finalmente, cabe señalar que la estructura del sistema tiene en cuenta que un mismo componente de aplicación puede tener diferentes implementaciones (*CompImplementation*). Además, se puede restringir la ejecución de una instancia de componente a uno o varios nodos concretos. Esto es útil en nodos que tienen elementos hardware como sensores o características necesarias para el componente.

3.2 ARQUITECTURA DEL MIDDLEWARE

La arquitectura del middleware propuesto, mostrada en la Figura 2, se basa en la plataforma JADE. Con el objetivo de cumplir los requisitos definidos en el apartado 2, se ha extendido la plataforma JADE con los módulos que se muestran en la parte superior de la Figura 2.

- El *Middleware Manager* (MM) como orquestador principal
- Un *Application Manager* (AM) por aplicación para gestionar sus componentes y los estados de ejecución.
- Un *Node Agent* (NA) en cada nodo proporciona información de ejecución útil para gestionar la disponibilidad.
- En *Event Manager* (EM) encargado de gestionar las acciones relacionadas con la adaptabilidad. Cada módulo de middleware se implementa en uno o varios agentes que se ejecutan sobre el sistema de multiagentes.

3.3. REQUISITOS FUNCIONALES Y TEMPORALES (R1, R2 y R3)

La plataforma JADE es una implementación completa del estándar *Foundation for Intelligent Phisical Agents* (FIPA), en el lenguaje Java. FIPA promueve la tecnología necesaria y establece estándares para la interacción y desarrollo de sistemas basados en agentes inteligentes. En este sentido, una infraestructura que cumpla FIPA debe soportar la gestión de agentes mediante los módulos que se muestran en la parte inferior de la Figura 2: el *Directory Facilitator* (DF), el *Agent Management System* (AMS), y el *Agent Communication Channel* (ACC). Siguiendo la especificación FIPA, debe haber al menos un agente DF en la plataforma encargado de proporcionar un servicio de *páginas amarillas* a los agentes que ofertan servicios y a los que los demandan; el AMS gestiona la creación, eliminación y migración de agentes; y el ACC soporta interoperabilidad entre plataformas. Finalmente, el denominado *Internal Platform Message Transport* (IPMT) proporciona servicios de enrutamiento entre agentes de una misma plataforma.

