UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR

DE INGENIEROS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

USO DE REDES BAYESIANAS PARA DIAGNOSTICAR FALLOS INCIERTOS DE RED

ÁLVARO CARRERA BARROSO

2010

PROYECTO FIN DE CARRERA

Título: Uso de redes bayesianas para diagnosticar fallos inciertos de red

Autor: Álvaro Carrera Barroso

Tutor: Javier González Ordás

Ponente: Mercedes Garijo Ayestarán

Departamento: Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente: D. Gregorio Fernández Fernández

Vocal: Dª. Mercedes Garijo Ayestarán

Secretario: D. Carlos Ángel Iglesias Fernández

Suplente: D. José Carlos González Cristobal

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR

DE INGENIEROS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

USO DE REDES BAYESIANAS PARA DIAGNOSTICAR FALLOS INCIERTOS DE RED

ÁLVARO CARRERA BARROSO

2010

RESUMEN:

El objetivo de este proyecto fin de carrera es el desarrollo de un sistema de diagnóstico distribuido usando redes bayesianas para alcanzar una lista de las causas más probables de fallo presentadas en el escenario del proyecto MAGNETO. Con ese objetivo, se ha desarrollado un sistema con arquitectura multi-agente que diagnóstica las causas de fallo dado un síntoma usando inferencia sobre redes bayesianas durante dicho proceso. Dentro del sistema existen diferentes tipos de agentes con sus funciones específicas que proporcionan un método distribuido de diagnóstico que persigue la escalabilidad y la modularidad.

El proceso de desarrollo sigue en gran parte la metodología descrita por el Proceso Unificado de Desarrollo de Software en cuanto a la aplicación de las diferentes fases que éste aplica. Previamente a la puesta en marcha de estas fases, se ha realizado un estudio sobre los diferentes aspecto de la gestión de red que afectan al sistema y sobre la herramienta matemática usada para el proceso de diagnóstico, las redes bayesianas. El objetivo de esta fase de estudio ha sido seleccionar el enfoque más adecuado para desarrollar el proyecto de una manera coherente a la vez que innovadora.

En la memoria se muestran los resultados de las diferentes fases del proyecto además de una evaluación de las diferentes herramientas utilizadas en él, varios anexos que tienen el fin de ampliar algunos aspectos que facilitan la comprensión de éstos y algunos manuales para proveer la información necesaria para el mantenimiento y el desarrollo de mejoras del sistema a futuros desarrolladores.

PALABRAS CLAVE: agente, diagnóstico, distribuido, gestión, incertidumbre, inferencia, JADE, MAGNETO, OSGi, red bayesiana, WADE.

ABSTRACT:

The aim of this project is the development of a distributed diagnosis system using Bayesian networks to achieve a list of the most probable fail causes in the MAGNETO project scenario. With this aim, it has been developed a system with a multi-agent architecture that diagnoses the fail causes given a symptom using Bayesian inference during this process. In the system, there are different types of agents with specific functionalities that provide a distributed way to diagnose. By this way, the scalability and the modularity are pursued.

The development process follows largely the methodology described in the Unified Development of Software Process applying the different phases suggested in it. Before the execution of these phases, a study has been realized. The study dealt with the different aspects of the network management that affect to the system and with the mathematical tool used by the diagnosis process, the Bayesian networks. The aim of this study phase has been to select the more effective approach to develop the project with consistency and innovation.

In this report, the results of the different project phases are shown. Furthermore, it includes an evaluation of the different used tools, several appendixes with the aim of increase some aspects that facilitate the understanding of these, and some tutorials to provide the needed information to future developers to maintain and develop system improvements.

KEYWORDS: agent, bayesian network, diagnosis, distributed, inference, JADE, MAGNETO, management, OSGi, uncertainty, WADE.

AGRADECIMIENTOS:

Me gustaría agradecer a todas aquellas personas que me han dado ánimos para realizar este proyecto y que han estado día a día motivándome para poder concluirlo.

Y más concretamente, me gustaría mencionar a mi madre y a mi padre por haberme brindado la oportunidad de llevar a cabo estos estudios, a mi novia por haber estado siempre ahí en los momentos de necesidad, a mis compañeros de universidad Gema, Carlos, Gonzalo, Andrés, Alberto, Jaime, Rafa, Víctor, Jorge, Sara, Juancho, Ariel, Rocío, Álvaro, Beatriz, Jonathan, Daniel, Julio, Iris, Narciso, Natalia, Marta y tantos otros, por haberme acompañado en el camino hasta aquí, a Jose, Raquel, Javier, Pablo, Javi, Andrés , Sergio, Gregory, Florian y Sebastian por trabajar junto a mí durante el desarrollo del proyecto y a Mercedes por servirme de guía en la realización del mismo.

Por supuesto, también al resto de familiares, amigos, compañeros y profesores de la universidad que no hayan sido ya mencionados. Gracias una vez más.

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

GLOSARIO

TÉRMINO	CONCEPTO
ACS	Automatic Configuration Server Servidor usado por diferentes protocolos de gestión remota como el TR-069 o el UPnP.
ADF	Agent Definition File Fichero de la plataforma JADEX que presenta tanto el estado inicial de un agente como sus funcionalidades.
AIC	Akaike Information Criterion Criterio de decisión dentro de los procesos de aprendizaje estructural de redes bayesianas.
AMS	Agent Management System Agente de la plataforma JADE que supervisa la plataforma de manera global.
AOP	Agent-Oriented Programming Paradigma software relativamente nuevo que lleva conceptos de teorías de inteligencia artificial a la corriente actual de sistemas distribuidos
API	Application Programming Interface Conjunto de funciones y procedimientos (o métodos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
BDI	Belief-Desire-Intention Modelo de diseño de agentes que expone claramente las capacidades y las metas del agente.
Bundle	Componentes software utilizados dentro de la plataforma de servicios OSGi.
CA	Controller Agents Agente de la plataforma WADE encargado de la recuperación de agentes dentro de un contenedor.
CEP	Complex Event Processing Concepto sobre el procesamiento de múltiples eventos que pueden derivar en el descubrimiento de otros eventos más interesantes en el sistema.
CFA	Configuration Agent Agente de la plataforma WADE encargado del arranque de los agentes en máquinas remotas.
CPT	Conditional Probability Table Tabla de probabilidad condicional de un nodo dentro de una red bayesiana.
DAG	Directed Acyclic Graph Modelo gráfico acíclico que permite modelar la estructura de una red bayesiana.
DAL	Device Adaptation Layer Capa de abstracción que envía eventos al sistema de diagnóstico para notificar fallos detectados. Es un actor del sistema.

DF	Directory Facilitator Agente de la plataforma JADE que ofrece un servicio de páginas amarillas para que el resto de agentes puedan descubrirse correctamente.
FIPA	Foundation for Intelligent Physical Agents Estándar que especifica un modelo de comunicación entre agentes.
F-Logic	Fuzzy Logic Tipo de lógica en la que los valores no tienen valores discretos, si no un rango de valores continuo.
GPL	General Public License
ISP	Internet Service Provider
J2ME	Java 2 Micro Edition Edición del entorno de ejecución de la máquina virtual JAVA con recursos mínimos.
J2SE	Java 2 Standard Edition Edición estándar del entorno de ejecución de la máquina virtual JAVA.
JADE	Java Agent Development Environment Plataforma de agentes desarrolla completamente en JAVA.
JADEX	Java and XML Agent Development Platform Plataforma de agentes basada en JADE que da soporte para el desarrollo de agentes basados en el modelo de diseño BDI.
JRS	Jade Runtime Service Servicio OSGi que ofrece JADE para la creación de agentes dentro de la plataforma de servicios.
LEAP	Lightweight and Extensible Agent Platform Versión de la plataforma de agentes JADE que permite crear agentes ligeros para la ejecución en entornos con recursos limitados.
LGPL	Lesser General Public License
MDL	Minimum Description Length Criterio de decisión dentro de los procesos de aprendizaje estructural de redes bayesianas.
MTP	Message Transport Protocol Protocolo de transporte de mensajes usado por la plataforma de agentes JADE que permite comunicarse con otras plataformas que cumplan con la especificación FIPA.
NAT	Network Address Translation Proceso de traducción de IP's que permite la creación de direcciones privadas ampliando así el uso de una misma dirección pública.
OSGi	Open Services Gateway initiative Tecnología con un conjunto de especificaciones que define un sistema de componentes dinámicos para JAVA.
OSS	Operational Support System Sistemas de ordenadores usados por las empresas de telecomunicaciones para ofrecer servicios.
RTP	Real-time Transport Protocol Protocolo que define un formato de paquetes estándar para el envío de audio y video a través de la red.
RTSP	Real-Time Streaming Protocol Protocolo usado para el establecimiento de sesiones entre puntos finales.
TR-069	Technical Report 069 Protocolo bidireccional de gestión de dispositivos remotos.

UML	Unified Modelling Language Lenguaje de modela estándar de propósito general usado en el campo de la ingeniería del software.
UPnP	Universal Plug and Play Conjunto de protocolos de red promulgados por el UPnP Forum.
WADE	Workflows and Agents Development Environment Aplicación basada en JADE que añade diferentes características de gestión y ejecución a la plataforma de agentes.
XDSL	XML Domain-Specific Language Formato variante del DSL (Domain-Specific Language) que permite modelar la información en formato XML.
XML	Extensible Markup Language Meta-lenguaje extensible creado por el World Wide Web Consortium (W3C).

1. INTRODUCCIÓN

1.1 MOTIVACIÓN

Este proyecto fin de carrera se enmarca dentro del proyecto MAGNETO, perteneciente al programa AVANZA convocatoria 2008-2009. Dicho proyecto lleva el sello de la oficina CELTIC (http://www.celtic-initiative.org) y es llevado a cabo entre los siguientes socios: *Ericsson, Data Ductus, GMV-SGI, Siemens AG Austria, Telefónica I+D, University of Evry Val d'Essonne, Vestel, Waterford Institute of Technology, University of Ottawa y Blusens Technology*.

El proyecto MAGNETO está enfocado hacia las redes de comunicaciones que están en constante evolución, dirigidas por las innovaciones tecnológicas que aumentan las capacidades de la red.

Más concretamente, éste se centra en la gestión de los bordes exteriores de la red (o "*Outer Edge*"), en las redes de área del hogar (o HAN); ya que, con el creciente despliegue de dispositivos dentro de estas redes de acceso restringido, la gestión de éstas es cada vez más compleja.

Su principal objetivo es desarrollar un marco de gestión distribuida innovador y un enfoque de gestión de servicio que permita ofrecer diferentes capacidades en los dispositivos de estas nuevas redes emergentes. Este marco de gestión debe tener en cuenta que se deben organizar los sistemas de gestión para ser capaz de escalar en el aspecto anterior en el que tantas redes están en expansión; para ello se propone un enfoque autonómico para la auto-gestión del sistema.

Para conseguir esto, el sistema debe configurar dispositivos de manera automática, manejar de manera autónoma las tareas de garantía de servicio relacionadas con la operación de estos una vez desplegados, detectar causas de fallo y recuperarse de estos automáticamente entre otros muchos aspectos.

Dentro del proyecto MAGNETO, se utiliza como servicio principal un servicio de video bajo demanda en el que los usuarios finales comparten contenido multimedia en unas condiciones previamente configuradas.

Este proyecto fin de carrera se centra en la capacidad de detectar causas de fallo del servicio ofrecido, para ello se ha desarrollado de un módulo de diagnóstico de fallos automático que permitirá al sistema conocer la causa de un síntoma de fallo.

Interactuando con los diferentes módulos que componen el proyecto MAGNETO en su conjunto, el módulo de diagnóstico de fallos recibe una notificación de síntoma de fallo e inicia un proceso de deducción para alcanzar la causa más probable. Una vez alcanzada una conclusión considerada como válida, el módulo de diagnóstico notifica a un módulo que transmite la conclusión alcanzada al módulo de recuperación de fallos.

1.2 OBJETIVO DEL PROYECTO

El objetivo de este proyecto fin de carrera es el desarrollo de un sistema de diagnóstico distribuido usando redes bayesianas para alcanzar una lista de las causas más probables de fallo presentadas en el escenario del proyecto MAGNETO.

Este objetivo principal presenta dos puntos clave para el proyecto. Uno de ellos es la característica de diagnóstico distribuido y el otro es el uso de redes bayesianas.

El primero de estos puntos es debido a que en los servicios actuales la causa de un síntoma de fallo puede encontrarse en cualquier punto de la red. Por lo tanto, si el diagnóstico se realizase de forma centralizada, habría zonas con acceso restringido en las que no se podría acceder de manera sencilla y además se sobrecargaría un punto concreto de la red. Sin embargo al realizarse de manera distribuida, partes diferentes del diagnóstico se pueden realizar en los puntos en los que se crea más conveniente de la red reduciendo así la carga de un único punto centralizado que debería realizar todas las tareas.

Con respecto al segundo punto clave del objetivo principal del proyecto, el uso de redes bayesianas para alcanzar conclusiones válidas en el diagnóstico, se plantean varias opciones. Ya que para alcanzar una conclusión se pueden utilizar diferentes enfoques, por ejemplo, usando motores de inferencia de reglas que vayan marcando paso a paso los diferentes pasos del diagnóstico, o incluyendo lógica difusa a estas reglas, añadiendo así probabilidades al razonamiento seguido en el diagnóstico. No obstante, una red bayesiana tiene unas capacidades de inferencia mayores al permitir la existencia de relaciones implícitas en ciertas estructuras y un diseño y manejo más simple. Para exponer razonadamente esta afirmación, se expone a continuación una pequeña definición sobre el concepto de red bayesiana.

Una red bayesiana se puede definir brevemente como un modelo gráfico acíclico directo (*Directed Acyclic Graph*, DAG) en el que se definen dependencias entre un conjunto de distribuciones de probabilidad de las variables representadas en los nodos del DAG. Cada variable o nodo consta de una serie de estados (normalmente variables discretas) y dichos nodos se relacionan entre sí a través de arcos directos que representan la dependencia de uno hacia otro, es decir, la dependencia entre variables.

Si una variable depende a la vez de dos o más variables aparentemente independientes, se genera una dependencia implícita en esta estructura entre esas dos o más variables. Este hecho es conocido habitualmente como estructura en V y aumenta mucho la aplicación de redes bayesianas para el diagnóstico ya que puede modificar la creencia de una variable hipótesis a través de otra hipótesis aparentemente independiente, lo cual no se da en un sistema de reglas con lógica difusa.

Además de esta característica a favor de las redes bayesianas, existe otra con respecto a su uso y diseño. Esta herramienta matemática permite modelar múltiples relaciones mediante las tablas de probabilidad interna perteneciente a un nodo de la red. Para exponer esta característica, se presenta el siguiente ejemplo: un nodo tiene cinco posibles estados y este nodo depende de tres nodos más con cuatro estados cada uno. Para representar la misma información que tiene la tabla de probabilidad condicional del nodo bajo estudio explícitamente en un sistema de reglas con lógica difusa se necesitaría un conjunto de 320 ($5x4^3$) reglas. Y además se perderían las relaciones implícitas en la estructura en V generada por la dependencia múltiple de la variable bajo estudio.

Como último punto importante de esta herramienta, los algoritmos de inferencia desarrollado para el uso de redes bayesianas hace de estas una manera muy eficiente de alcanzar conclusiones manejando incertidumbre como factor esencial en la operación.

Por los argumentos mostrados anteriormente y como resumen, los dos objetivos principales del proyecto son:

- El desarrollo de un sistema distribuido para realizar diagnósticos sin sobrecargar puntos críticos de la red y así tener mayor alcance en la obtención de las causas de fallo.
- La exposición de cómo el uso de redes bayesianas permite alcanzar conclusiones manejando durante todo momento la incertidumbre inherente a un proceso de diagnóstico.

1.3 RESUMEN DE LA SOLUCIÓN PROPUESTA

Para cumplir el objetivo del proyecto, se ha seguido el enfoque de "Programación orientada a agentes". Así, un conjunto de agentes colaboran entre sí para alcanzar un diagnóstico válido utilizando redes bayesianas para deducir la causa de fallo más probable dentro de las posibles.

Para alcanzar este objetivo, dicho conjunto de agentes usan una ontología que ha sido diseñada para contener los conceptos más útiles en el diagnóstico con redes bayesianas. En dicha ontología también se definen las acciones que los diferentes agentes pueden solicitarse unos a otros para alcanzar unas creencias finales.

Dentro del sistema de diagnóstico de fallos, existen diferentes tipos de agentes:

Tipos de agentes	Descripción
Agente de interfaz	Agente que es la interfaz externa más importante del sistema interactuando con los sistemas externos recibiendo síntomas o enviando informes.
Agente de diagnóstico	Agente director del diagnóstico solicitando datos a diferentes agentes en el sistema.
Agente especialista	Agente que realiza pruebas o parte de la inferencia interactuando con otros agentes.

TABLA 1: TABLA DE RESUMEN DE AGENTES

El funcionamiento básico del sistema de agentes comienza cuando un agente de interfaz recibe un evento de detección de síntoma de fallo. En ese instante, ese agente procesa la información debidamente y solicita a un agente de diagnóstico que comience un proceso para alcanzar las causas más probables de fallo. El agente de diagnóstico consultará las posibles acciones que puede tomar para alcanzar el mejor resultado posible y comenzará a consultar a los diferentes agentes especialistas. Este tipo de agente y algunos de los agentes especialistas realizan inferencia bayesiana durante un proceso de diagnóstico en el cual comparten creencias para conseguir un diagnóstico de manera conjunta. Una vez que el diagnóstico ha finalizado, el agente de

diagnóstico informa al agente que lo solicitó y éste envía un informe al sistema externo con capacidad para procesarlo.

Por último y ya que el sistema de diagnóstico se despliega de una red final de usuario, el software final se despliega dentro de una plataforma de servicios OSGi, que cumple una serie de requisitos que permite gestionar con facilidad el ciclo de vida del software (despliegue, ejecución, actualización, etc.)

1.4 ESTRUCTURA DE LA MEMORIA DEL PROYECTO

Este documento presenta el trabajo realizado en diferentes secciones. Esta memoria se estructura en diez capítulos, que a su vez están divididos en diferentes secciones, y cinco anexos, que presentan información de interés para la comprensión de algunos aspectos del proyecto o las plataformas utilizadas en él.

En el presente capítulo, capítulo 1: **Introducción**, se muestra una visión global del proyecto y de la memoria. También se presentan unas ideas generales sobre la motivación del proyecto, un resumen sobre la solución presentada y un breve resumen sobre la estructura de la memoria del proyecto.

En el capítulo 2: **Técnicas de gestión de red**, se presentan algunas de las características más relevantes en los sistemas de gestión de red actualmente al igual que algunos de los proyectos o enfoques más importantes dentro de esas características.

En el capítulo 3: **Redes bayesianas**, se presenta un estudio sobre redes bayesianas presentando tanto una definición formal, como los métodos de inferencia usados y los diferentes métodos de construcción y aprendizaje de redes bayesianas.

En el capítulo 4: **Herramientas utilizadas**, se muestra una comparativa sobre las opciones que se presentan a la hora de usar las diferentes herramientas utilizadas tanto en el desarrollo como en el despliegue del sistema. Cada herramienta presenta una tabla comparativa con el resto de opciones y una conclusión final en la que se expone el motivo de usar una u otra.

En el capítulo 5: **Análisis del problema**, se realiza un estudio sobre el problema a resolver mostrándose el escenario. Tras esto se desglosan los diferentes casos de uso y los requisitos que el sistema debe cumplir.

En el capítulo 6: **Diseño arquitectónico**, se presenta la solución propuesta para el problema y el diseño del sistema, tanto la ontología como los diferentes agentes con sus respectivos diagramas UML2.0 que representan el comportamiento del sistema.

En el capítulo 7: **Plan de despliegue**, se muestran los requisitos hardware y software que se necesitan para el correcto despliegue del sistema. Además se añade una explicación detallada sobre la configuración del sistema dentro del entorno de producción.