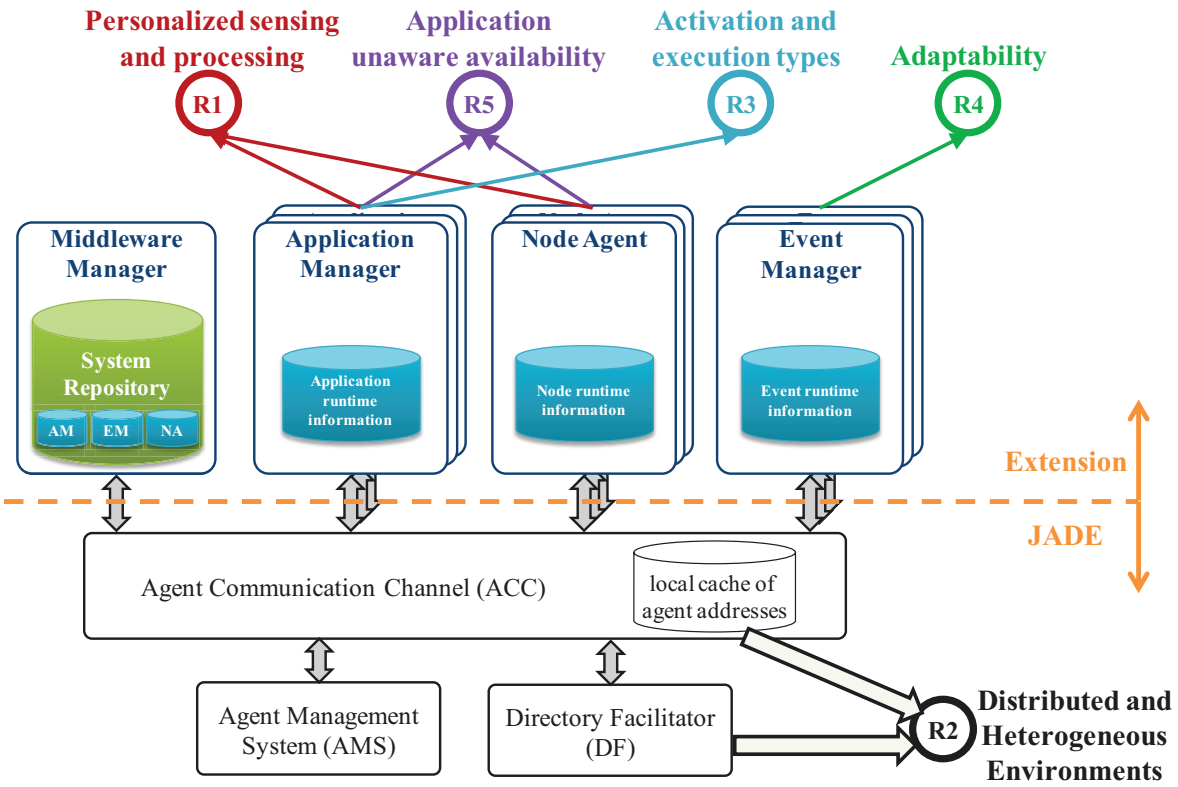

Figura 2: Arquitectura del middleware

El requisito R2 es intrínseco a la naturaleza distribuida de los agentes. Además, debido a que Java es multiplataforma y a que JADE permite su ejecución incluso en dispositivos con recursos limitados, el middleware propuesto soporta sistemas desde los muy limitados como sensores o dispositivos móviles hasta aquellos de altas prestaciones.

Adicionalmente, como los agentes distribuidos cooperan intercambiando mensajes se han definido tres ontologías FIPA para gestionar las comunicaciones: (1) ontología de datos para el intercambio de mensajes con información funcional, como valores de sensores o resultados del procesamiento de datos; (2) ontología de control para comandos que permitan a un usuario o a los propios agentes interactuar con los módulos del middleware y viceversa; (3) ontología de estado para gestionar el estado de ejecución de cada agente (valor de las variables relevantes del componente).

El módulo MM almacena y gestiona la información del sistema en el repositorio de sistema (*System Repository,* SR). La estructura jerárquica del SR, representada en la Figura 3, consta de dos ramas principales. Por una parte la información en tiempo de ejecución que incluye el estado de los eventos, nodos, aplicaciones y sus componentes. Y por otra parte, el diseño del sistema que se debe registrar siguiendo la estructura de la arquitectura de sistema propuesta.

Los nodos físicos son los dispositivos hardware, donde se ejecutan las instancias de componentes. Esto incluye el acceso a los sensores, actuadores y unidades de procesamiento. Cada nodo contiene una instancia del módulo NA que proporciona información física (número de procesadores, capacidad de almacenamiento y memoria, ancho de banda de red, tipo de CPU y plataforma) y la información de tiempo de ejecución (utilización de las CPU, almacenamiento y memoria libre) del nodo. El NA se registra automáticamente en el MM durante el arranque del nodo. También es el encargado de los procesos de negociación cuando lo requiere el AM.

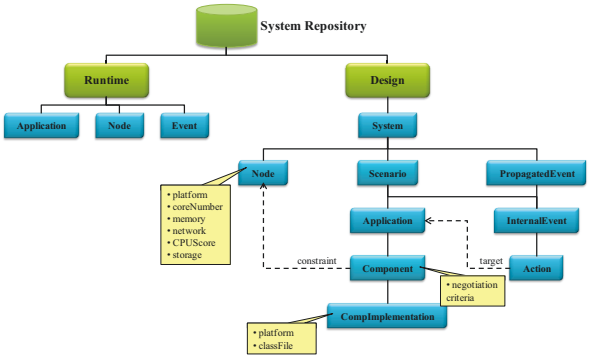

Figura 3: Estructura del *System Repository* en el *Middleware Manager*

Cabe resaltar que la estructura del código de los componentes se ha fijado para cumplir los requisitos R1 (detección y procesamiento personalizados) y R4 (disponibilidad independiente de la aplicación). Más concretamente, cada componente implementa la máquina de estados finitos (*Finite State Machine*, FSM) representada en la parte izquierda de la Figura 4 y que contiene los siguientes estados de FSM:

• *Inicio*: durante este estado de la FSM el agente espera a que se cumplan las condiciones necesarias para su inicio. Esto permite ejecutar las acciones de inicialización así como sincronizar el inicio de diferentes instancias de componente.

• *Ejecución*: el agente ejecuta su funcionalidad. Para permitir restaurarlo en caso de fallo del nodo, su estado de ejecución (valor de las variables relevantes

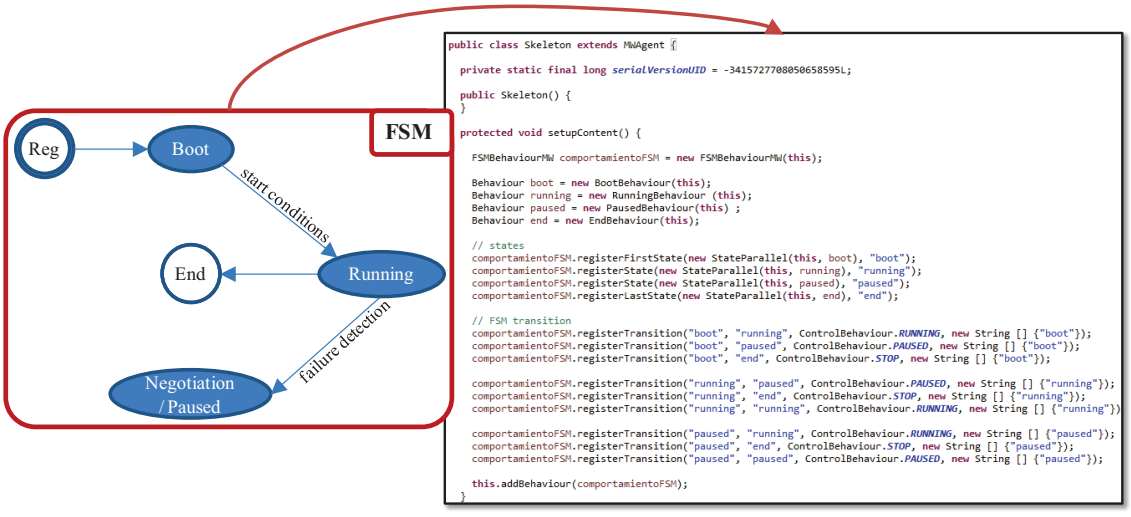

Figure 4. Finite State Machine (FSM) y su implementación en Java.

del componente) se refresca en el AM cada ciclo de ejecución.
• *Negociación/pausa:* Cuando se detecta un fallo, el AM configura el agente en este estado de la FSM. Los NA negociarán para restaurar la instancia del componente en su estado de ejecución.
• *Finalización:* en este estado de FSM finaliza la ejecución del componente, incluyendo las tareas de apagado.

La estructura del código Java que implementa esta FSM se muestra en la parte derecha de la Figura 4.
El desarrollador de software completa el código de cada estado de la FSM. En particular, si el componente requiere acciones de inicialización se incluirán todas las acciones necesarias en el estado de inicio de la FSM. Se procederá del mismo modo con las acciones de finalización en el estado correspondiente de la FSM. Por otra parte, el estado FSM de ejecución tiene que ser personalizado incluyendo la lógica de datos y la lógica de lanzamiento de eventos. Con el fin de facilitar esta tarea, se han definido dos plantillas de agente según el modo de activación (requisito R3):
(1) Temporizada: esta plantilla extiende la clase JADE TickerBehaviour. Implementa un agente con activación periódica que en cada ciclo ejecuta su funcionalidad, envía los resultados, y refresca el estado de ejecución en el AM.
(2) Esporádica: esta plantilla extiende la clase JADE CyclicBehaviour. Se utiliza en los componentes que se activan con la recepción de mensajes. Por lo tanto, tras recibir los datos de entrada ejecutan la funcionalidad, envían los resultados, y refrescan el estado de ejecución en el AM.