En el capítulo 8: **Plan de pruebas**, se presenta la maqueta de pruebas donde se ha desplegado el sistema para realizar las pruebas apropiadas y a su vez un listado de las pruebas con el resultado obtenido.

En el capítulo 9: **Manual del desarrollador**, se realiza una presentación más detallada sobre la estructura del código del proyecto para facilitar la adaptación al proyecto a nuevos desarrolladores que continúen con el trabajo realizado.

En el capítulo 10: **Conclusiones y trabajo futuro**, se presentan las conclusiones obtenidas tras la realización del proyecto y las diferentes líneas de trabajo o investigación que pueden seguirse para ampliar las características del proyecto.

2. TÉCNICAS DE GESTIÓN DE RED

2.1 INTRODUCCIÓN

Desde el inicio de las redes de comunicación, la gestión de red ha estado orientada a un modo centralizado y jerárquico, como en las recomendaciones *Telecommunications Management Networks*(ITU-T Princ 2000),(ITU-T Recom 2000) de ITU-T. Estos modelos han resultado de gran utilidad en el pasado, pero no están diseñados expresamente los entornos de red más heterogéneos y dinámicos que están emergiendo actualmente.

Otro reto importante en los sistemas de gestión de red está relacionado con la alta escalabilidad que debe presentar para cubrir el gran compromiso que presenta gestionar millones de equipos que puede incluir equipo personalizado. Estos problemas de escalabilidad son solucionados normalmente con arquitecturas jerárquicas que tienen el inconveniente de ser algo inflexibles.

Como conclusión, las nuevas generaciones de sistemas de gestión deben ser capaces de manejar información incompleta o incierta.

En las siguientes secciones, se muestran algunas de las técnicas más relevantes en el proyecto con respecto a la gestión de la red.

2.2 GESTIÓN DISTRIBUIDA

Todas las aplicaciones de gestión de red, por su propia naturaleza, operan sobre un conjunto de elementos física o lógicamente distribuidos, con partes de una cierta aplicación corriendo en varios elementos de dicha red. Una solución centralizada al problema de gestión normalmente conlleva una alta carga de mensajería en un mismo punto de la red introduciendo un gran gasto de ancho de banda; lo que provoca un uso ineficiente de los recursos, un punto fallo crítico y un cuello de botella en la red.

Desde otro punto de vista, sin embargo, existe un número de arquitecturas de gestión actualmente usadas en redes Ad-hoc, basadas en una combinación de aproximaciones distribuidas y jerárquicas. El proyecto MADEIRA(Frints, Ortega Abad y Arozarena Llopis 2006) introdujo una arquitectura distribuida para gestión de redes de dispositivos dinámicos, con la ventaja de usar una jerarquía completamente dinámica.

Esta arquitectura consiste en la división de la red en diferentes *clusters* (o subgrupos) de estaciones de comunicaciones las cuales son gestionadas por un *cluster head* (o líder de grupo). Éste es asignado dinámicamente en el caso de que el líder anterior desaparezca o falle. Con este principio y jerarquizando la distribución de los *clusters*, se consigue una arquitectura flexible que le permite adaptarse a fallos y cambios en la red, lo cual es básico para obtener una gestión eficaz de las nuevas redes dinámicas a gran escala que deberán ofrecer comportamientos adaptativos y descentralizados de control.

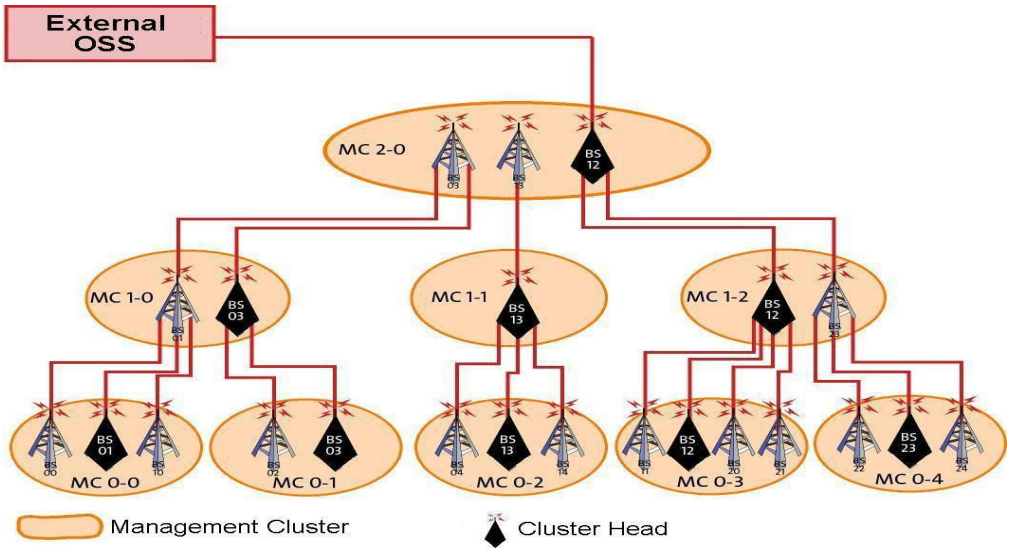

ILUSTRACIÓN 1: AGRUPACIONES JERÁRQUICAS EN MADEIRA

Con respecto a la garantía de la calidad de servicio, enormes cantidades de eventos deben ser procesados por el sistema cada minuto realizando correlaciones generalmente basadas en patrones de transformación estáticos o dinámicos.

Tradicionalmente las soluciones para garantizar un servicio, como el procesamiento y correlación de eventos, a través de sofisticados algoritmos, son llevadas a cabo con máquinas muy potentes y por consiguiente, de costes elevados. Sin embargo, hay diferentes soluciones que proponen arquitecturas de procesamiento de eventos distribuidos realizando agregaciones y correlaciones asociativas, como las siguientes: (Pencolé 2001), (Martin-Flatin 2004) y (Leitner, y otros 2007).

Los sistemas basados en eventos han ganado rápidamente importancia en muchos dominios de aplicación desde la monitorización de sistemas en tiempo real, hasta la gestión y el modelado de casos de negocio en seguridad y finanzas.

Una importante tecnología que ha estado emergiendo en los últimos años es el CEP o "*Complex Event Processing*"(Luckham y Frasca 1998) (también llamado "*Event Stream Processing*" o ESP). Esto se entiende como un concepto de procesamiento de eventos que trata sobre la tarea de procesamiento de múltiples eventos con la meta de identificar los más significativos dentro de la nube de eventos generados por el sistema. CEP emplea técnicas como la detección de patrones complejos de multitud de eventos, correlación y abstracción de eventos, jerarquías de eventos, o incluso, diferentes relaciones entre eventos como la causalidad, la temporización, etc.

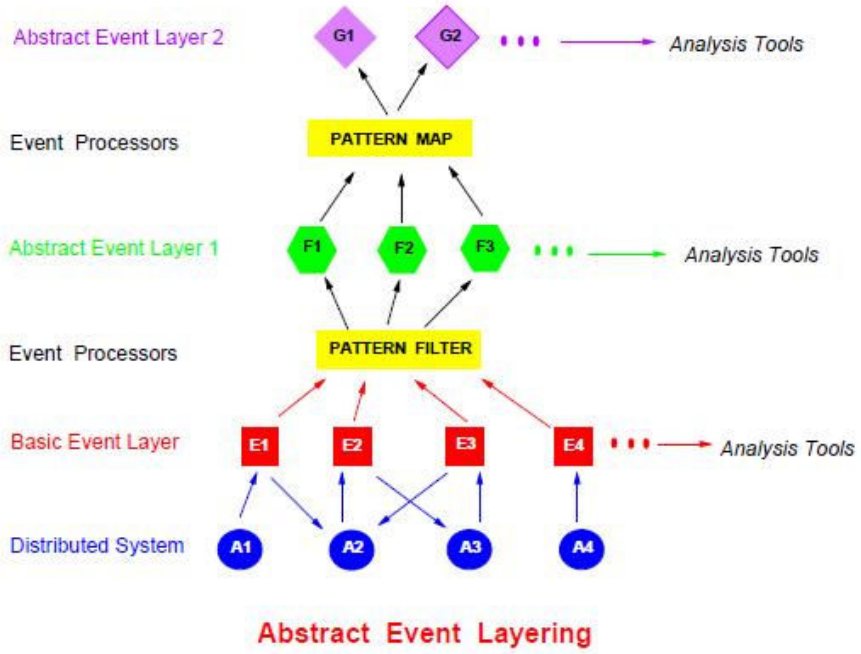

ILUSTRACIÓN 2: ORGANIZACIÓN JERÁRQUICA EN EL PROCESAMIENTO DE EVENTOS

Tras mostrar algunas de las ventajas del CEP, hay que puntualizar el punto débil de las tecnologías de procesamiento distribuido. Éste es la complejidad de los sistemas de mensajería que deben asegurar la entrega ordenada en todos los extremos del escenario de acción. Dependiendo del nivel de perfección (retardo temporal,

mensajería causal o completamente ordenada, etc.) que se requiera para la aplicación, existen diferentes protocolos y variantes para las diferentes implementaciones de los mencionados sistemas de mensajería (Collouris, Dollimore y Klindberg 1994) (Babaoglu y Marzullo 1993).

2.3 GESTIÓN AUTONÓMICA

Los nuevos servicios de internet se ofrecen en estructuras diversas e inter-relacionadas entre sí, además de tener un soporte multi-tecnológico para cada servicio. Con respecto a esto, las soluciones autonómicas proponen el concepto de sistemas auto-gestionados. El objetivo es proveer a los elementos de la red de ciertas capacidades de gestión que les permitan reaccionar automáticamente con su entorno mediante la realización de tareas tales como la auto-configuración, auto-reparación, etc.(Jennings, y otros 2007).

Los conceptos alrededor de la gestión autonómica se enfocan sobre un elemento de gestión fundamental: el bucle de control(Murch 2004). Dicho bucle consiste en componentes que son responsables de monitorizar la entidad gestionada, recolectando información relevante de ella y del entorno en el que ésta está operando. Estos datos son continuamente analizados y comparados con el estado deseado; y si el estado no es el deseado (se ha dado un fallo o la configuración no es óptima), se toma una acción para intentar solucionar este problema.

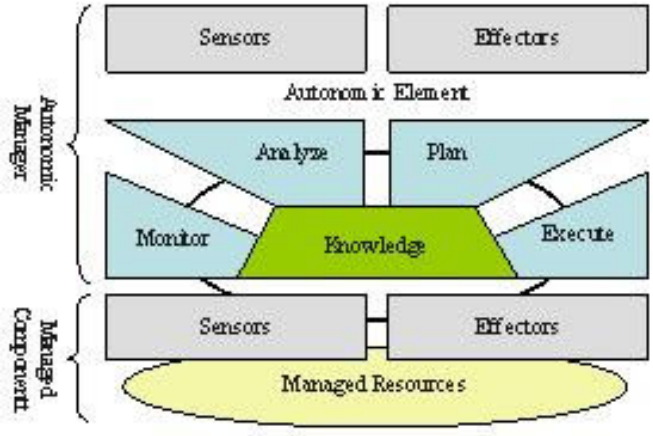

ILUSTRACIÓN 3: BUCLE DE CONTROL

La implementación de este bucle de control es conocida como MAPE, que significa "*Monitor, Analyse, Plan and Execute*". La función de monitorizar está directamente relacionada con la recolección de datos y su correspondiente filtrado, sólo entonces esta información será entregada a la función de análisis. Dicha función sirve para entender los datos recolectados y, tras esto, determinar si la entidad gestionada está

realizando su tarea como se desea. La función de planificación recibe la información si el módulo de análisis lo considera oportuno y entonces, se determina si una acción debe ser llevada a cabo para reconfigurar debidamente la entidad gestionada usando políticas predefinidas que establecen unos objetivos para cumplir con la gestión autonómica. La función de ejecución traduce el plan en un conjunto de comandos que directamente llevan a cabo cualquier reconfiguración requerida.

No obstante, es obvio que este ciclo de monitorización y acción es aplicable a muchos campos de acción, como se puede observar en el modelo "Círculo de Deming" (Walton 1986) muy utilizado en los "Sistemas de Gestión de Seguridad de la Información" (SGSI, o "*Information Security Management System*", ISMS) que aúnan diferentes políticas de gestión de la información.

Sin embargo, existen más trabajos relacionados con la gestión autonómica. Por ejemplo, la arquitectura FOCALE(Strassner, Agoulmine y Lehtihet 2007) se construye sobre el trabajo de IBM, pero añade una dimensión extra con miras hacia las necesidades de un nivel de negocio más elevado, como la gestión de servicios. Además del bucle simple de gestión autonómica antes mencionado; esta arquitectura sugiere un grupo de funciones automáticas que incluyen la auto-configuración, la auto-reparación, la auto-optimización y la auto-protección. FOCALE reconoce que para implementar estas cuatro "auto-funciones", los sistemas deben ante todo ser capaces de conocerse a ellos mismos y a su entorno.

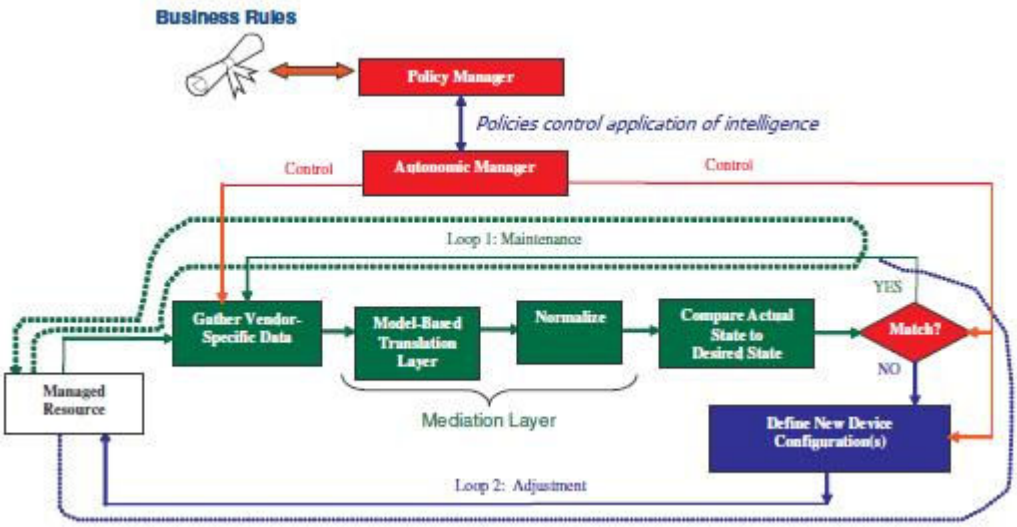

ILUSTRACIÓN 4: ARQUITECTURA FOCALE

Y para terminar, el principal inconveniente es la complejidad conjunta del sistema, ya que requiere una serie de recursos que podrían sobrecargar a pequeños dispositivos, y además, la necesidad de implementar diferentes métodos de medida y reconfiguración tiene una dificultad variable dependiendo del sistema al que se le quiera aplicar. Con esto, la aplicación más viable de estos conceptos de gestión

autonómica es en nuevos dispositivos en cuyos diseños se tengan en cuenta estas necesidades desde el inicio; ya que adaptarlo a antiguos dispositivos puede ser muy complejo.

2.4 GESTIÓN DE LOS EXTREMOS DE LA RED

En las redes finales de usuario, las operadoras de servicios encuentran un hito difícil de superar debido a la privacidad de información de éste, además de dispositivos con unos recursos heterogéneos y escasos. No obstante, un cierto número de enfoques diferentes compiten por la gestión de las redes del hogar, entre las que se incluyen DSL Forum TR-069(DSL Forum Technical Report 2007) o PacketCable(Miller, Andreasen y Russell 2001). La mayoría de estos enfoques están relacionados con un servidor central para la gestión remota de los dispositivos, llamado *Auto-Configuration Server* (ACS).

El protocolo más conocido entre los equipos finales de usuario (CPE o "*Customer Premises Equipment*") y la entidad de gestión central es el Protocolo de Gestión CPE WAN TR-069(DSL Forum Technical Report 2007). La principal funcionalidad implementada actualmente por los productores está relacionada con la auto-configuración y la gestión del ciclo de vida del software. El protocolo TR-069 también soporta un número de otros estándares (existentes o en desarrollo) para realizar una monitorización de la calidad del servicio ofrecido.

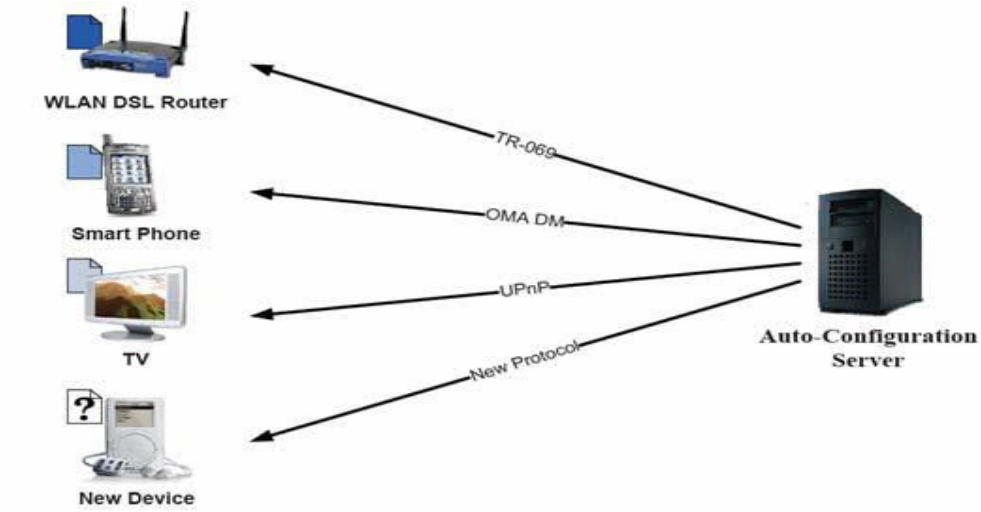

ILUSTRACIÓN 5: SERVIDOR ACS

Además de estos enfoques y protocolos, existen varias tecnologías para interconectar dispositivos del hogar y proveer un entorno de ejecución del servicio, siendo OSGi (OSGi Alliance 2009) la más extendida entre ellas. La especificación de la plataforma de servicios OSGi permite el despliegue y la gestión remota de servicios en la red del

hogar. Los componentes del servicio son desplegados como "*bundles*" por encima de la plataforma OSGi, la cual provee como funcionalidades básicas la gestión del ciclo de vida de los servicios, el registro y descubrimiento de servicios, almacenamiento de datos con persistencia, seguridad y compartición de código. Ha de anotarse que, si se desea usar dicha plataforma de servicios, la gestión del hogar ha de ser considerada y, por lo tanto, también modelada como un servicio, el cual será desplegado como un "*bundle* OSGi".

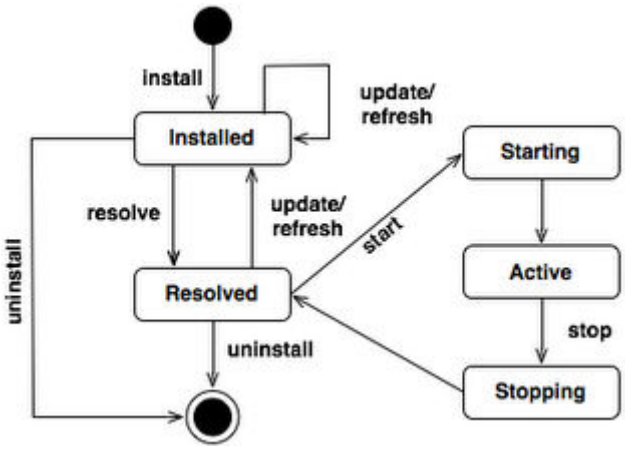

ILUSTRACIÓN 6: CICLO DE VIDA DE UN BUNDLE OSGI

A esto, se le debe sumar la necesidad de disponer suficientes recursos de procesamiento en los diferentes dispositivos que se desean gestionar, ya que la mayoría de éstos suelen tener las mínimas capacidades posibles para reducir al máximo los costes de producción. Aún así, el mercado tiende a evolucionar las pasarelas residenciales del usuario para poder disponer de más capacidad de procesamiento dentro de las redes finales del usuario.