Como se ha comentado anteriormente, cada instancia de componente en el sistema es un agente en ejecución. El MM despliega tantos AM como aplicaciones en ejecución. Es función de cada AM supervisar la ejecución de los componentes que la forman. Esto incluye las siguientes tareas:
• Inicio de componentes. Esta tarea consiste en seleccionar el nodo apropiado donde ejecutar la instancia de componente. La decisión se realiza mediante un proceso de negociación entre los NA.
• Gestión del estado de ejecución de los componentes con estado.
• Gestión del ciclo de vida de los componentes. El AM puede forzar cambios en el estado de la FSM del componente. Esto ocurre por ejemplo cuando se ha detectado un fallo.
• Recuperación ante fallos. El AM es el encargado de coordinar el proceso de recuperación de un componente en fallo.

3.4. ADAPTABILIDAD (R4)

Para hacer frente a las necesidades de adaptabilidad (requisito R4), los eventos registrados en el repositorio del sistema son supervisados por instancias del EM. El MM despliega tantas instancias del EM como eventos activos haya. Cada EM comprueba que se lleven a cabo las acciones establecidas para el evento en orden y tiempo. La interacción entre una instancia de componente y el EM se realiza mediante una sentencia en la implementación del agente. A modo de ejemplo, la Figura 5 representa el diagrama de secuencia de la ejecución de las acciones asociada a un evento. En este caso, la instancia de componente *checkRelaxed001* activa un evento (gestionado por *EM_relaxed*) que tiene como acciones el inicio (*action1_create*) de una nueva aplicación (*AM_BloodPressure* inicia el componente *bpAcquisition002*) y la parada (*action2_destroy*) de una aplicación existente (*AM_checkRelaxed* finaliza el componente *checkRelaxed005*).

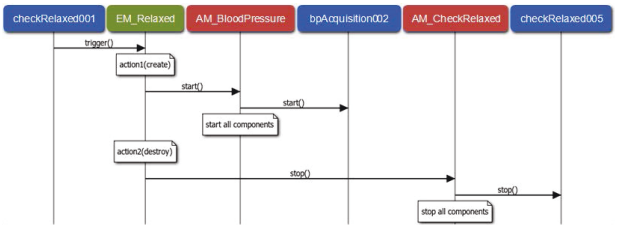

Figura 5: Ejecución de evento

La relación entre el componente y el EM correspondiente al evento a ejecutar es por código. Es decir, el identificador del EM a invocar es un dato que el desarrollador del componente conoce y lo escribe en una sentencia en el código fuente del componente. Como los eventos y sus acciones se relacionan en el repositorio, el hecho de añadir nuevos escenarios o aplicaciones no implica cambiar el código fuente del componente ya que basta con cambiar a nivel de registro las acciones que lo componen. De esta manera, cuando se invoca un EM concreto se ejecutan las acciones asociadas en el repositorio del sistema.

3.5. DISPONIBILIDAD CON ESTADO DE EJECUCIÓN (R5)

Los AM y NA son los principales responsables del soporte de la disponibilidad (requisito R5). Como se ha comentado anteriormente, el mecanismo de disponibilidad propuesto se basa en encontrar el nodo más adecuado para ubicar la instancia de un componente que ha fallado. Por lo tanto, por un lado es necesario detectar la caída de un componente, y por otro lado es necesario ejecutar las tareas necesarias para su recuperación. Como ejemplo, la secuencia ilustrada en la Figura 6 detalla el proceso de recuperación de un componente con su estado.