2.5 TÉCNICAS PROBABILÍSTICAS DE GESTIÓN

Los proveedores de servicios de telecomunicaciones necesitan tener información detallada para poder conocer la calidad de los servicios que ofrecen. Basándose en este conocimiento, ellos deben ser capaces de reconfigurar sus productos y servicios para utilizar la red de la manera más eficiente posible.

La gestión de red cuenta con algoritmos deterministas que, con un conjunto de datos, son capaces de determinar el estado del sistema y la causa de los problemas detectados. Sin embargo, no siempre es posible conocer toda la información necesaria o puede no ser conveniente recolectarla por cuestiones de eficiencia y sólo se consigue un subconjunto de los datos necesarios que permiten alcanzar unas conclusiones válidas con un cierto margen de error.

Los enfoques probabilísticos son muy usados, y además con resultados muy satisfactorios, en diferentes disciplinas científicas como la sismología (Turcotte 1989), el control del tráfico aéreo (Cheng, Crawford y Menon 1999), el análisis de inversiones de riesgo (Granger, Henrion y Small 2007) o en el suministro de agua(Cui y Kuczera 2003). El uso de estos enfoques permite el diseño de modelos muy ricos que de otro modo serían casi imposibles de modelar.

Diferentes enfoques de gestión explotan este potencial, de ellos, los más conocidos son los enfoques de monitorización que usan técnicas de muestreo para recolectar datos útiles para establecer el estado actual del sistema. Estas técnicas de muestreo han demostrado ser muy útiles para monitorizar el nivel IP de la red (Kandula, Katabi y Vasseur 2005); para funciones de monitorización distribuida que permitan calcular, casi en tiempo real, una estimación de flujos de VoIP en redes grandes (Dantu y Kolan 2005); o para filtros anti-spam en listas de correo (Andoutsopoulos, y otros 2003). Además de reducir drásticamente el coste de gestión (por ejemplo, en el coste de sondas físicas, recolectores y almacenadores de datos, etc.), las técnicas probabilísticas permiten hacer cálculos con exactitud de los datos recolectados.

Como en otras muchas disciplinas, la incertidumbre es un factor ineludible en la gestión de redes y servicios. Mientras que dicha incertidumbre puede verse como un factor que debe ser mitigado, también puede enfocarse como un facilitador para el diseño de algoritmos que puedan procesar, predecir e incluso influenciar en el comportamiento del sistema gestionado. La necesidad de tratar con la incertidumbre en el plano de gestión se hace obvia cuando se considera la gestión de las redes modernas, especialmente de las redes ad-hoc (Badonnel, State y Festor April 2006).

Aunque las diferentes técnicas probabilísticas de gestión pueden manejar grandes cantidades de datos que un experto en la materia tardaría un tiempo excesivo en evaluar; el principal inconveniente es el modelado y la correlación de la información que el diseñador del sistema debe hacer para intentar conseguir que el comportamiento de éste lo más semejante a la decisión que tomaría el experto en dicha situación.

3. REDES BAYESIANAS

3.1 INTRODUCCIÓN

La principal idea que motivó el estudio de las redes Bayesianas fue la de buscar un modelo computacional para el razonamiento de inferencias de los humanos, nominalmente, el mecanismo por el cual la gente integra datos de múltiples fuentes diferentes y genera una interpretación coherente de dichos datos.

Y aunque los sistemas de razonamiento basados en reglas habían demostrado tener sus campos de uso en los cuales se adaptaban perfectamente al problema manejando factores certeros; desde hacía tiempo se había observado que presentaban ciertas limitaciones en la representación del conocimiento y el razonamiento bajo incertidumbre.

Con estos hechos, los investigadores pusieron su atención en los modelos de interpretación probabilística con factores discretos. Dicho esfuerzo condujo a la definición de las redes Bayesianas (Pearl y Kim, A computational model for causal and diagnostic reasoning in inference systems 1983)(Pearl, Bayesian Networks: A model of self-activated memory for evidential reasoning 1985).

3.2 DEFINICIÓN

Una red Bayesiana se puede definir brevemente como un modelo gráfico acíclico directo (*Directed Acyclic Graph*, DAG) en el que se define una dependencia entre un conjunto de distribuciones probabilísticas de las variables que están representadas en los nodos del DAG. Dichas dependencias están definidas en los arcos del propio DAG.(Kjaerilff y Madsen 2008)

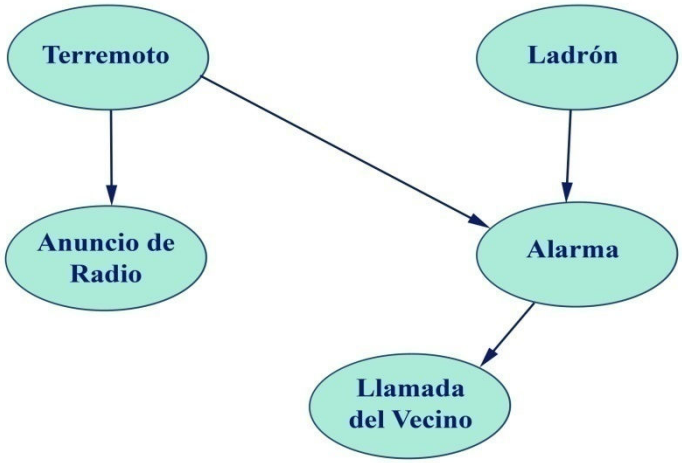

ILUSTRACIÓN 7: EJEMPLO DE RED BAYESIANA

Más concretamente, para un DAG, $\vartheta = (V, E)$, donde V es el conjunto de nodos y E es el conjunto de arcos directos entre parejas de nodos. El conjunto de distribución de probabilidades, $P(X_V)$, de un conjunto de variables (normalmente discretas) dadas en V, puede ser representado como:

$$P(X_V) = \prod_{v \in V} P\left(X_V \middle| X_{pa(v)}\right)$$

Donde $X_{pa(v)}$ representa el conjunto (preferiblemente pequeño) de variables padre de la variable V para cada $v \in V$. La función anterior tiene en cuenta un conjunto de dependencias no representadas directamente en el DAG en términos de parejas de nodos unidas directamente por arcos. Esto unido al conjunto de nodos padre de cada nodo, hace posible la definición de probabilidades condicionales y la realización efectiva de inferencia en redes Bayesianas.

Cada distribución de probabilidad condicional, $P\left(X_V \middle| X_{pa(v)}\right)$, representa un conjunto de "reglas"; donde cada "regla" (probabilidad condicional) toma la forma de:

Si $X_{pa(v)} = x_{pa(v)}$ entonces; $X_V = x_V$ con una probalidad z

Donde x_V y $x_{pa(v)}$son, respectivamente, un valor asignado a X_V y un vector de valores asignados a las variables padres de X_V. Por ejemplo, dado uno de los cinco posibles valores que pudiesen ser asignados a la variable X_V y dado que tuviera cuatro nodos padre, cada uno de los cuales tuviera tres posibles estados, entonces $P\left(X_V \middle| X_{pa(v)}\right)$ representa una colección de $5 \times 3^4 = 405$ reglas del tipo a la mostrada con anterioridad.

Tras decir esto, se ha de remarcar que el concepto de regla aparece sólo implícitamente dentro de las redes Bayesianas. Lo que explícitamente define a las redes Bayesianas son las distribuciones probabilísticas condicionales, donde cada regla (como la mostrada anteriormente) se expresa como una probabilidad condicional de la siguiente forma:

$$P\left(X_V = x_V \middle| X_{pa(v)} = x_{pa(v)}\right) = z$$

O, de manera simplificada:

$$P\left(x_V \middle| x_{pa(v)}\right) = z$$

3.3 INFERENCIA EN REDES BAYESIANAS

Al contrario que los sistemas basados en reglas, la inferencia en las redes bayesianas es siempre coherente y la habilidad de manejar el problema de manera explicativa está empotrada naturalmente en el modo en el que la inferencia es realizada en las redes bayesianas. Sin embargo, en general, resolver la inferencia en las redes Bayesianas es un gran problema (Cooper 1990)(Dagum y Luby 1993). Afortunadamente, han sido desarrollado algoritmos eficientes de inferencia tales que la inferencia en redes Bayesianas puede ser realizada en fracciones de segundo incluso en redes con cientos de variables (Lauritzen y Spiegelhalter 1988)(Jensen, Lauritzen y Olesen 1990). Dicha eficiencia en la inferencia, sin embargo, está fuertemente ligada a la estructura del DAG, así las redes con un número relativamente pequeño de variables, a veces, pueden no tener una inferencia exacta, en cuyo caso se pueden usar métodos aproximados.

Las redes Bayesianas suelen representar declaraciones causales, del tipo $X \rightarrow Y$, donde X es la causa de Y, y donde Y suele tomar el rol de un efecto observable de X que no puede ser observado por sí mismo. Por lo tanto, debemos derivar en posteriores distribuciones de probabilidad $P(X|Y = y)$ dada la observación de $Y = y$ usando la anterior distribución de probabilidad $P(X)$ y la distribución de probabilidad condicional $P(Y|X)$especificada explícitamente en el modelo. Thomas Bayes (1702-1761) promovió el famoso teorema de Bayes realizando el siguiente cálculo:

$$P(X|Y=y) = \frac{P(Y=y|X)P(X)}{P(Y=y)},$$

Donde $P(Y=y) = \sum_x P(Y=y|X=x)P(X=x)$. Esta regla (o teorema) tiene un papel central en la inferencia porque la probabilidad de una causa puede ser deducida cuando su efecto ha sido observado. Primero se desarrolló un método para hacer inferencia en redes Bayesianas que involucraba múltiples aplicaciones del teorema de Bayes(Olmsted 1983)(Shachter 1986). Más tarde, apareció otro método basado en el traspaso de mensajes en una estructura en árbol ("*Junction Tree*") derivado de la estructura de la red Bayesiana(Lauritzen y Spiegelhalter 1988).

El algoritmo más moderno y más utilizado actualmente ("*Recursive Conditioning*") trata de hacer suposiciones para reducir el coste computacional del algoritmo (Darwiche 2000).

3.4 CONSTRUCCIÓN DE REDES BAYESIANAS

Como ya se ha dicho anteriormente, una red bayesiana puede ser descrita como un grafo acíclico directo (DAG) y un conjunto de distribuciones de probabilidad que representan un conjunto de un conjunto de probabilidades condicionales definidas por las estructura del DAG.

Ahora, basándonos en esto, la construcción de una red Bayesiana se divide en dos fases:

1. Dado un problema, se identifican las variables relevantes y las relaciones causales entre ellas. Como resultado de este proceso, se dará un DAG.
2. Este DAG especifica un conjunto de dependencias e independencias asumidas que se enmarcarán en un conjunto de distribuciones probabilísticas, las cuales tendrán que ser expresadas en distribuciones condicionales.

Una vez se obtenga el DAG y el conjunto de distribuciones de probabilidad se tendrá terminada la red Bayesiana. No obstante, hay que mencionar que este proceso puede ser manual, semi-automático o automático; dependiendo de los datos *a priori* que se tengan y que la construcción manual puede ser una tarea laboriosa que requiera tanto una gran habilidad y creatividad como un conocimiento experto sobre el dominio del problema(Kjaerilff y Madsen 2008).

3.4.1.1 APRENDIZAJE EN REDES BAYESIANAS

Principalmente, en el dominio de las redes Bayesianas, hay dos tipos de aprendizaje claramente diferenciados: el aprendizaje estructural y el aprendizaje paramétrico. La

diferencia entre estos dos tipos de aprendizaje son los requisitos de entrada. Para ejecutar un algoritmo de aprendizaje estructural, se necesita una gran cantidad de datos para poder alcanzar una red Bayesiana con un cierto nivel de confianza. Mientras que para una el aprendizaje paramétrico, se necesita la estructura de la red Bayesiana, además de los datos ya mencionados; es decir, un DAG con las relaciones causales que este conlleva.

3.4.1.2 APRENDIZAJE ESTRUCTURAL

La forma de construir algoritmos de aprendizaje estructural para redes Bayesianas es definir dos componentes (Witten y Frank 2005):

- Una función para evaluar una red Bayesiana dada y unos datos sobre los que evaluarla
- Y un método de búsqueda dentro del espacio de las redes posibles dentro del dominio del problema.

En este tipo de aprendizaje, los nodos de la red están predeterminados, al igual que los posibles estados de estos. En otras palabras, las variables que entrarán en juego en el proceso de aprendizaje están implícitas en los datos de entrada. El aprendizaje de la estructura final de la red se encuentra buscando a través de todo el espacio de posibles conjuntos de arcos, estimando las tablas de probabilidades condicionales (CPT) para cada conjunto y procesando la probabilidad resultante de cada nodo basada en los datos como una medida de la calidad de la red. Los algoritmos de aprendizaje estructural difieren principalmente en el modo en el que navegan o buscan a través de todo el espacio de estructuras potenciales de la red.

Pero tras esto, hay que tener en cuenta que la forma más exacta de representar un problema a través de una red Bayesiana, implica un mayor número de arcos, es decir, un mayor número de CPT's que pueden ser muy complejas y por lo tanto aumentar la complejidad de la inferencia. Para evitar esto, se pueden emplear varios métodos. Uno de ellos podría ser una validación cruzada para evaluar el grado de fiabilidad de la red. Otro puede ser el añadir un penalizador a las redes basándose en su complejidad, esto es, en el número total de relaciones estimadas independientes de todo el conjunto de tablas de probabilidad. Dentro de esta categoría, existen diferentes criterios, los dos más populares son: *Akaike Information Criterion* (AIC)(Akaike 1974) y la métrica MDL (*Minimum Description Length)*(Roure Alcobé 2007). Una tercera posibilidad es asignar una distribución prioritaria sobre la estructura de la red y encontrar la combinación más parecida a ésta, lo cual ya implica tener un cierto conocimiento sobre el dominio del problema en cuestión.

Como conclusión, la búsqueda de una buena estructura para una red Bayesiana puede simplificarse mucho si se usan las métricas correctas.

Algunos de los algoritmos específicos más conocidos son:

- el *K2* (Ruiz 2005) que depende del orden de entrada de los parámetros y por lo tanto, aunque es un algoritmo bastante rápido, hay que ejecutarlo repetidas veces alterando el orden de los datos de entrada para conseguir una confianza suficiente sobre el resultado. No obstante, existen diferentes trucos para hacerlo más eficiente, como por ejemplo, usar la capa de Markov(Pearl y Kauffman, Probabilistic Reasoning in Intelligent Systems: networks of plausible inference 1988) para observar si se ha mejorado o empeorado con respecto a las iteraciones anteriores.
- Y el *augmented Naïve Bayes (TAN)* (Cerquides y López de Màntaras 2003), que es un buen algoritmo para sistemas clasificadores. Este algoritmo añade nuevos arcos a la estructura de una red anterior y mediante diferentes criterios intenta alcanzar una mayor confianza en los datos de entrada mediante diferentes iteraciones.

3.4.1.3 APRENDIZAJE PARAMÉTRICO

Dependiendo del escenario del problema, puede ser mucho más interesante la construcción de una red Bayesiana de manera manual porque se cuente con la ayuda de un experto en la materia que conoce a priori las relaciones causales entre las diferentes variables. Dicho experto será capaz de generar una red Bayesiana más ajustada a la realidad (al menos estructuralmente) de lo que podría generar cualquier algoritmo de aprendizaje estructural; ya que estos algoritmos, bien por no tener como entrada un conjunto suficientemente grande de datos o por descubrir relaciones causales falsas.

Como ejemplo de esto último, se puede observar la comparación entre diferentes algoritmos de aprendizaje estructural hecha con datos históricos de un hospital; donde se buscaba una red Bayesiana para diagnosticar pacientes en urgencias (Acid, y otros 2004). En este estudio se podía observar como dependiendo del algoritmo, el día de la semana podría ser una variable a tener en cuenta o no. Y, por supuesto, un experto en diagnóstico médico podría haber proporcionado una red Bayesiana más acorde con las verdaderas relaciones causales entre las variables y, una vez alcanzada una estructura correcta, ajustar sus valores del conjunto de distribuciones probabilísticas con algoritmos de aprendizaje paramétrico.

Dentro de los posibles algoritmos de este tipo de aprendizaje, el más usado es el *Maximum Likelihood Estimator* (Liu y Soetjipto 2004) que además puede usar una corrección Bayesiana para los parámetros indefinidos en un conjunto de datos dado. Esto consiste en asignar un valor de entropía máxima a las variables que no aparezcan en el conjunto de entrada, pero sí en la estructura ya definida de la red. El uso de esta

corrección Bayesiana hace que las distribuciones probabilísticas resultantes del algoritmo no sean tan extremas. El funcionamiento de este algoritmo es muy sencillo, simplemente debe buscar las relaciones conocidas de antemano (ya que la estructura de la red viene dada) entre el conjunto de datos de entrada.

4. HERRAMIENTAS UTILIZADAS

4.1 HERRAMIENTAS DE REDES BAYESIANAS

Las herramientas más importantes en el manejo de redes bayesianas (tanto en su uso dinámico para la inferencia, la construcción o el aprendizaje) son: *BNJ*, *MSBNx, Netica, SamIam* y *Genie & SMILE*.

4.1.1 HERRAMIENTAS EVALUADAS

4.1.1.1 BNJ

BNJ (Bayesian Networks tools in Java) (http://bnj.sourceforge.net/) es un conjunto de herramientas software para la investigación y desarrollo que usa modelos gráficos de probabilidad. Está implementada completamente en Java y distribuida bajo licencia GPL (http://www.gnu.org/copyleft/gpl.html) por el Laboratorio de Descubrimiento de Conocimiento en Bases de Datos (http://www.kddresearch.org/) de la Universidad del Estado de Kansas (http://www.cis.ksu.edu/). Tiene como ventaja que está completamente escrita en JAVA; sin embargo, el inconveniente de esta herramienta es

que su desarrollo/mantenimiento está abandonado desde hace varios años y aún con diferentes errores por corregir.

4.1.1.2 MSBNx

MSBNx (*Microsoft Bayesian Network Editor and Tool Kit*) (http://research.microsoft.com/en-us/um/redmond/groups/adapt/msbnx/) es una aplicación de Windows para la creación y la evaluación de redes Bayesianas, creado por "*Microsoft Research*". Esta herramienta presenta la opción de guardar las redes Bayesianas en un formato XML que permite una fácil gestión de los datos; no obstante las características que ofrece son bastante pobres comparada con otras herramientas.

4.1.1.3 NETICA

Netica (http://www.norsys.com/netica.html) es un programa potente, completo y de fácil uso de ámbito comercial que ofrece diferentes tipos de inferencia. Como principal diferencia, *Netica* ofrece la posibilidad de encontrar decisiones óptimas para obtener para reducir el mínimo la entropía que se presenta en un instante dado. Con esto puede construir planes condicionales, tomando decisiones en el futuro dependiendo de observaciones que aún no se hayan hecho teniendo en cuenta temporizadores e inter-relaciones entre nodos. Este paquete software ha sido desarrollado por *Norsys Software Corp.* (http://www.norsys.com/).

4.1.1.4 SAMIAM

SamIam (Sensitivity Analysis, Modelling, Inference And More) (http://reasoning.cs.ucla.edu/samiam/) es una herramienta para el modelado y el razonamiento con redes Bayesianas, desarrollada en Java por el Grupo de Razonamiento Automatizado del profesor Adnan Darwiche en la Universidad de California, Los Angeles (UCLA). *SamIam* incluye dos principalmente dos componentes: una interfaz gráfica de usuario y un motor de razonamiento. La interfaz gráfica permite modelar y guardar las redes en diferentes formatos. Y el motor de razonamiento soporta muchas tareas diferentes, en otras, inferencia bayesiana y aprendizaje paramétrico.

4.1.1.5 GENIE & SMILE

Y finalmente, *Genie & SMILE* (http://genie.sis.pitt.edu/). SMILE (*Structural Modelling, Inference, and Learning Engine*) es una herramienta implementada en C++ que ofrece métodos gráficos de soporte de decisión, como redes Bayesianas y diagramas de influencia, fácilmente portables a sistemas inteligentes. Su interfaz gráfica de usuario en Microsoft® Windows, *Genie* es un versátil y amigable entorno de desarrollo para los modelos gráficos de decisión. Ambos módulos, desarrollados por el Laboratorio de

Sistemas de Decisión de la Universidad de Pittsburgh, han estado disponibles desde Julio de 1998 y ahora tienen varios miles de usuarios en todo el globo. SMILE también dispone de *"wrappers"* para JAVA (*jSMILE)* y .NET *(SMILE.NET)*.

4.1.2 COMPARATIVA

Herramienta	Algoritmos de inferencia	Lenguaje de programación	Mantenimiento	Formato de datos	Otros
BNJ	Bajo	JAVA	Nulo	Propio	Gratuita
MSBNx	Bajo	C++	Medio	XML	Gratuita
Netica	Alto	C++	Alto	XML/otros	Comercial
SamIam	Alto	JAVA	Alto	XML/otros	Gratuita
Genie & SMILE	Alto	C++ (Wrapper JAVA)	Medio	XML/otros	Gratuita GUI

TABLA 2: COMPARATIVA DE HERRAMIENTAS DE REDES BAYESIANAS

4.1.3 CONCLUSIÓN

Las herramientas usadas finalmente, han sido *SamIam* y *Genie & SMILE*. *SamIam* se utilizó, dada su implementación en Java, para la inferencia dentro del sistema, ya que facilitaba enormemente la distribución del software; mientras que *Genie & SMILE* se usó, principalmente, para la edición gráfica de las redes bayesianas, ya que ofrece una interfaz gráfica más completa y más acorde con las características requeridas.

4.2 HERRAMIENTAS DE ONTOLOGÍAS

Con respecto a la creación y uso de ontologías existen múltiples herramientas útiles, pero aquí sólo se han evaluado las siguientes: *KAON, KAON2, Hozo, Ontostudio, Ontolingua, Knoodl y Protégé*.

4.2.1 HERRAMIENTAS EVALUADAS

4.2.1.1 KAON

KAON (http://kaon.semanticweb.org/) es una infraestructura de gestión de ontologías de código abierto orientada a aplicaciones de negocio. Esto incluye un conjunto de herramientas colaborativas de creación y gestión para construir aplicaciones basadas en ontologías. Un punto fuerte de *KAON* es la escalabilidad y la eficiencia en el razonamiento al trabajar con ontologías.

4.2.1.2 KAON2

KAON2 (http://kaon2.semanticweb.org/) es una infraestructura para la gestión de ontologías OWL-DL, SWRL y F-logic. Es la evolución del anteriormente mencionado *KAON* el cual usaba extensiones de propiedades RDFS, mientras que *KAON2* está basado en OWL-DL y F-logic.

Hay que destacar que *KAON2* es un sistema completamente diferente a *KAON* no una nueva versión y tampoco es compatible con su antecesor.

Tanto *KAON* como *KAON2* han sido desarrollados conjuntamente por el *Grupo de Ingeniería de Procesos de Información* del *Centro de Investigación para Tecnologías de la Información*, por el *Instituto de Informática Aplicada y Métodos de Descripción Formal* de la *Universidad de Karlsruhe* y por el *Grupo de Gestión de la Información* de la *Universidad de Manchester*.

4.2.1.3 HOZO

Otra de las diferentes herramientas es *Hozo* (http://www.hozo.jp/hozo_eng/), un editor gráfico de ontologías creado especialmente para la creación de ontologías pesadas o con el conocimiento más adaptado a la realidad posible (Mizoguchi, y otros 2007).

Esta herramienta fue desarrollada en Japón a través de una colaboración entre el *Departamento de Sistemas de Conocimiento (Laboratorio Mizoguchi, Universidad de Osaka)* (http://www.osaka-u.ac.jp/en) y la empresa *Enegate Co., Ltd.* (http://www.osaki.co.jp/).

4.2.1.4 ONTOSTUDIO

Otra herramienta, pero comercial en este caso, es *Ontostudio* (http://www.ontoprise.de/en/home/products/ontostudio/). Presenta una fácil integración con bases de datos gracias a su interfaz gráfica, también permite exportar consultas a las ontologías en el formato de *WebService*. Además, ofrece la posibilidad de desarrollar ontologías de manera colaborativa gracias a otra herramienta de la misma empresa llamada servidor *OntoBroker Collaboration*. *Ontostudio* ha sido desarrollado por la empresa *Ontoprise* (http://www.ontoprise.de/).

4.2.1.5 ONTOLINGUA

Ontolingua (http://www.ksl.stanford.edu/software/ontolingua/) provee un entorno de desarrollo colaborativo conceptualmente similar al anterior para trabajar con ontologías. Este entorno ha sido desarrollado por la *Universidad de Stanford*.

4.2.1.6 KNOODL

Knoodl es otra herramienta que está orientada al desarrollo de la comunidad de ontologías OWL y bases de conocimiento RDF. También sirve como plataforma de tecnología semántica, ofreciendo una interfaz basada en servicios JAVA y otra basada en SPARQL con las que la comunidad puede construir sus propias aplicaciones semánticas usando sus ontologías y bases de conocimiento.

Knoodl es un producto de *Revelytix, Inc.* (http://www.revelytix.com/) y está corriendo en los servidores de Amazon EC2 (http://aws.amazon.com/ec2/) siendo así su uso libre; aunque también se pueden obtener licencias para su uso privado.

4.2.1.7 PROTÉGÉ

Y finalmente, *Protégé* (http://protege.stanford.edu/). Desarrollado por el *Centro de Investigaciones Informáticas de Biomedicina de Standford*, es la herramienta de construcción de ontologías más usada actualmente. Permite crear fácilmente clases y jerarquías, declarar propiedades para las clases, crear instancias e introducir valores, todo ello en un entorno gráfico de usuario fácil de usar.

Está implementado completamente en JAVA, lo que ha generado en torno a él toda una comunidad que contribuye activamente a su ampliación con todo tipo de *plug-ins*; lo cual provoca que esta herramienta sea sumamente potente.

4.2.2 COMPARATIVA

Herramienta	Uso colaborativo	Escalabilidad	Eficiencia de razonamiento	Interacción con WebServices	Otros
KAON	Si	Alto	Alto	No	Gratuito
KAON2	Si	Alto	Alto	No	Gratuito Basado en OWL-DL y F-Logic
HOZO	No	Alto	Alto	No	Gratuito
Ontostudio	Si	Medio	Medio	Si	Comercial
Ontolingua	Si	Medio	Medio	Si	Gratuito GUI
Knoodl	Si	Medio	Medio	Si	Gratuito
Protégé	No	Medio	Alto	No	Gratuito GUI Plug-in

TABLA 3: COMPARATIVA ENTRE HERRAMIENTAS DE ONTOLOGÍAS

4.2.3 CONCLUSIÓN

En el proyecto se ha usado *Protégé*, principalmente por el *plug-in* que facilita el uso de la ontología por los diferentes agentes JADE, *OntologyBeanGenerator* (http://protege.cim3.net/cgi-bin/wiki.pl?OntologyBeanGenerator).

Este *plug-in* permite generar archivos JAVA que representan una ontología para que puedan ser usados en entornos JADE. Con esta herramienta, puedes generar ontología con el formato FIPA/JADE desde proyectos *Protégé*, RDF(S) y XML.

4.3 PLATAFORMA DE AGENTES

De las múltiples plataformas de agentes que están disponibles, se han evaluado las más significativas de las que estaban desarrolladas en Java. Éstas han sido las siguientes: *Diet, Jade, Jadex* y *Wade*.

4.3.1 HERRAMIENTAS EVALUADAS

4.3.1.1 DIET

La plataforma de agentes *DIET* (http://diet-agents.sourceforge.net/) está completamente escrita en Java y es de código abierto. Sus mejores características son su ligereza, su escalabilidad, su robustez, su adaptabilidad y su extensibilidad. Está especialmente diseñada para un desarrollo rápido de aplicaciones peer-to-peer y/o aplicaciones adaptativas distribuidas. Y aún con todo esto, su punto débil es su falta de soporte y de mantenimiento, ya que su última versión data del año 2005. Fue desarrollado como parte del proyecto *DIET* conjuntamente por BT (http://www.bt.com/), DFKI (http://www.dfki.uni-kl.de/), Technical University of Crete (http://www.tuc.gr/english/index.html) y la Universidad Carlos III de Madrid (http://www.uc3m.es/).

4.3.1.2 JADE

Jade (Java Agent Development Framework) (http://jade.tilab.com/) es un *framework* completamente implementado en Java. Esto simplifica la implementación de un sistema multi-agente a través de este *middle-ware* que cumple con la especificación FIPA (http://www.fipa.org/) y que además cuenta con una interfaz gráfica que puede ser utilizada tanto en la fase de depuración como en la fase de despliegue. La plataforma de agentes puede estar distribuida a través de diferentes máquinas (en las cuales ni siquiera debe ejecutarse el mismo sistema operativo) y su configuración puede ser controlada a través de una interfaz gráfica de manera remota. La

configuración inicial puede cambiar incluso en tiempo de ejecución moviendo agentes de una máquina a otra cuando sea requerido.

Además de esto, ofrece diferentes *add-ons* que amplían las posibilidades de la plataforma. Existe la posibilidad de desarrollar agentes en dispositivos con bajas capacidades en los que haya una máquina J2ME-CLDC MIDP a través de sus librerías *LEAP*, también disponible en su página web como un *add-on* de *Jade*. También existe otro *add-on* para integrar fácilmente la plataforma *Jade* en un entorno *OSGi*, exportando un servicio de gestión de agentes a través de las especificaciones de la "OSGi Alliance" (http://www.osgi.org/).

Jade es software libre distribuido por Telecom Italia en carácter de código abierto bajo los términos de la licencia LGPL (*Lesser General Public License Version 2*) y ellos mismos ofrecen un buen mantenimiento y un desarrollo constante teniendo nuevas versiones cada cierto tiempo. La última versión de este *framework* es de mediados de 2009.

4.3.1.3 JADEX

Jadex (http://jadex.informatik.uni-hamburg.de/xwiki/bin/view/About/Overview) tiene como propósito principal el desarrollo de un sistema de agentes tan sencillo como sea posible sin sacrificar el potencial expresivo del paradigma de los agentes. *Jadex* está basado en Java, cumple la especificación FIPA (http://www.fipa.org/) y permite el desarrollo de agentes orientados a metas siguiendo el modelo BDI (*Belief-Desire-Intention*). *Jadex* está basado en la plataforma *Jade*. Ésta provee la arquitectura de la plataforma, los servicios básicos y los mecanismos de transporte requeridos en la especificación FIPA.

Este *framework* proporciona un modelo de creencias, metas y planes para toma de decisiones, que conjuntamente con un API, un modelo de ejecución y una funcionalidad genérica reusable permite al desarrollador especificar los agentes en un tipo de ficheros (ADF, *Agent Definition File*) con una estructura XML donde se especifica el estado inicial del agente, además de sus diferentes capacidades.

Jadex fue desarrollado por el Departamento de Informática de la Universidad de Hamburg (http://vsis-www.informatik.uni-hamburg.de/) y su última versión estable es de mediados de 2007, mientras que su última versión beta es de finales del 2008.

4.3.1.4 WADE

Por último, *Wade* (http://jade.tilab.com/wade/) es una plataforma software basada en *Jade* que provee soporte para la ejecución de tareas de acuerdo al paradigma de flujos de trabajo. El componente clave de *Wade* es el "*WorkflowEngineAgent*" que es un agente *Jade* con un motor de flujos de trabajo pequeño y ligero. Esto permite

combinar la expresividad de los flujos de trabajo con un lenguaje de programación como Java. Hay que mencionar que *Wade* ofrece la posibilidad de programar todas las funcionalidades de los agentes, no obstante, también proporciona un pequeño entorno gráfico de desarrollo llamado *Wolf* que facilita la creación de aplicaciones basadas en *Wade*. *Wolf* es un *plug-in* de *Eclipse* lo cual facilita su uso para los desarrolladores habituados a este *workbench*.

Wade es software libre distribuido por Telecom Italia bajo la misma licencia que *Jade* (LGPL). Su última versión fue lanzada el 2 de julio de 2009.

4.3.2 COMPARATIVA

Herramienta	Mantenimiento	Escalabilidad	Soporte modelo BDI	Soporte plataforma OSGI	Otros
DIET	Muy Bajo	Alto	No	No	Gratuito Diseño para escenarios P2P
JADE	Alto	Medio	No	Si	Gratuito
JADEX	Bajo	Medio	Si	No	Gratuito
WADE	Medio	Medio	No	No	Gratuito

TABLA 4: COMPARATIVA DE HERRAMIENTAS DE PLATAFORMA DE AGENTES

4.3.3 CONCLUSIÓN

Para terminar, mencionar que la plataforma utilizada finalmente fue *Jade*, debido a su facilidad de integrarlo con OSGi y al contacto mantenido con Telecom Italia para la resolución de diferentes tareas.

4.4 PLATAFORMA OSGI

Existen muchas implementaciones de la especificación de la plataforma de servicios OSGi, pero sólo se han evaluado dentro de las de código abierto las siguientes: *Apache Felix, Eclipse Equinox, FUSE ESB 4* y *Knopflerfish*.

4.4.1 HERRAMIENTAS EVALUADAS

4.4.1.1 APACHE FELIX

Apache Felix (http://felix.apache.org/) ha sido desarrollado completamente en la Fundación Software Apache. *Felix* es un esfuerzo de la comunidad para la implementación de la plataforma de servicios OSGi (*Release* 4), la cual incluye el

framework OSGi y unos servicios estándar, así como también provee y soporto otras tecnologías relacionadas con OSGi. Actualmente, *Felix* cumple gran parte de la especificación OSGi y siguen trabajando para conseguir una implementación completa. Desdeñando este hecho, el *framework* proporcionado por *Felix* es muy estable y de fácil despliegue.

Esta herramienta está en constante evolución y su última versión data de 18 de febrero de 2010.

4.4.1.2 ECLIPSE EQUINOX

Eclipse Equinox (http://www.eclipse.org/equinox/) tiene como meta ser una comunidad de primera clase de las tecnologías OSGi y fomentar la visión de *Eclipse* como modelo de *bundles*. Además, el proyecto está abierto a la implementación de todos los aspectos de las especificaciones OSGi, la investigación y el desarrollo de futuras versiones de las especificaciones OSGi, etc.

Esta herramienta ha sido desarrollada por la fundación *Eclipse* y su última versión data del 17 de septiembre de 2009.

4.4.1.3 FUSE ESB 4

FUSE ESB 4 (http://fusesource.com/products/enterprise-servicemix4) es una distribución que incluye un completo *framework* OSGi y está basado en *Apache ServiceMix 4*. Esta herramienta soporta OSGi tan bien como cualquier otra integración de estándares en una plataforma única, facilitando así a los desarrolladores para implementar los patrones de integración que necesitan con el modelo de programación de su elección. Sus principales características son la manera rápida y estándar de crear y desplegar software en la plataforma OSGi, el uso nativo de Spring (http://www.springsource.org/) para crear componentes usando Spring XML y la interfaz gráfica de usuario que integra los sistemas usando EIPs (*Enterprise Integration Patterns*).

Esta herramienta ha sido desarrollada por la Comunidad FUSE OPEN SOURCE y su última versión data de principios de 2009.

4.4.1.4 KNOPFLERFISH

Knopflerfish (http://www.knopflerfish.org/) es una distribución de código abierto de OSGi, cuya meta es el desarrollo y la distribución de un código de fácil uso, herramientas y aplicaciones relacionados con OSGi. Lo más destacable de esta herramienta es la posibilidad que ofrece a las empresas de asegurar un soporte completo con la versión *Knopflerfish Pro* que es un paquete comercial que contiene

más funcionalidades, destacando que *Knopflerfish Pro 2.0* está certificado por la *OSGi Alliance* como acatador completo de la especificación OSGi *Release* 4.

Knopflerfish ha sido desarrollada y mantenida por la empresa *MakeWave* (http://www.makewave.com/), esta herramienta lanzó su última versión el 11 de septiembre de 2009.

4.4.2 COMPARATIVA

Herramienta	Mantenimiento	Escalabilidad	Estabilidad	Otros
Apache Felix	Alto	Alto	Alto	Gratuito Gran variedad de *bundles* Fácil despliegue
Eclipse Equinox	Alto	Alto	Medio	Gratuito Implementación bastante completa de la especificación OSGI
FUSE ESB 4	Medio	Medio	Medio	Gratuito Uso nativo de Spring
Knopflerfish	Medio	Medio	Medio	Gratuito Versión profesional comercial

TABLA 5: COMPARATIVA DE HERRAMIENTAS DE PLATAFORMA OSGI

4.4.3 CONCLUSIÓN

Dada su alta estabilidad, su fácil despliegue, la buena y sencilla gestión del software sobre la plataforma y su mantenimiento por parte de la comunidad, la herramienta utilizada en el proyecto como plataforma de servicios OSGi es *Apache Felix*.

5. ANÁLISIS

5.1 INTRODUCCIÓN

En este capítulo se aborda el proceso de análisis que se ha seguido para la obtención de requisitos del sistema. Para ello se muestran en primer lugar los objetivos y el ámbito del proyecto para enmarcar correctamente el área de trabajo.

Una vez presentado el ámbito del proyecto, se muestran los casos de uso y los requisitos derivados de estos.

5.2 OBJETIVO DEL PROYECTO

Dado el crecimiento que las redes de comunicaciones están sufriendo en los últimos tiempos, la dificultad de conocer por completo el estado global de una red y los diferentes estados de cada uno de sus elementos ha aumentado de manera exponencial. Por lo tanto, el diagnóstico de un fallo dentro de una de estas redes suele ser un proceso que se caracteriza por su elevada complejidad y que suele requerir de la actuación de operarios especializados. A esto, se le debe sumar el hecho de que tanto el origen como la posible solución a un problema detectado en la red, en un

servicio o en un dispositivo, en un gran porcentaje de los casos, está fuera del alcance de quien detecta sus síntomas; el ejemplo más evidente de esto es un usuario final que detecta un fallo de servicio.

Dada la situación comentada en el párrafo anterior, se hace evidente la utilidad de una herramienta de autogestión de red y servicio, es decir, una herramienta que, en otras funciones, detecte y resuelva problemas automáticamente sin necesitar la intervención de ningún usuario u operador. Aún así, es patente que la complejidad de dicho sistema, que intentará simular a un operario experimentado, no es trivial.

Una herramienta de autogestión debe llevar a cabo diferentes procesos, que podrían considerarse independientes, aunque conlleven a un mismo fin. De manera simplificada, dichos procesos son el análisis del estado de la red y de sus elementos en cada momento; en caso de darse algún síntoma de fallo o problema se pasaría al diagnóstico de él; y, por último, la resolución del problema para que el sistema, la red o el dispositivo vuelva a un estado de normalidad y funcionalidad correcta siempre que sea posible.

Expuestas las ideas anteriores, el objetivo de este proyecto fin de carrera es el desarrollo de un sistema de diagnóstico distribuido usando redes bayesianas para alcanzar una lista de las causas más probables de fallo. Esta idea se puede plasmar esta misma idea en dos ideas más sencillas:

- El desarrollo de un sistema distribuido para realizar diagnósticos sin sobrecargar puntos críticos de la red y así tener mayor alcance en la obtención de las causas de fallo.
- La exposición de cómo el uso de redes bayesianas permite alcanzar conclusiones manejando durante todo momento la incertidumbre inherente a un proceso de diagnóstico.