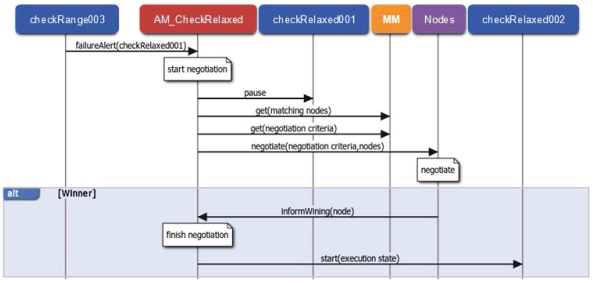

Figura 6: Detección de fallo y recuperación con estado de ejecución

En particular, el componente *checkRelaxed* es el que falla. El error es detectado por el componente *checkRange003* que a su vez informa al AM (*AM_CheckRelaxed*). El AM cambia el estado FSM del componente en fallo a *pausa* (si el componente no ha fallado completamente esta acción trata de evitar que continúe activo). A continuación, el AM inicia las labores de reubicación del componente: (1) consulta al MM la lista de nodos candidatos a ubicarlo; (2) lanza la negociación entre nodos; (3) recibe respuesta del nodo ganador y finalmente restaura el componente en su último estado de ejecución.

4 EVALUACIÓN DE LA SOLUCIÓN PROPUESTA

El rendimiento se ha evaluado teniendo en cuenta los objetivos principales del trabajo: adaptabilidad y disponibilidad. En particular, la adaptabilidad se evalúa en términos del tiempo de reacción para adaptarse a un cambio del contexto, mientras que la disponibilidad se evalúa de acuerdo con el tiempo de recuperación de un fallo.

Ambos parámetros se ponen a prueba mediante el uso de experimentos similares. El punto de partida es una aplicación muy simple y secuencial que captura un valor de un sensor, lo procesa y muestra el resultado. En ambos casos se incrementa el número de nodos disponibles para mantener instancias de componentes. Para las pruebas de disponibilidad se incrementa el número de tareas de procesamiento, es decir, el número de componentes de la aplicación.

Sin embargo, para la adaptabilidad, se incrementa el número de acciones activadas por el evento. Con el fin de evitar que las diferentes capacidades de procesamiento de nodos interfieran el análisis de los resultados, y teniendo en cuenta que en un escenario real hay muchos dispositivos con recursos limitados, todos los nodos en el experimento son Raspberry Pi.

En cuanto a la disponibilidad, la Figura 7 muestra el tiempo de recuperación de las diferentes pruebas. Se mide el tiempo desde la caída de un nodo hasta la recuperación de todos los componentes afectados. Como era de esperar, el tiempo de recuperación aumenta con el número de nodos y componentes. De hecho, el tiempo de recuperación aumenta casi proporcionalmente al número de nodos, ya que aumenta el número de nodos que participan en la negociación y debido a la baja capacidad de procesamiento de la Raspberry Pi. Este fenómeno condiciona el proceso de negociación. Del mismo modo, el tiempo de recuperación aumenta con el número de componentes de aplicación.

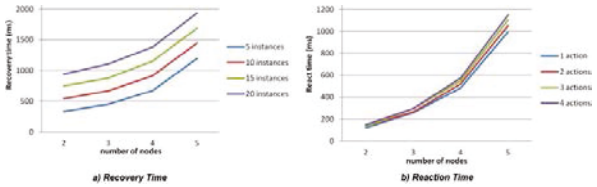

Figura 7: Tiempo de recuperación (a) y tiempo de reacción (b)

En lo que se refiere a la capacidad de adaptación, la Figura 7 representa el tiempo de reacción en milisegundos desde que el evento se activa hasta que finalizan las acciones asociadas. Para simplificar, todas las acciones activadas por el evento son de creación de una nueva aplicación. Por lo tanto, el tiempo resultante incluye la puesta en marcha de las aplicaciones. Una vez más, como se esperaba, el número de nodos y el número de acciones incrementan el tiempo de reacción.

5 CONCLUSIONES

Este trabajo presenta el middleware MAS-RECON, una solución para implementar la gestión de la adaptabilidad y la disponibilidad en aplicaciones distribuidas. Se ha adoptado tecnología de multiagentes para convertir componentes en entidades inteligentes. En este contexto, el middleware propuesto extiende la plataforma JADE proporcionando mecanismos que permiten facilitar la adaptabilidad y que aseguran la disponibilidad incluso para aplicaciones con estado. Concretamente, el módulo *Event Manager* gestiona la ejecución de las acciones asociadas a un evento. Como resultado, se logra optimizar el uso de los recursos. La disponibilidad se asegura recuperando el componente que ha fallado con su estado de ejecución en el nodo apropiado. Se facilitan mecanismos de detección de fallos, recuperación del estado de ejecución y negociación entre nodos orquestada por el módulo *Application Manager*.