5.3 ÁMBITO DEL PROYECTO

5.3.1 ESCENARIO

Para enmarcar correctamente el estudio realizado, se debe anotar que el escenario en el que se plantea el proyecto es una conexión punto a punto en la cual se da un servicio de compartición de contenidos entre dos usuarios finales de la red. En esta situación pueden producirse multitud de fallos, tanto en conexión como en suministración de servicios. El sistema está destinado a ofrecer al usuario final o a un operador el resultado final del diagnóstico realizado una vez recibida una notificación de un síntoma de fallo. En dicho informe, el resultado está expresado en porcentajes que representan la certeza de la ocurrencia de una hipótesis dada.

En el caso ideal en el que este informe de diagnóstico fuera recibido por un sistema de *self-healing* (ver sección 2.3), gracias a este informe, dicho sistema dispondría de las causas más posibles de fallo para poder actuar en consecuencia y restablecer un estado estable y funcional del sistema completo.

Como se ha descrito en el párrafo anterior, el escenario representa una situación en la que dos usuarios pueden compartir contenidos digitales a través de un servicio que se lo permite.

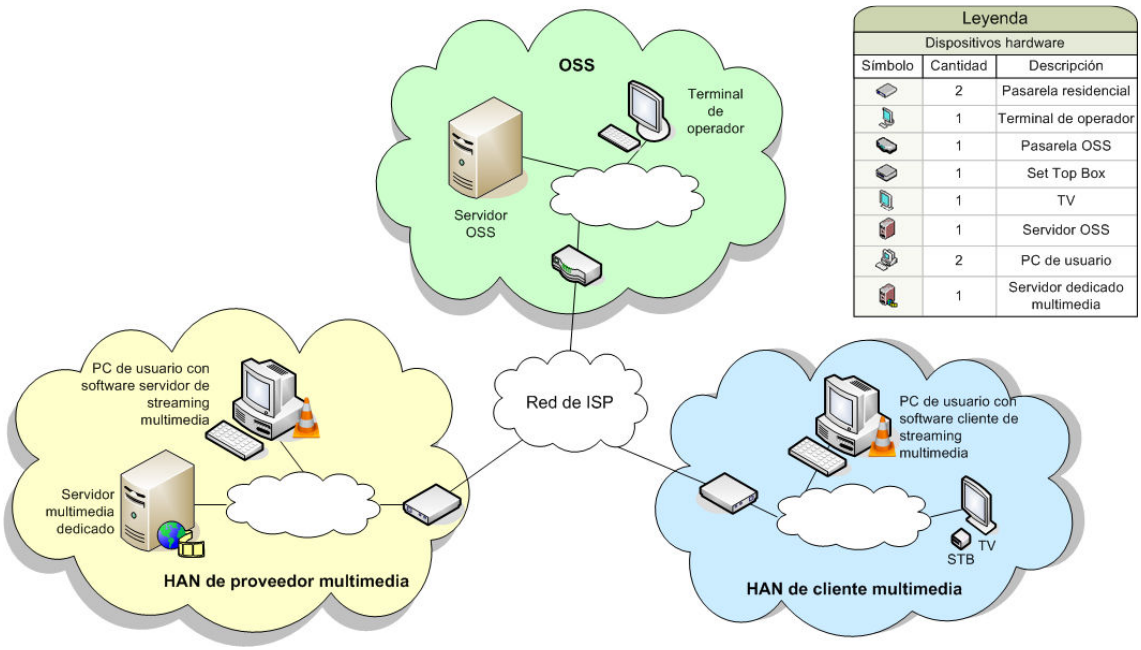

ILUSTRACIÓN 8: ESCENARIO

En la Ilustración 8 se puede observar como los dos usuarios obtienen su conexión a la red a través de una pasarela residencial por la que viajan los datos que comparten dichos usuarios. Hay que apuntar que, en el caso de que el contenido deba ser transmitido en tiempo real (por ejemplo, audio o video), el protocolo usado es RTSP (*Real Time Streaming Protocol*) para iniciar la sesión y RTP (*Real-time Transport Protocol*) para el envío de contenidos.

En este escenario, se diagnostican posibles causas de fallo dado un evento que indica la detección de un síntoma en el acceso a un determinado recurso. Una vez recibido, se inicia el proceso de diagnóstico correspondiente que da como resultado una serie de hipótesis con sus correspondientes porcentajes de certeza.

5.3.2 CONSIDERACIONES DE ENTORNO

Dentro del escenario planteado, se encuentran diferentes consideraciones a tener en cuenta, como las siguientes:

- la necesidad del hardware adecuado para la construcción de un escenario simulado (dada la dificultad de desarrollar un prototipo dentro de un escenario real en el que no se tiene control sobre todos los elementos del escenario),
- la necesidad de unas pasarelas residenciales más potentes de las que normalmente tienen los usuarios finales en sus hogares, ya que en ellas se ejecuta parte del software,
- la necesidad del hardware y software adecuado para realizar una compartición de contenidos multimedia extremo a extremo,
- y por último, la necesidad de usar tecnologías aplicables a un escenario real, es decir, que posibilite una gestión asequible en redes de locales de los usuarios finales.

5.3.3 RELACIONES CON OTROS SISTEMAS

El software de diagnóstico interactúa con varios sistemas dentro de este escenario. Estas interacciones están expuestas claramente en las secciones posteriores donde se muestran los casos de uso del sistema.

Aún así, de manera general, los sistemas son los siguientes:

- Un sistema que sirve como capa de adaptación de los distintos dispositivos generadores de eventos pueden interactuar con el sistema a fin de iniciar un diagnóstico.
- Un sistema de gestión de los recursos compartidos dentro del escenario del cual obtener información para afianzar las hipótesis del diagnóstico.
- Un servidor multimedia al cual se accede para interrogar sobre su configuración y estado.
- Un sistema que sirve como interlocutor entre el sistema de diagnóstico y el usuario mostrando el resultado del diagnóstico a este último.

5.4 CASOS DE USO

En esta sección se identifican los casos de uso generales de uso normales en el sistema, con el objeto de conseguir una especificación completa de utilización esperada del mismo que permita establecer un listado completo de requisitos.

A continuación se expone una tabla de los actores que se identifican dentro de los casos de uso. Después, mediante diagramas UML de casos de uso y unas descripciones de los mismos, se establecen las relaciones entre dicho actores y el sistema.

5.4.1 ACTORES

En la siguiente tabla se muestran los actores principales y secundarios identificados en los casos de uso.

Identificador de actor	Nombre	Rol	Descripción
DAL	Device Adaption Layer	Solicitante de diagnóstico	Generador de eventos
MNG	Multimedia Resources Manager	Sistema de gestión de recursos multimedia	Gestor de recursos compartidos por los usuarios finales
MUS	Multimedia Server	Sistema servidor	Servidor de recursos multimedia
REP	Report Notifier	Sistema de notificación	Receptor de informes de diagnóstico
ADM	Administrator	Administrador	Administrador del sistema

TABLA 6: DICCIONARIO DE ACTORES

5.4.2 CASO DE USO 1: DIAGNÓSTICO DE UN SÍNTOMA

5.4.2.1 DESCRIPCIÓN

El solicitante del diagnóstico envía un evento en el formato adecuado que el sistema recibe. Entonces, el sistema analiza el evento, obtiene información de él y toma una decisión sobre las pruebas a realizar. En la realización de esas pruebas interactúa cuando sea posible con el sistema de gestión de recursos compartidos para solicitarle la información requerida y con el servidor multimedia para evaluar su estado. Una vez realizado el diagnóstico, el sistema notifica al receptor de informes.

5.4.2.2 ESPECIFICACIÓN DE CASO DE USO

CU-1	Diagnóstico de un síntoma
Descripción	El sistema diagnostica un síntoma habiendo recibido un evento.
Actores	DAL, MNG, MUS, REP
Secuencia normal	1. DAL envía un evento 2. El sistema comienza el diagnóstico 3. El sistema realiza diferentes pruebas 4. El sistema solicita información a MNG 5. El sistema evalúa el estado de MUS

	6. El sistema envía un informe del diagnóstico a REP
Excepciones	Los pasos 4 y 5 pueden omitirse dependiendo de la disponibilidad de los sistemas requeridos.

TABLA 7: ESPECIFIACIÓN DE CASO DE USO 1

5.4.2.3 DIAGRAMA DE CASO DE USO

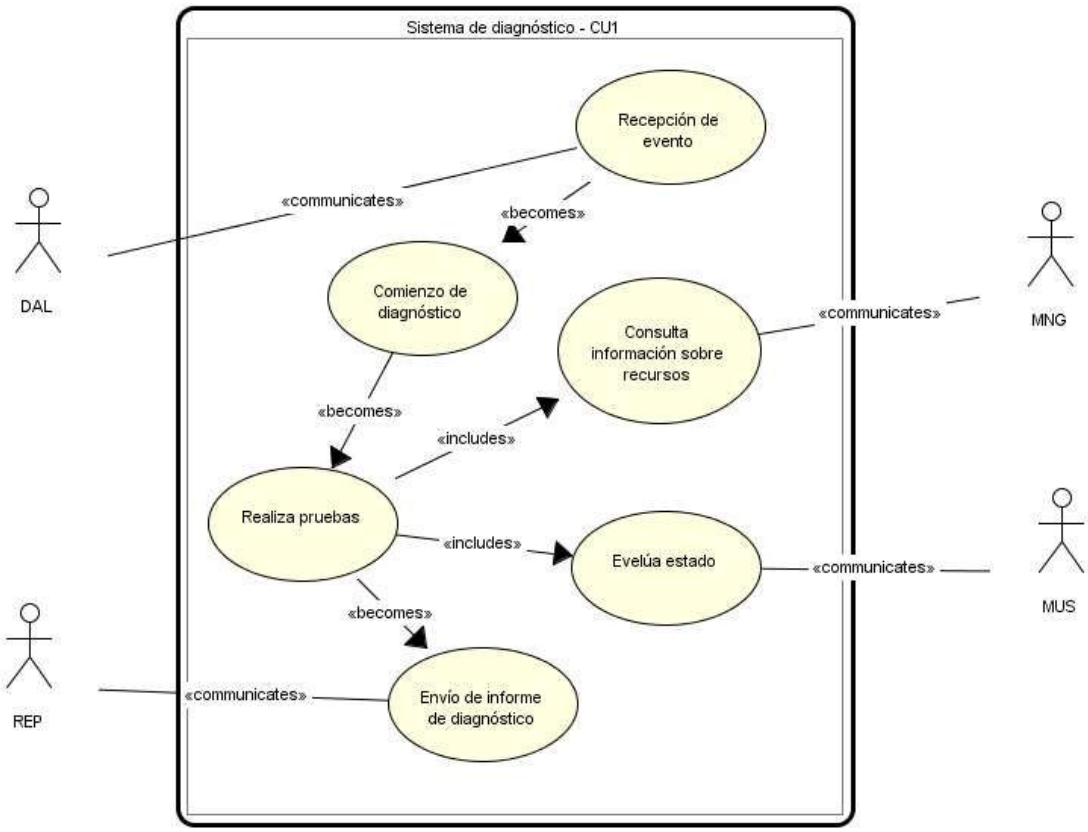

ILUSTRACIÓN 9: DIAGRAMA DE CASO DE USO 1

5.4.3 CASO DE USO 2: ACTUALIZACIÓN DE CONOCIMIENTO

5.4.3.1 DESCRIPCIÓN

El administrador del sistema despliega un nuevo conocimiento que posee en el formato apropiado a través de una pantalla de actualización donde se especifica la ruta del archivo que contiene el conocimiento. Este despliegue sólo se hace en un punto del sistema y automáticamente el conocimiento se envía a todas las partes de la red interesadas en él. Una vez desplegada en todos los puntos de interés, esta última versión es considerada como la más actual y se usa en el resto de diagnósticos. Tras la actualización del conocimiento, el administrador es notificado del éxito o el fracaso de la operación.

5.4.3.2 ESPECIFICACIÓN DE CASO DE USO

CU-2	Actualización de conocimiento
Descripción	El administrador actualiza el conocimiento disponible en el sistema, pasando a ser usado en los diagnósticos.
Actores	ADM
Secuencia normal	1. ADM tiene un archivo de conocimiento para actualizar 2. ADM accede a la pantalla de actualización 3. ADM especifica la ruta del archivo a actualizar y da la orden de actualizar 4. El sistema recibe el nuevo conocimiento 5. El sistema analiza dónde se está interesado en actualizar el conocimiento 6. Se envía el conocimiento a los puntos de interés 7. Se recibe el conocimiento, se actualiza la base de conocimiento local y se guarda una copia para cargar en futuros arranques del sistema 8. ADM es notificado del resultado de la operación
Excepciones	

TABLA 8: ESPECIFICACIÓN DE CASO DE USO 2

5.4.3.3 DIAGRAMA DE CASO DE USO

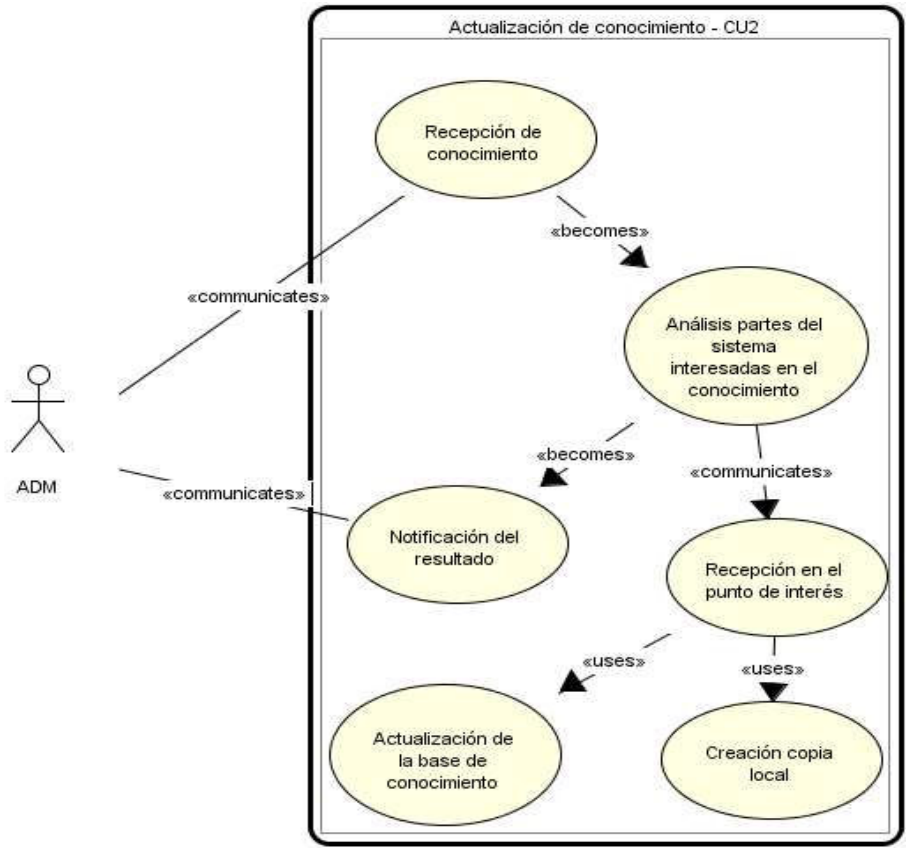

ILUSTRACIÓN 10: DIAGRAMA DE CASO DE USO 2

5.5 REQUISITOS

En esta sección se presentan los requisitos del sistema obtenidos a través de los casos de usos presentados y el escenario global. Todos los requisitos tienen una serie de datos que ofrecen una mayor trazabilidad que permite un seguimiento adecuado de los requisitos.

Cada requisito se presenta en una tabla que contiene:

- Identificador: Identificador único del requisito. Tiene el siguiente formato:
  - RF-n: Requisito funcional número "n"
  - RNF-n: Requisito no funcional número "n"
- Título: Nombre corto del requisito
- Descripción: Especificación de las características que debe cumplir el sistema
- Prioridad: Necesidad de cumplimiento del requisito. Puede tener los siguientes valores:
  - Esencial: Debe cumplirse obligatoriamente para que el sistema tenga el funcionamiento esperado.
  - Alta: Es muy recomendable el cumplimiento de este requisito; no obstante podría no darse un cumplimiento estricto, sino parcial.
  - Media: Requisito opcional que de cumplirse aumentaría la calidad y el uso del sistema, pero no tiene porque ser obligatorio.
  - Baja: Requisito de poca importancia que de cumplirse añadiría pequeñas mejoras al sistema que podrían ser imperceptibles por el usuario.
- Casos de uso relacionados: Aquí se especifican los casos de uso que, tras su análisis detallado, han dado como resultado este requisito.
- Requisitos relacionados: En este apartado se muestran otros requisitos con los que puede estar relacionado éste.

5.5.1 REQUISITOS FUNCIONALES

RF-1	Determinación de la causa de fallo de servicio
Descripción	El sistema debe ser capaz de determinar la causa más probable de fallo de servicio.
Prioridad	Esencial
Casos de uso relacionados	CU-1
Requisitos relacionados	RF-2,RF-3,RF-4

TABLA 9: RF-1 DETERMINACIÓN DE LA CAUSA DE FALLO DE SERVICIO

RF-2	Inicio automático de diagnóstico
Descripción	El sistema debe iniciar un diagnóstico automáticamente al recibir un evento de fallo. Este diagnóstico debe llevar a la determinación más probable de fallo.
Prioridad	Esencial
Casos de uso relacionados	CU-1
Requisitos relacionados	RF-1

TABLA 10: RF-2 INICIO AUTOMÁTICO DE DIAGNÓSTICO

RF-3	Causas posibles
Descripción	Dentro de la causas posibles de fallo, deben estar al menos los siguientes: • Fallo de conectividad en las redes del hogar de los usuarios • Fallo de conectividad en el ISP • Fallo en la gestión de archivos compartidos • Recurso inexistente
Prioridad	Alta
Casos de uso relacionados	CU-1
Requisitos relacionados	RF-1

TABLA 11: RF-3 CAUSAS POSIBLES

RF-4	Informe de diagnóstico
Descripción	El sistema debe enviar un informe después de realizar un diagnóstico. El solicitante del diagnóstico debe recibir un listado con las causas más probables de fallo.
Prioridad	Esencial
Casos de uso relacionados	CU-1
Requisitos relacionados	RF-1

TABLA 12: RF-4 INFORME DE DIAGNÓSTICO

RF-5	Acceso al estado de los diferentes dispositivos
Descripción	El sistema debe consultar el estado de los diferentes dispositivos que intervienen en el escenario para diagnosticar correctamente.
Prioridad	Alta
Casos de uso relacionados	CU-1

Requisitos relacionados	RF-1,RF-2

TABLA 13: RF-5 ACCESO AL ESTADO DE LOS DIFERENTES DISPOSITIVOS

RF-6	Actualización de conocimiento
Descripción	El sistema debe permitir que el administrador del sistema despliegue nuevo conocimiento para futuros diagnósticos.
Prioridad	Esencial
Casos de uso relacionados	CU-2
Requisitos relacionados	RF-1

TABLA 14: RF-6 ACTUALIZACIÓN DE CONOCIMIENTO

RF-7	Despliegue de conocimiento en un solo punto del sistema
Descripción	El administrador debe especificar el nuevo conocimiento en un solo punto del sistema.
Prioridad	Media
Casos de uso relacionados	CU-2
Requisitos relacionados	RF-6,RF-8

TABLA 15: RF-7 DESPLIEGUE DE CONOCIMIENTO EN UN SOLO PUNTO DEL SISTEMA

RF-8	Despliegue de conocimiento transparente
Descripción	El sistema debe desplegar el conocimiento en los lugares apropiados de manera transparente al administrador del sistema.
Prioridad	Media
Casos de uso relacionados	CU-2
Requisitos relacionados	RF-6,RF-8

TABLA 16: RF-8 DESPLIEGUE DE CONOCIMIENTO TRANSPARENTE

5.5.2 REQUISITOS NO FUNCIONALES

RNF-1	Escalabilidad
Descripción	El sistema debe ser lo suficientemente escalable para un escenario real en el que haya múltiples usuarios.
Prioridad	Alta
Casos de uso relacionados	CU-1,CU-2
Requisitos relacionados	RF-1;RF-6

TABLA 17: RNF-1 ESCALABILIDAD

RNF-2	Estabilidad
Descripción	El sistema debe ser lo suficientemente estable para un escenario real en el que debe estar casi siempre operativo.
Prioridad	Alta
Casos de uso relacionados	CU-1
Requisitos relacionados	RF-1

TABLA 18: RNF-2 ESTABILIDAD

RNF-3	Sistema descentralizado
Descripción	El sistema no debe tener un punto central, sino que debe realizar los diagnósticos de forma distribuida.
Prioridad	Alta
Casos de uso relacionados	CU-1
Requisitos relacionados	RF-1, RF-6,RNF-1

TABLA 19: RNF-3 SISTEMA DESCENTRALIZADO

RNF-4	Sistema ligero
Descripción	El sistema debe ser capaz de ejecutar las diferentes partes del proceso de diagnóstico en dispositivos ligeros.
Prioridad	Media
Casos de uso relacionados	CU-1
Requisitos relacionados	RF-1, RNF-1

TABLA 20: RNF-4 SISTEMA LIGERO

RNF-5	Sistema de fácil despliegue
Descripción	El sistema debe ser fácilmente desplegable para poder ser usado en las redes locales de los usuarios.
Prioridad	Alta
Casos de uso relacionados	CU-1,CU-2
Requisitos relacionados	RF-1,RF-6,RNF-1,RNF-4

TABLA 21: RNF-5 SISTEMA DE FÁCIL DESPLIEGUE

RNF-6	Sistema de fácil gestión
Descripción	El sistema debe ser fácil de gestionar al estar ejecutándose en la red de área local del usuario final.
Prioridad	Media
Casos de uso relacionados	CU-1,CU-2
Requisitos relacionados	RNF-1,RNF-4,RNF-5

TABLA 22: RNF-6 SISTEMA DE FÁCIL GESTIÓN

RNF-7	Rendimiento
Descripción	El sistema debe realizar diagnósticos en un tiempo asequible para el usuario, no superior a 30 segundos.
Prioridad	Alta
Casos de uso relacionados	CU-1
Requisitos relacionados	RF-1,RNF-1

TABLA 23: RNF-7 RENDIMIENTO

5.5.3 RESUMEN DE REQUISITOS

A continuación se muestra una tabla resumen sobre los requisitos y algunos datos importantes sobre los mismos.

ID	Prioridad	Caso de uso relacionado
RF-1	Esencial	CU-1
RF-2	Esencial	CU-1
RF-3	Alta	CU-1
RF-4	Esencial	CU-1
RF-5	Alta	CU-1
RF-6	Esencial	CU-2
RF-7	Media	CU-2
RF-8	Media	CU-2
RNF-1	Alta	CU-1,CU-2
RNF-2	Alta	CU-1
RNF-3	Alta	CU-1
RNF-4	Media	CU-1
RNF-5	Alta	CU-1,CU-2
RNF-6	Media	CU-1,CU-2
RNF-7	Alta	CU-1

TABLA 24: RESUMEN DE REQUISITOS DE SISTEMA

6. DISEÑO ARQUITECTÓNICO

6.1 INTRODUCCIÓN

En este capítulo se muestra la fase de diseño del sistema con un modelo de diseño multi-agente, donde los agentes interactúan entre sí para alcanzar un fin común obteniendo diferentes recursos en instantes dados de tiempo.

En las siguientes secciones se presenta el diseño de los agentes donde se especifica la funcionalidad y el comportamiento de cada uno de los tipos de agentes del sistema y las diferentes instancias de agentes de cada tipo que se dan en el caso de estudio.

Para comprender el diseño de los agentes, se debe tener en cuenta la ontología, presentada en la sección 6.3.1, que estos usan en los mensajes que intercambian entre sí para alcanzar su meta final.

El fin de este capítulo es la exposición completa de los agentes que componen el sistema de diagnóstico de fallos. De este modo, se presentan los correspondientes diagramas UML2.0 para facilitar la comprensión del sistema.

6.2 ARQUITECTURA

A continuación se muestra una lista de los diferentes tipos de agentes y un diagrama de secuencia en el que se observan algunas de las comunicaciones que pueden darse entre ellos.

6.2.1 TIPOS DE AGENTES

Además de los agentes que se exponen a continuación, dependiendo del escenario de aplicación y otros factores puede ser necesaria la existencia de otros agentes adicionales.

Tipos de agentes	Descripción
Agente de interfaz	Su función es recibir las detecciones de fallos y, tras un diagnóstico completo, devolver un informe al destinatario adecuado. Es la interfaz externa más importante del sistema.
Agente de diagnóstico	Es el director del diagnóstico. Recibe solicitudes desde agentes de interfaz y siguen un algoritmo iterativo en el que van añadiendo nuevas evidencias o creencias a través de otros agentes especializados hasta alcanzar una cierta confianza en el diagnóstico. Este tipo de agente realiza inferencia Bayesiana y es el punto de toma de decisiones más importante del sistema.
Agente especialista	Este tipo de agente permite, a un agente de diagnóstico, obtener información necesaria para el diagnóstico sobre una observación o una creencia concreta.
Agente de conocimiento	Este tipo de agente es el encargado de la distribución del conocimiento adecuado a cada agente interesado en él. Ofrece una interfaz externa al administrador del sistema que le permite desplegar conocimiento al sistema.

TABLA 25: TIPOS DE AGENTES

El tipo de agente especialista se desdobla a su vez en dos subtipos de agentes los cuales ofrecen diferentes posibilidades a un agente de diagnóstico. En la Tabla 26: Subtipos de agentes especialistas se muestran los dos subtipos de agentes especialistas junto a una breve descripción de ellos.

Subtipos de agentes especialistas	Descripción
Agente de observación	Es un tipo de agente especializado en un recurso concreto de la red o el servicio. Su función es ofrecer evidencias a los agentes que las soliciten realizando pruebas, analizando alarmas, etc. Este tipo de agente tiene interfaces con los dispositivos, las herramientas o los servicios con los que tiene relación.
Agente de creencia	A diferencia de los agentes de observación, no ofrece evidencias o hechos; sino creencias sobre un hecho concreto. Esto permite delegar partes más complejas del diagnóstico a otros agentes y distribuir el conocimiento y la inferencia Bayesiana en diferentes puntos de interés en el sistema. A su vez, estos agentes pueden hacer uso de otros agentes de observación o de creencia.

TABLA 26: SUBTIPOS DE AGENTES ESPECIALISTAS

Tras esta breve descripción de los diferentes tipos de agentes, se muestra en la Ilustración 11 un diagrama de secuencia en el que se puede observar algunos de los diferentes tipos de mensajes que se intercambian los agentes durante un diagnóstico. No obstante, en los apartados de diseño de cada tipo de agente, se muestran diagramas más concretos.

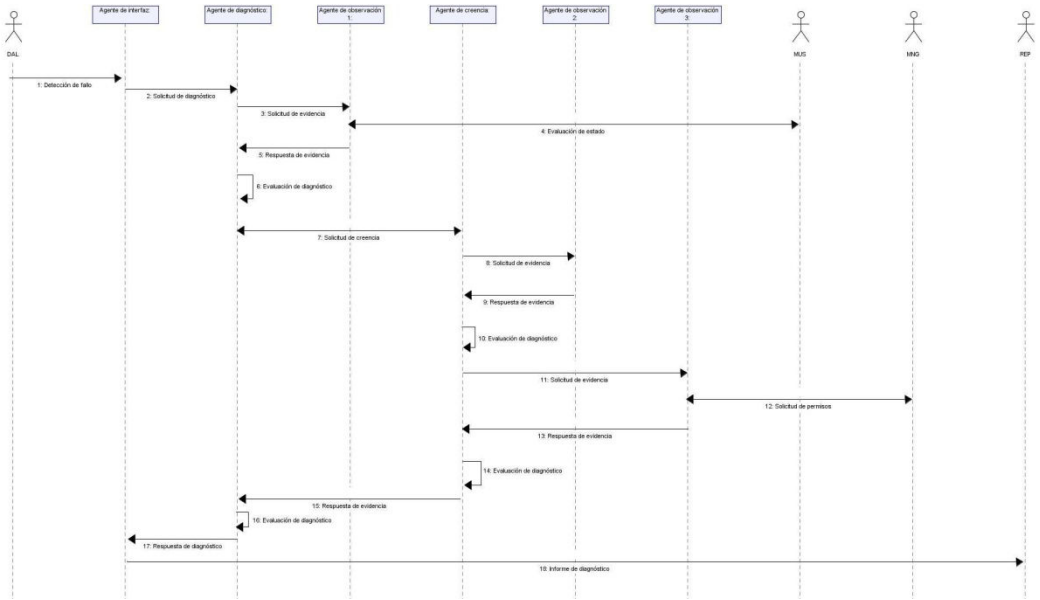

ILUSTRACIÓN 11: DIAGRAMA GENERAL DE SECUENCIA

Los mensajes intercambiados entre los diferentes agentes usan el estándar FIPA-ACL conjuntamente con una ontología específica para el diagnóstico con redes bayesianas.

Dicha ontología está descrita en la sección 6.3.1, donde se exponen tanto el dominio de conocimiento sobre redes bayesianas, las diferentes operaciones que pueden ejecutar los agentes y el modelado de una operación de diagnóstico con sus parámetros y resultados.

6.2.2 DISTRIBUCIÓN DE AGENTES

Los tipos de agentes de la Tabla 25 pueden distribuirse a través de un escenario real de múltiples maneras dependiendo de las necesidades de cada escenario y del diagnóstico a realizar. No obstante, en la Ilustración 12 se presenta una posible distribución de los diferentes agentes dentro de una red.

Como puede observarse, los agentes podrían desplegarse en cualquier dispositivo que soportase JAVA con unos bajos requisitos de capacidad, sin necesidad de ninguna otra homogeneidad entre los dispositivos donde se ejecutarían los diferentes agentes.

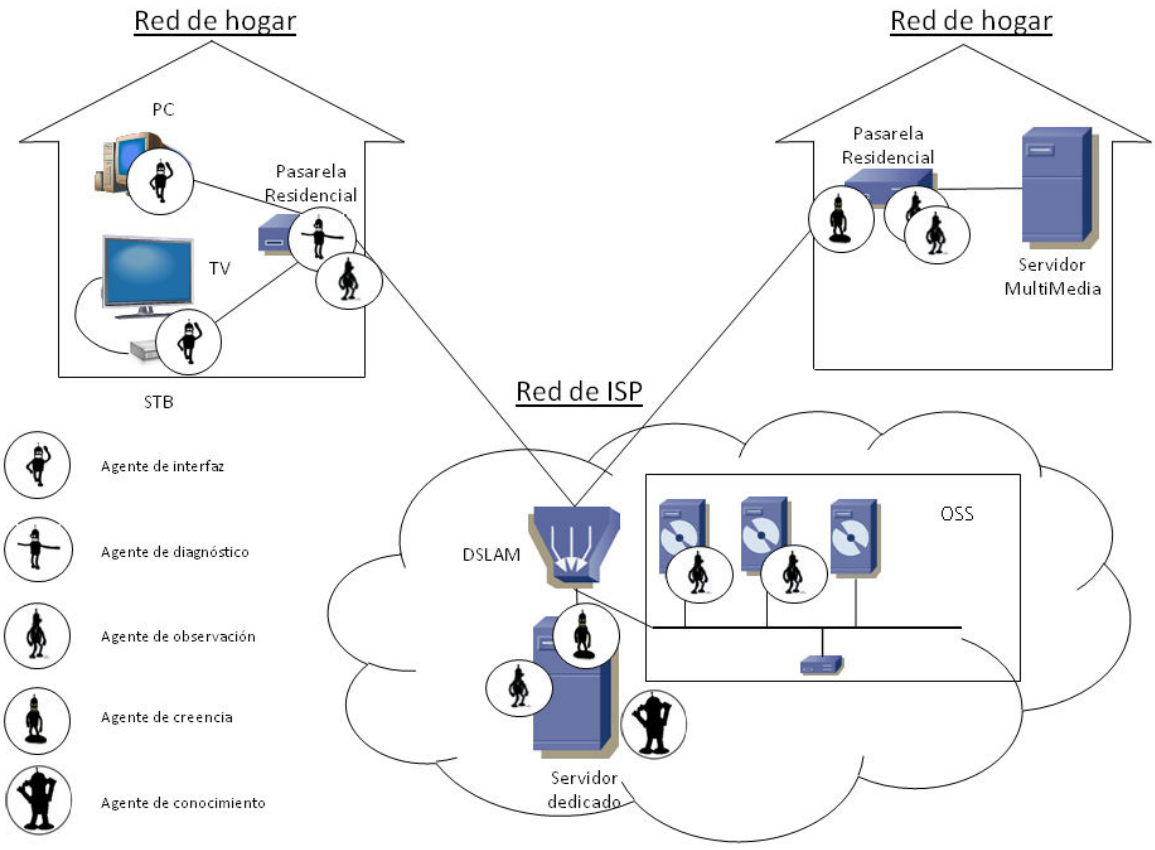

ILUSTRACIÓN 12: DISTRIBUCIÓN DE LOS AGENTES

6.3 DISEÑO DETALLADO

En este apartado se expone la ontología usada en el sistema y el diseño de los agentes. Siguiendo este orden concreto, se pretende presentar de una manera clara, en primer lugar, el dominio de conocimiento en el que los agentes interactúan y, en segundo, el modo en el que dichos agentes interactúan para alcanzar un objetivo común.

6.3.1 ONTOLOGÍA DEL SISTEMA

La ontología utilizada en el sistema modela el conocimiento usado en un sistema de diagnóstico englobando los diferentes parámetros que definen una red bayesiana y las diferentes acciones que se pueden tomar sobre esta a través de unas operaciones dadas.

Para el diseño de ésta, se ha usado la herramienta *Protégé* (ver sección 4.2.1.7) conjuntamente con el *plug-in Ontology Bean Generator* (ver sección 4.2.3) para generar los *beans* que representan a esta ontología en el dominio de clases JAVA que usan los agentes JADE, ya que dicho *plug-in* está completamente integrado con este *framework* de agentes.

Durante el diseño de la ontología se presenta un punto de decisión evidente en el que elegir entre los diferentes lenguajes para representar un conocimiento dado.

La elección de usar este formato de clases JAVA en lugar de otros, como podría ser OWL con su mayor expresividad e capacidad de inferencia entre sentencias, está dirigida por la búsqueda de los requisitos de rendimiento y escalabilidad (ver sección 5.5.2); ya que el trasiego de información a través de diferentes librerías de conversión OWL-JAVA y viceversa ralentizaría el proceso de diagnóstico.

En las siguientes secciones se presenta el conjunto de clases JAVA que modelan la ontología diferenciando tres campos:

- El modelado del conocimiento sobre la red bayesiana
- El modelado de una operación de diagnóstico
- Y el modelado de las diferentes acciones que pueden llevar a cabo los agentes.

Para mostrar estas áreas de conocimiento, dentro de dichas secciones, se usan un diagrama UML2.0 de clases JAVA y una tabla explicativa sobre el contenido del diagrama.

6.3.1.1 *CONOCIMIENTO BAYESIANO*

En la Ilustración 13 se muestra el diagrama que modela el conocimiento sobre la red bayesiana que maneja el sistema.

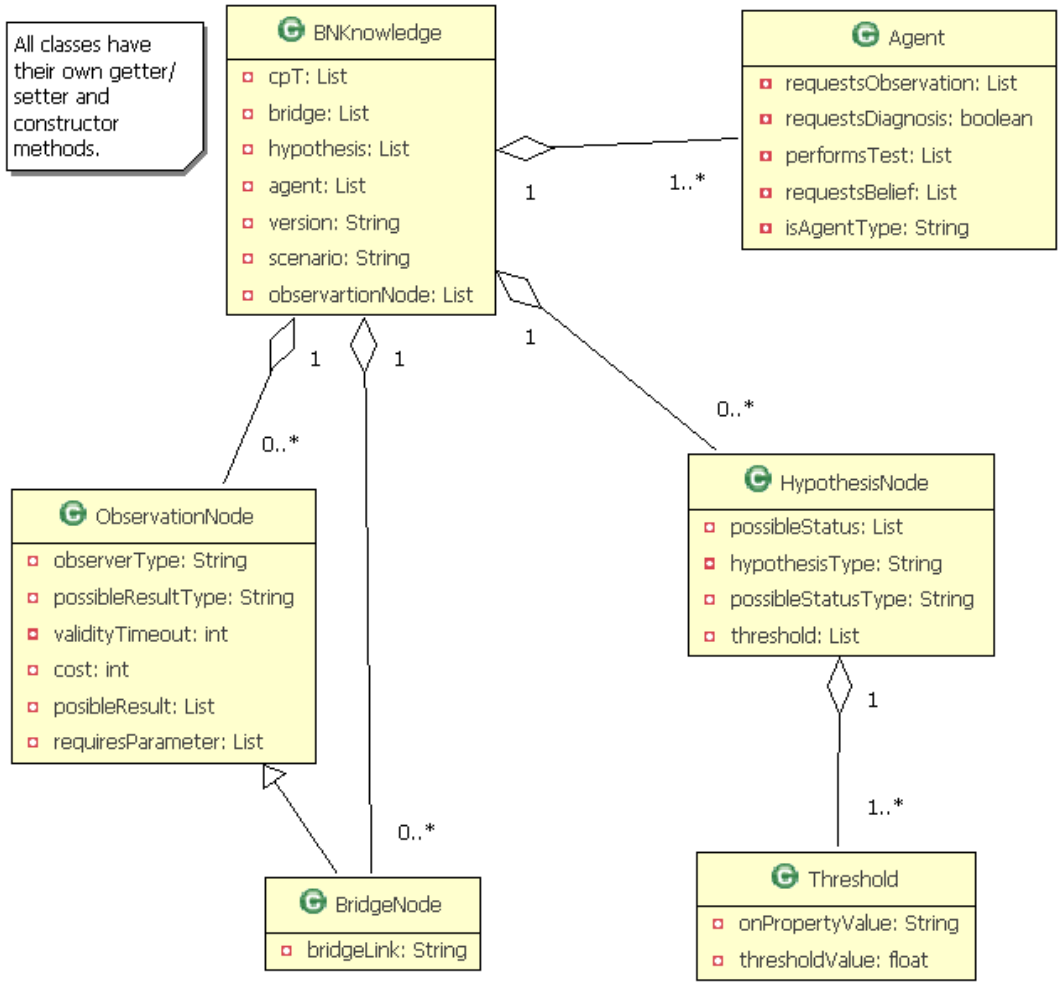

ILUSTRACIÓN 13: CONOCIMIENTO BAYESIANO

Cada una de las clases anteriores modela el siguiente conocimiento:

Clase	Significado
BNKnowledge	Representa el elemento de la red bayesiana al completo, con todas las variables de interés dentro de ésta.
HypothesisNode	Representa un nodo hipótesis dentro de una red bayesiana. Este puede tener diferentes estados y, cada uno de ellos, su propio umbral.
Threshold	Representa un umbral para presentar una

	hipótesis como diagnóstico válido.
ObservationNode	Representa un nodo de observación dentro de una red bayesiana. Estos nodos representan a su vez pruebas que necesitan una serie de parámetros para ser realizadas.
BridgeNode	Representa un nodo auxiliar necesario para realizar un intercambio de creencias entre diferentes redes bayesianas.
Agent	Representa el agente en el que reside la red bayesiana y, con él, las diferentes acciones que este soporta.

TABLA 27: CONOCIMIENTO BAYESIANO

6.3.1.2 OPERACIÓN DE DIAGNÓSTICO

En la Ilustración 14 se modela el conocimiento sobre una operación de diagnóstico del sistema.