Se comprueba experimentalmente que el tiempo de recuperación (disponibilidad) y el tiempo de reacción (adaptabilidad) se ven afectados cuando el número de nodos que pueden contener instancias de componentes y número de acciones asociadas al evento aumentan

respectivamente. Sin embargo, no hay soporte de disponibilidad para los módulos del propio middleware. Por ejemplo, si el nodo que ejecuta un AM falla, los datos y el estado de ejecución de los componentes de su aplicación se pierden. Y por otra parte, el middleware carece de mecanismos de control de admisión que ordenen o incluso denieguen la entrada de nuevas aplicaciones cuando los recursos no sean suficientes. Por lo tanto, el trabajo futuro está dirigido a la exploración de la distribución del repositorio del sistema para mejorar la disponibilidad de los módulos de middleware, y al desarrollo del control de admisión.

Adicionalmente, como se ha demostrado en la sección de evaluación, la limitación de recursos reduce el rendimiento del middleware debido a una mayor lentitud de las acciones de negociación. Por lo tanto, el trabajo futuro también se centra en el soporte de QoS flexibles para aplicaciones no críticas.

Agradecimientos

Este trabajo se ha subvencionado en parte por el Gobierno de España (MCYT) bajo el proyecto DPI-2015-68602-R y por la Universidad del País Vasco (UPV/EHU) con subvención UFI11/28.

Referencias

[1] Agirre, A.; Parra, J.; Armentia, A.; Ghoneim, A.; Estévez, E.; Marcos, M., (2015) QoS management for dependable sensory environments. Multimed. Tools Appl., doi:10.1007/s11042–015–2781–4.

[2] Armentia, A.; Gangoiti, U.; Priego, R.; Marcos, M., (2015) A Multi-Agent Based Approach to Support Adaptability in Home Care Applications. In Proceedings of the 2nd Conference on Embedded Systems, Computational Intelligence and Telematics in Control, Maribor, Slovenia, pp. 1-6

[3] Becker, M., (2008) Software Architecture Trends and Promising Technology for Ambient Assisted Living Systems. In Proceedings of Dagstuhl Seminar, Dagstuhl, Germany; p.p. 1-18.

[4] Bajo, J.; Fraile, J.A.; Pérez-Lancho, B.; Corchado, J.M., (2010) The THOMAS architecture in Home Care scenarios: A case study. Expert Syst. Appl., 37, 3986–3999.

[5] Bloomer, J. Power, (1992) Programming with RPC; O'Reilly Media: Sebastopol, CA, USA.

[6] Capra, L.; Emmerich, W.; Mascolo, C., (2003) CARISMA: Context-Aware Reflective middleware System for Mobile Applications. IEEE Trans. Softw. Eng., 29, 929–945.

[7] Corchado, J.M.; Bajo, J.; Abraham, A. GerAmi, (2008) Improving Healthcare Delivery in Geriatric Residences. IEEE Intell. Syst., 23, 19–25.

[8] Foundation for Intelligent Physical Agents. Standard FIPA Specifications. Available online: http://www.fipa.org/repository/standardspecs.html (accessed on 7 September 2015).

[9] Krupitzer, C.; Roth, F.M.; VanSyckel, S.; Schiele, G.; Becker, C., (2014) A survey on engineering approaches for self-adaptive systems. Pervasive Mob. Comput. 17, 184–206.

[10] Object Management Group. Object Request Broker Available online: www.omg.org/ gettingstarted/ orb_basics.htm (accessed oct-2015).

[11] OSGiTM Alliance. The OSGi Architecture. Available online: http://www.osgi.org/Technology/ WhatIsOSGi (Accessed oct-2015)

[12] Vitabile, S.; Conti, V.; Militello, C.; Sorbello, F., (2009) An extended JADE-S based framework for developing secure Multi-Agent Systems. Comput. Stand. Interfaces, 31, 913–930.