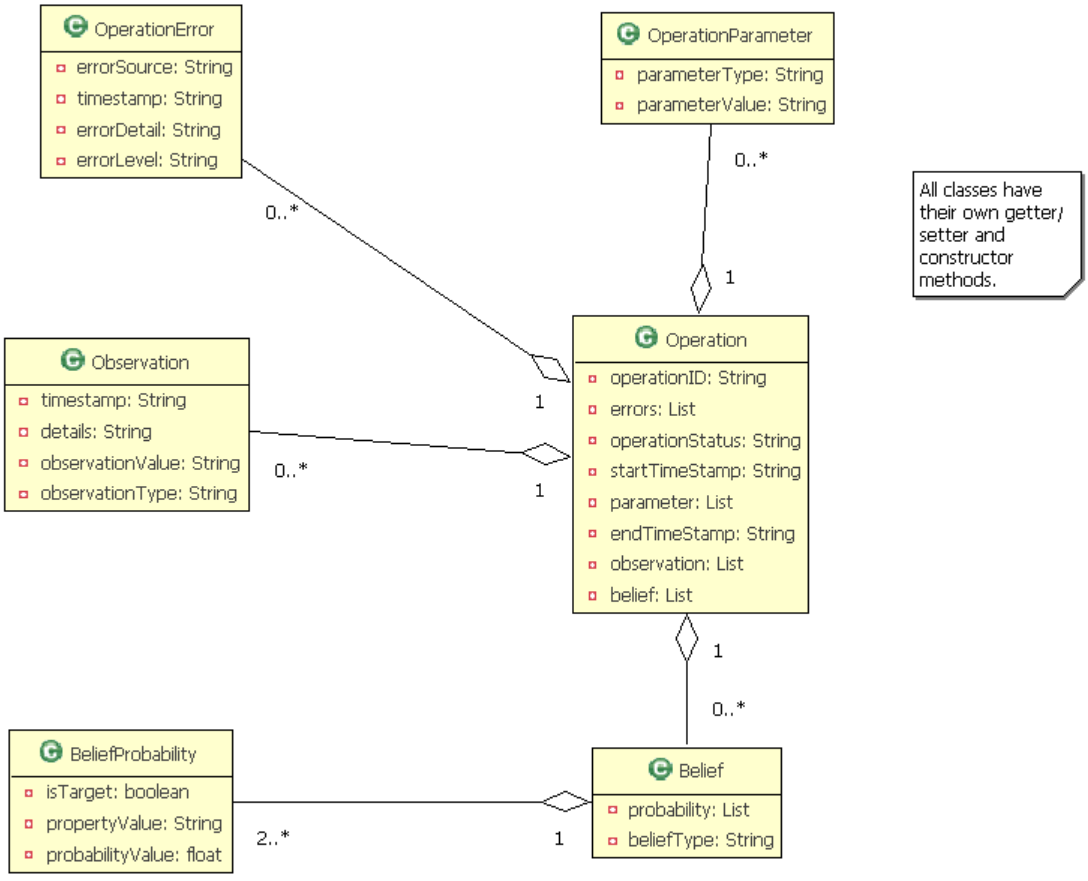

ILUSTRACIÓN 14: OPERACIÓN DE DIAGNÓSTICO

Cada una de las clases modela el siguiente conocimiento:

Clase	Significado
Operation	Representa una operación de diagnóstico con sus creencias, observaciones realizadas, parámetros, etc.
OperationParameter	Representa un parámetro que servirá para realizar diferentes pruebas dentro de la operación de diagnóstico.
OperationError	Representa un error dentro de la operación de diagnóstico.
Observation	Representa una observación realizada dentro de la operación que servirá como evidencia para la red bayesiana.
Belief	Representa una creencia concreta sobre una hipótesis dentro del diagnóstico.
BeliefProbability	Representa una probabilidad sobre una creencia y además modela si es una de las creencias que se buscan dentro del diagnóstico.

TABLA 28: OPERACIÓN DE DIAGNÓSTICO

6.3.1.3 ACCIONES DE AGENTES

Para facilitar la comprensión de las diferentes acciones que los agentes pueden llevar a cabo, se presentan dos tipos de acciones:

- Las que los agentes pueden hacer durante una operación de diagnóstico
- Y las que pueden hacer fuera de una de estas operaciones.

Las primeras de ellas se modelan en la Ilustración 15.

Cada una de estas clases modela las siguientes acciones:

Acción	Descripción
RequestDiagnosis	Solicitud de diagnóstico. Todo agente que quiera iniciar un diagnóstico puede ejecutar esta acción para que el agente apropiado tome el resto de decisiones por él.
InformDiagnosis	Informe de diagnóstico. Cuando un agente llega al final del diagnóstico, devuelve el resultado al agente que lo solicitó.
RequestObservation	Solicitud de observación. Esta acción la puede solicitar un agente que quiera más información para alcanzar un diagnóstico válido.
InformObservation	Informe de observación. Esta acción

	permite informa del resultado de una observación a él solicitante.
RequestBelief	Solicitud de creencia. Si un agente quiere obtener una creencia concreta sin tener que conocer los detalles de cómo conseguirlo, tiene la posibilidad de solicitar la creencia de un nodo en concreto.
InformBelief	Informe de creencia. Una vez alcanzado un estado estable, el agente puede servir la información solicitada con esta acción.

TABLA 29: OPERACIONES DE AGENTES – PARTE 1

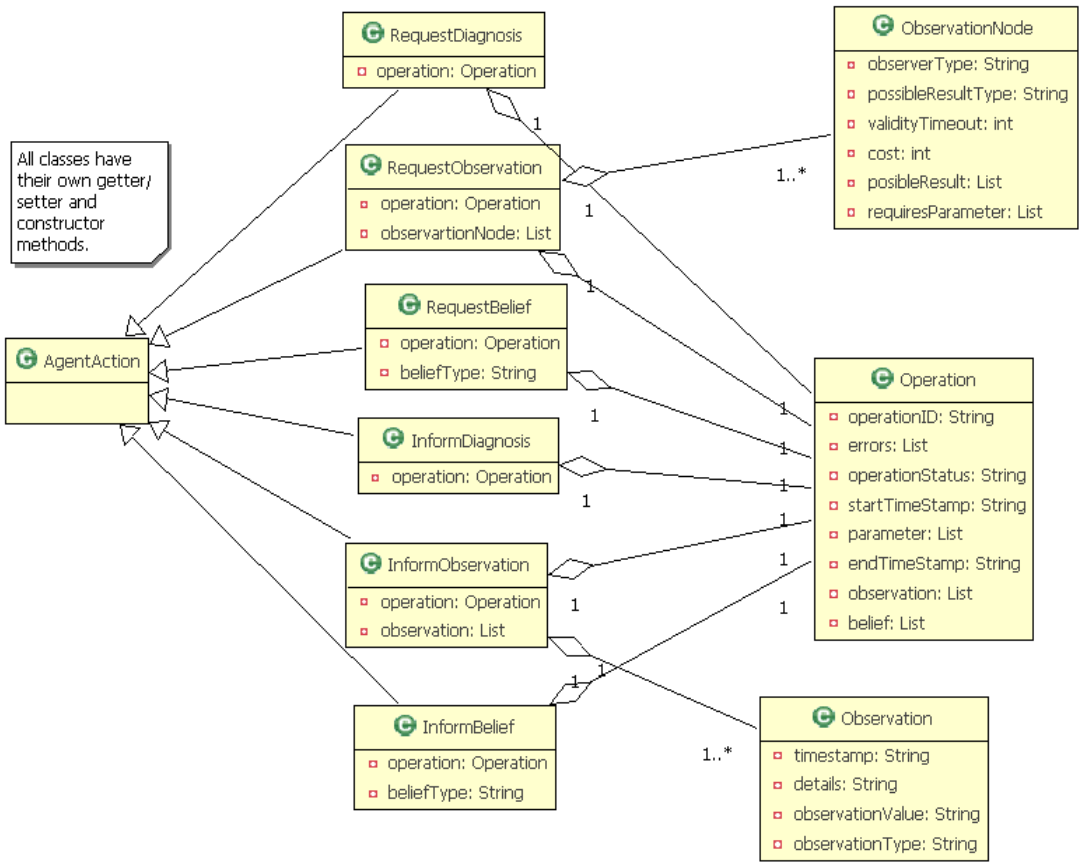

ILUSTRACIÓN 15: OPERACIONES DE AGENTES – PARTE 1

En la Ilustración 16 se modelan las operaciones que los agentes pueden realizar fuera de un diagnóstico.

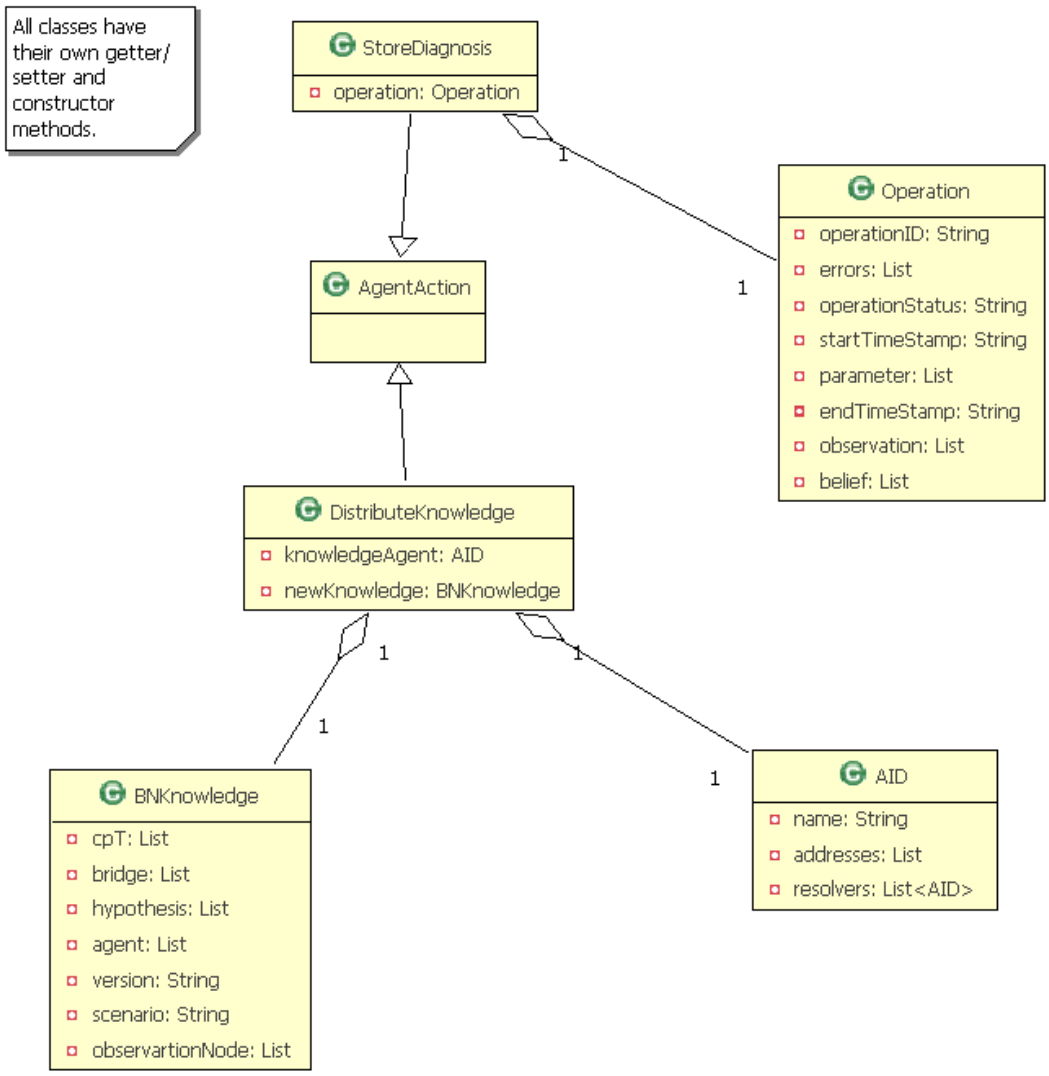

ILUSTRACIÓN 16: OPERACIONES DE AGENTES – PARTE 2

Cada una de estas clases modela las siguientes acciones:

Acción	Descripción
StoreDiagnosis	Almacenar diagnóstico. Esta acción permite a un agente hacer una solicitud de almacenamiento de diagnóstico de manera persistente.
DistributeKnowledge	Distribuir conocimiento. Esta acción permite a un agente de conocimiento enviar una actualización de conocimiento a los interesado se un escenario concreto.

TABLA 30: OPERACIONES DE AGENTES – PARTE 2

Gracias a estas operaciones, el sistema puede adaptarse con facilidad a un módulo de aprendizaje con el cual mejorar los diagnósticos. En primer lugar, la operación "*StoreDiganosis*" permite guardar el resultado de un diagnóstico en una base de datos

de manera persistente para que el módulo de aprendizaje pueda consultar históricos de datos. Y, en segundo lugar, la operación de "*DistributeKnowledge*" permite distribuir el nuevo conocimiento a los agentes interesados en él. Esta operación también permite cumplir una característica esencial en el sistema para la mayoría de los requisitos obtenidos del Caso de uso 2: Actualización de conocimiento.

6.3.1.4 USO DE LA ONTOLOGÍA POR EL SISTEMA

Esta ontología es usada por los agentes mediante un conjunto de interfaces que facilitan la abstracción del nivel de detalle que presenta ésta. Al igual que la ontología ha sido definida en tres bloques diferenciados, en esta sección se presentan las herramientas que interactúan con cada uno de los bloques.

La primera de estas es el gestor de conocimiento bayesiano "*KnowledgeManager*". Este gestor es usado tanto para facilitar el modo de encontrar las mejores acciones posibles dentro de un diagnóstico como para recargar un nuevo conocimiento que el agente pueda recibir en un momento dado.

Para el proceso de diagnóstico, el concepto de operación es ampliamente utilizado tanto para consultar los parámetros disponibles o para comprobar si alguna creencia supera su umbral y se puede dar por concluido un diagnóstico. Dada la importancia de este concepto, cada agente tiene un almacén de operaciones "*OperationStore*" en el que se pueden acceder y actualizar los datos de la operación según sea necesario.

El último bloque de la ontología, que trata sobre las posibles acciones que pueden llevar a cabo los agentes, dispone de una librería de mensajería "Messenger" que permite traducir de manera transparente las instancias de la ontología a mensajes FIPA-ACL y viceversa. De este modo, los agentes manejan información en un formato más adecuado para su procesamiento.

Estas herramientas se engloban dentro del componente "*knowledge*" en los diagramas de componentes de los agentes presentados en las siguientes secciones.

6.3.2 DISEÑO DE AGENTES

En esta sección se presenta el modelado de los agentes del sistema. Cada agente se presenta con:

- un modelo BDI,
- un diagrama de actividad UML2.0
- un diagrama de secuencia UML2.0 y
- un diagrama de componentes UML2.0.

6.3.2.1 DISEÑO DE AGENTE DE INTERFAZ

Este agente provee al sistema de una ventana de interacción con el sistema de monitorización (actor DAL en el Caso de uso 1: Diagnóstico de un síntoma, ver Tabla 6: Diccionario de actores) y con el notificador de informes de diagnóstico (actor REP en el Caso de uso 1: Diagnóstico de un síntoma, ver Tabla 6: Diccionario de actores).

En la Tabla 31 se muestra el modelo BDI de este agente. Según este modelo, el agente es consciente de que la existencia de los dos actores (sistemas externos) que interactúan con él, del evento de entrada y los datos que conlleva y, por último, de la existencia de un agente que ofrece un servicio de diagnóstico que puede ser solicitada.

Para alcanzar su objetivo, dispone de dos planes que ejecutará en el momento adecuado. Primero, solicitará el diagnóstico a un agente competente para ello y, cuando obtenga un resultado, notificará al sistema apropiado.

Modelo BDI	Agente de interfaz
Creencias	• Evento de entrada • Agente de diagnóstico • DAL • REP
Objetivos	• Informar de resultado de diagnóstico
Planes	• Solicitar diagnóstico • Enviar evento de informe

TABLA 31: MODELO BDI - AGENTE DE INTERFAZ

El diagrama de actividad de este agente es el mostrado en la Ilustración 17. En este diagrama se representan las diferentes actividades que realiza el agente para alcanzar su objetivo.

- En el estado inicial el agente escucha eventos de detección de fallos
- Una vez recibido un evento, el agente decide realizar una solicitud de diagnóstico a un agente apto para ello.
- Tras esto, espera un informe de diagnóstico.
- Una vez recibe un informe del agente de diagnóstico al finalizar este la operación, éste se transforma a un modelo apto para el sistema al que se le debe notificar las conclusiones alcanzadas.
- Para alcanzar su objetivo final, el agente envía un evento de notificación al sistema apropiado y vuelve al estado inicial.

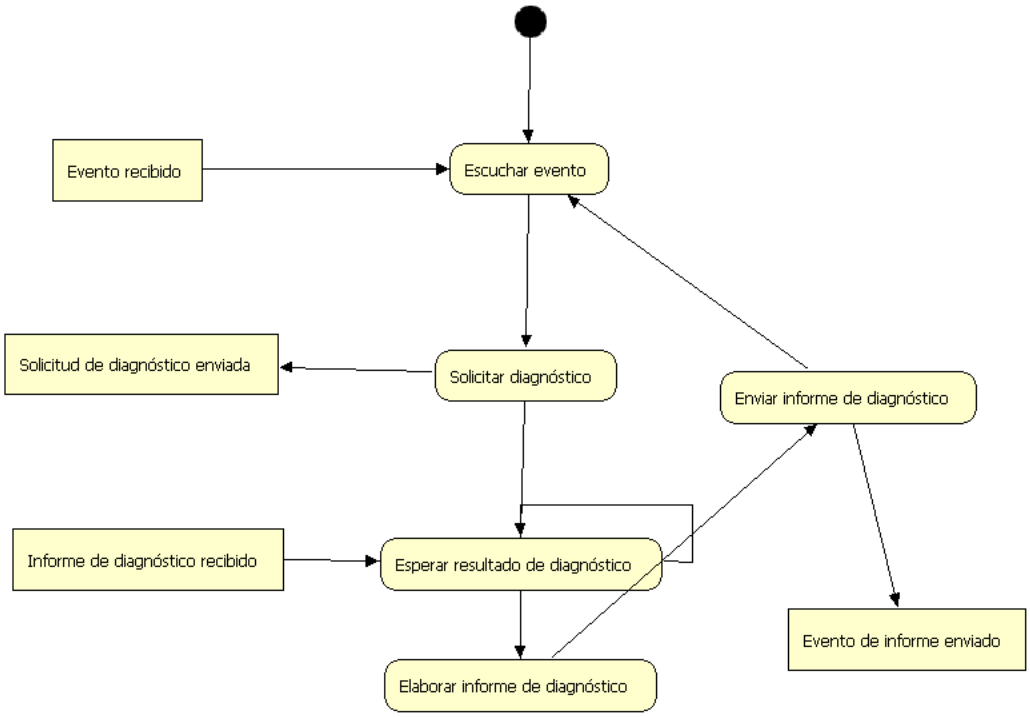

ILUSTRACIÓN 17: DIAGRAMA DE ACTIVIDAD - AGENTE DE INTERFAZ

Para comprender mejor el intercambio de mensajes del agente con otros agentes o sistemas, se adjunta el diagrama: Diagrama de secuencia, Ilustración 18.

Hay que anotar que las interacciones con los actores se dan a través de eventos OSGi y con el agente de diagnóstico, a través de mensajes FIPA-ACL.

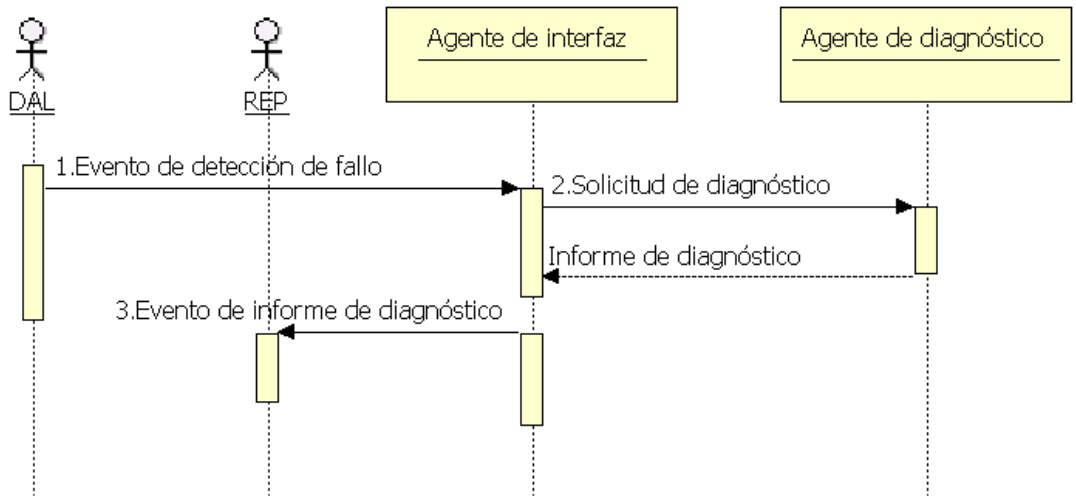

ILUSTRACIÓN 18: DIAGRAMA DE SECUENCIA - AGENTE DE INTERFAZ

Profundizando más en el diseño del agente, se muestra en la Ilustración 19 un diagrama de componentes donde se exponen los módulos usados por este agente.

En este diagrama se observa que usa tanto el paquete de manejo de los eventos de la plataforma OSGi ("*osgiEventIF*") como el paquete de manejo de mensajes e instancias de la ontología del sistema ("*knowledge*"); para así, como poder actuar como interfaz entre los dos dominios.

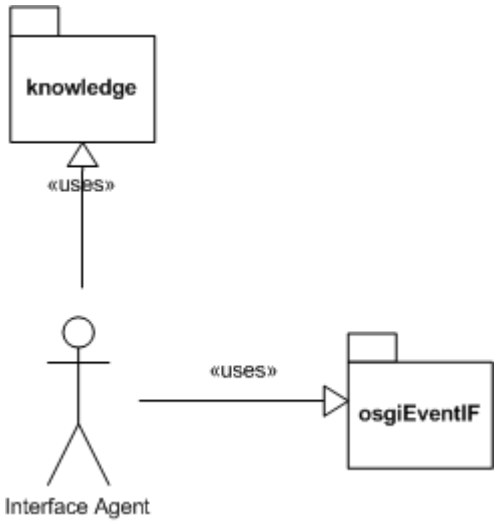

ILUSTRACIÓN 19: DIAGRAMA DE COMPONENTES - AGENTE DE INTERFAZ

6.3.2.2 DISEÑO DE AGENTE DE DIAGNÓSTICO

El agente de diagnóstico es el encargado principal de un diagnóstico. Dispone de una red bayesiana en la que se modelan las diferentes causas que puede llegar a diagnosticar. No obstante, con la ayuda de diferentes agentes de creencia, puede diagnosticar más hipótesis que las modeladas en su propia red bayesiana.

Comienza recibiendo un evento de detección de fallo a través del agente de interfaz e inicia un diagnóstico con dicha información. Una vez iniciado el proceso de diagnóstico, busca las mejores acciones que puede ejecutar en un momento dado y las realiza. Cuando termina un diagnóstico, ya sea por superar el umbral de alguna hipótesis o por haber realizado todas las acciones posibles para alcanzar una conclusión válida, el agente de diagnóstico envía un informe al agente que solicitó la operación.

En la Tabla 32 se muestra el modelo BDI de este agente. El agente de diagnóstico es consciente de los agentes con los que interactúa, como son el agente de interfaz de quien recibe el evento de detección de fallo, o los agentes de creencia u observación cuyos servicios pueden ser solicitados por el agente de diagnóstico. Además de esto, también tiene conocimiento de las diferentes hipótesis que se manejan durante el diagnóstico y del conjunto de acciones que pueden realizarse para alcanzar una conclusión válida.

Tiene dos objetivos principales, atender solicitudes de diagnóstico que puedan recibirse y realizar dichos diagnósticos. Para ello, tiene una serie de acciones que puede realizar para obtener más información y presentar unas hipótesis más acertadas. Una vez terminado el proceso de diagnóstico, el agente alcanza su primer objetivo enviando un informe de diagnóstico al agente solicitante.

Modelo BDI	Agente de diagnóstico
Creencias	• Agente de interfaz • Agentes de creencia • Agentes de observación • Conjunto de hipótesis • Conjunto de posibles acciones
Objetivos	• Atender solicitudes de diagnóstico • Alcanzar diagnóstico fiable
Planes	• Solicitar creencia • Solicitar observación • Realizar observación • Enviar informe de diagnóstico

TABLA 32: MODELO BDI - AGENTE DE DIAGNÓSTICO

En la Ilustración 20 se muestra el diagrama de actividad que tiene un agente de diagnóstico.

- En el estado inicial, comienza escuchando los mensajes de solicitud de diagnóstico de un agente de interfaz.
- Una vez recibido un mensaje de solicitud de diagnóstico, obtiene los parámetros que contiene el mensaje para poder decidir qué acción tomar dada la información en ese instante.
- Se inicia el proceso de diagnóstico.
- Se decide por la mejor acción para tomar en ese momento que dependerá tanto del coste como de la región en la que se quiere actuar.
- Si decide solicitar una observación o una creencia a un agente remoto, envía la solicitud correspondiente y espera la respuesta.
- Si decide realizar una prueba local, la ejecuta y guarda los datos obtenidos dentro de la operación.
- Una vez realizada la acción y obtenida nueva información, se actualiza la base de creencias.
- Se realiza inferencia bayesiana para intentar alcanzar un diagnóstico fiable.
- Con las hipótesis actualizadas gracias a la inferencia, se evalúa el estado actual del diagnóstico.
- Si aún no se ha superado el umbral de ninguna hipótesis, se vuelve a buscar la mejor acción que aún esté pendiente de realizar.

- Si se ha superado un umbral o no quedan más acciones que realizar, se finaliza el diagnóstico.
- Una vez finalizado, se genera un mensaje de informe de diagnóstico, que tiene como destinatario el agente de interfaz que solicitó el diagnóstico.
- Tras enviar el mensaje, se vuelve al estado inicial en el que espera nuevas solicitudes de diagnóstico.

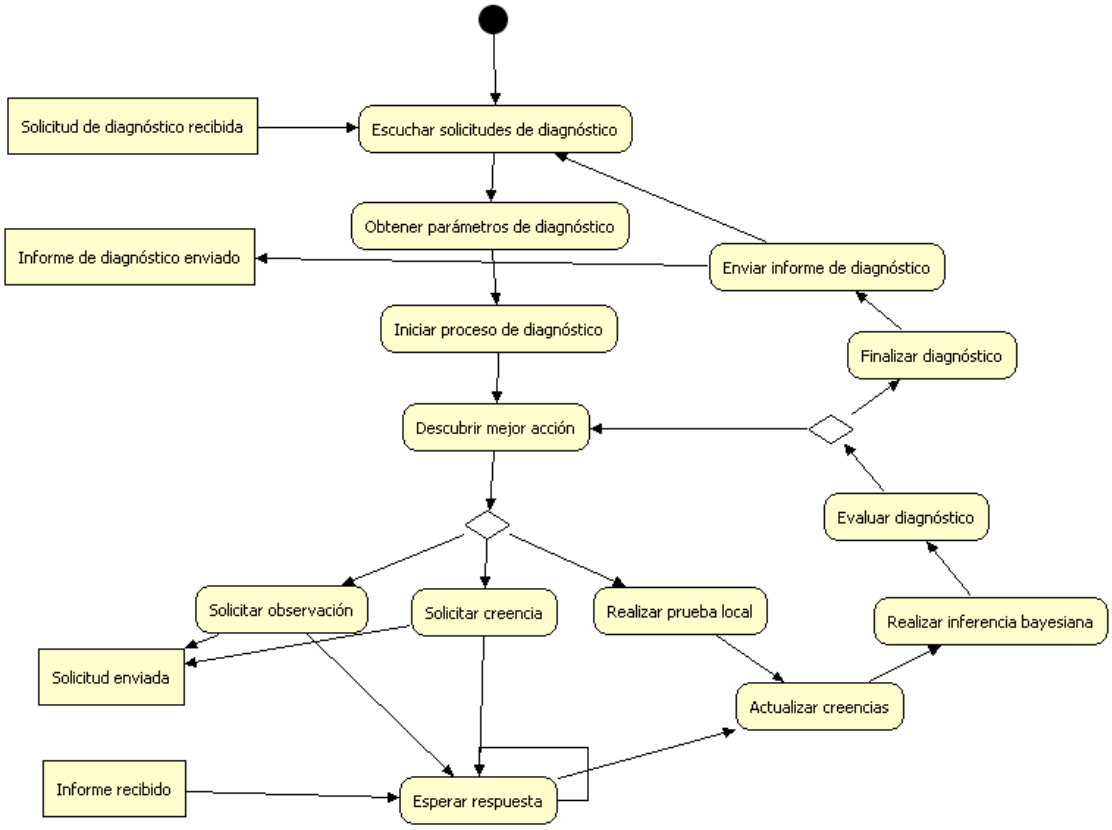

ILUSTRACIÓN 20: DIAGRAMA DE ACTIVIDAD - AGENTE DE DIAGNÓSTICO

El diagrama de secuencia que representa las interacciones de un agente de diagnóstico con su entorno está representado en la Ilustración 21.

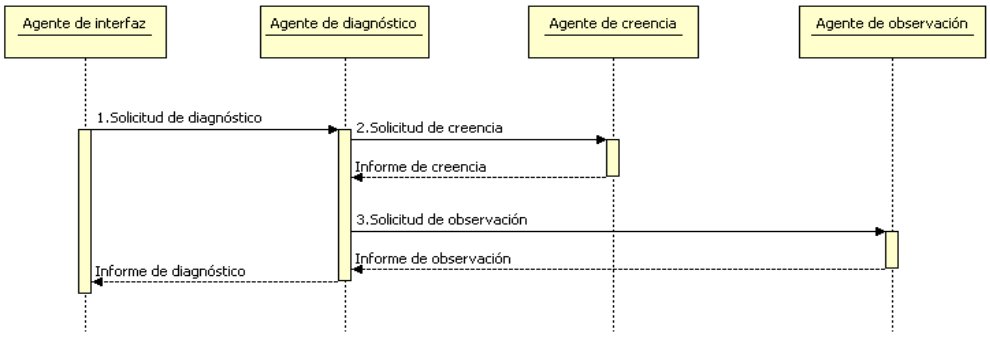

ILUSTRACIÓN 21: DIAGRAMA DE SECUENCIA - AGENTE DE DIAGNÓSTICO

El agente de diagnóstico interactúa con el resto de agente mediante mensajes FIPA-ACL en los cuales viaja información asociada al diagnóstico.

Profundizando más en el diseño del agente, se muestra en la Ilustración 22 un diagrama de componentes donde se exponen los módulos usados por este agente.

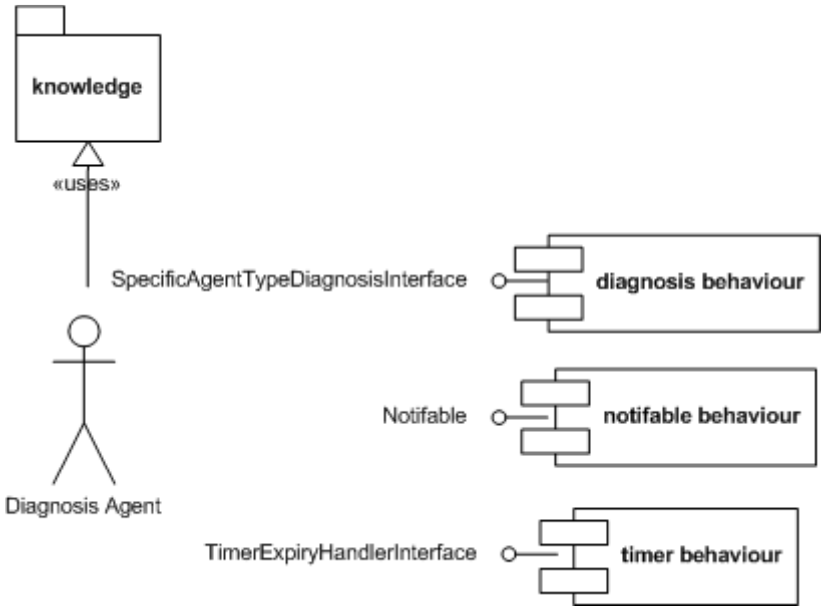

ILUSTRACIÓN 22: DIAGRAMA DE COMPONENTES - AGENTE DE DIAGNÓSTICO

En este diagrama se observa que el agente usa el recurso "*knowledge*" para interactuar con las diferentes instancias de la ontología, las operaciones o los mensajes intercambiados. Además implementa diferentes interfaces para poder usar los componentes que le permiten tener diferentes comportamientos:

- "*diagnosis behaviour*" contiene un proceso de diagnóstico genérico para el sistema,
- "*notifable behaviour*" contiene un comportamiento genérico para recibir notificaciones sobre, por ejemplo, actualizaciones de conocimiento,
- "*timer behaviour*" contiene un comportamiento cíclico que libera memoria del dispositivo en el que se ejecute el agente borrando diagnósticos pasados. Este comportamiento permite una posible vía para la implementación de futuras características en el sistema (ver sección 10.3).

6.3.2.3 DISEÑO DE AGENTE ESPECIALISTA: AGENTE DE OBSERVACIÓN

El agente de observación es un tipo de agente que tiene un conocimiento enfocado a un conocimiento concreto y por lo tanto está especializado en éste. Comienza escuchando solicitudes de observación de diferentes agentes (ya sean agentes de diagnóstico o agentes de creencia). Al recibir dicha solicitud, realiza una prueba local y devuelve un resultado (una observación) al agente solicitante para que este actualice

su base de conocimiento y así sus creencias sobre las diferentes hipótesis a diagnosticar.

El modelo BDI del agente de observación se muestra en la Tabla 33. Un agente de observación no conoce qué agentes pueden solicitarle sus servicios, sino que es consciente de qué agente en concreto hace la solicitud para poder responder adecuadamente a ésta. También es consciente de los diferentes parámetros que necesita para realizar las observaciones solicitadas correctamente. Para alcanzar su objetivo de atender solicitudes, tiene un primer plan que consiste en realizar una prueba local y un segundo plan de enviar el informe al agente adecuado.

Modelo BDI	Agente de observación
Creencias	• Agente solicitante • Parámetros de prueba
Objetivos	• Atender solicitudes de observación
Planes	• Realizar prueba local • Enviar informe de observación

TABLA 33: MODELO BDI - AGENTE DE OBSERVACIÓN

Para exponer más claramente el proceso que sigue este tipo de agente, en la Ilustración 23 se presenta un diagrama de actividad que lo representa.

- En el estado inicial, comienza escuchando las solicitudes de diagnóstico que puede recibir.
- Tras recibir una solicitud, el agente obtiene los parámetros necesarios para realizar su prueba local.
- Realiza la prueba.
- Y envía un informe de observación al agente que la solicitó.

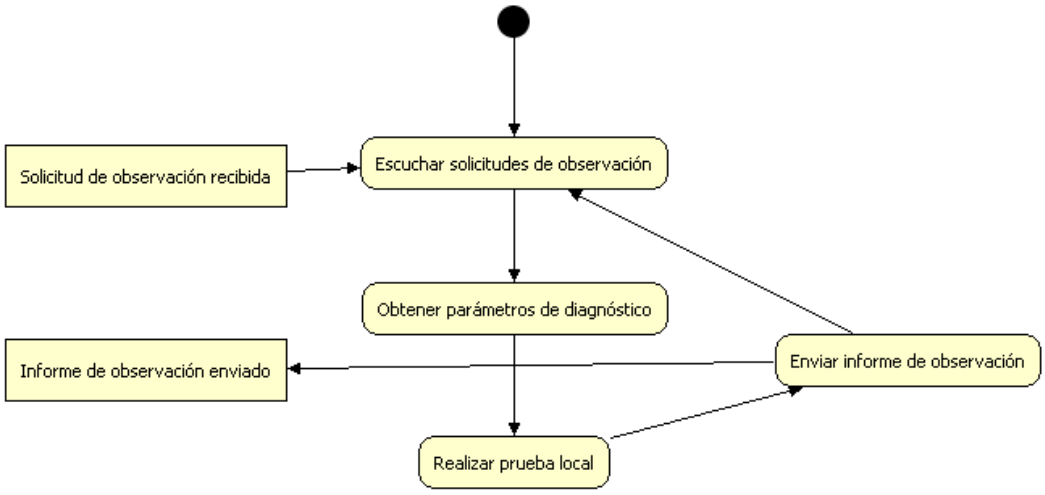

ILUSTRACIÓN 23: DIAGRAMA DE ACTIVIDAD - AGENTE DE OBSERVACIÓN

Un agente de observación podrá intercambiar, mediante mensajes FIPA-ACL, solicitudes e informes de observación con dos tipos diferentes de agentes:

- agentes de diagnóstico y
- agentes de creencia;

Tal y como se muestra en la Ilustración 24.

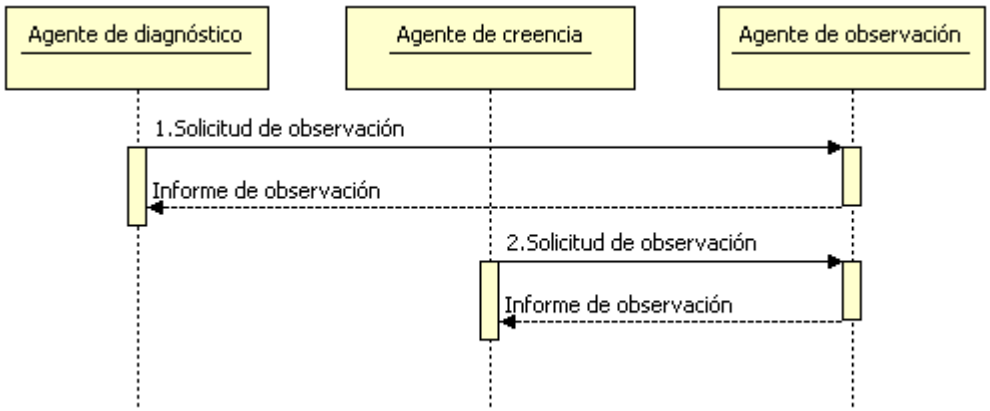

ILUSTRACIÓN 24: DIAGRAMA DE SECUENCIA - AGENTE DE OBSERVACIÓN

Profundizando más en el diseño del agente, se muestra en la Ilustración 25 un diagrama de componentes donde se exponen los módulos usados por este agente.

En este diagrama se observa como el agente sólo usa el recurso "*knowledge*" para interactuar con el resto de agentes del sistema a través de mensajes, ya que no realiza inferencia alguna y no necesita ningún comportamiento más que el comportamiento de ejecución de pruebas. No obstante, este comportamiento requiere una implementación concreta de la prueba a realizar por el agente en cuestión.

ILUSTRACIÓN 25: DIAGRAMA DE COMPONENTES - AGENTE DE OBSERVACIÓN

6.3.2.4 Diseño de agente especialista: Agente de creencia

El agente de creencia dispone, al igual que el agente de diagnóstico, de una red bayesiana sobre la cual realizar inferencia. Si las redes bayesianas son modeladas correctamente, un agente de creencia puede especializarse en regiones de

conocimiento concretas (ya sea en un servicio concreto, una región física concreta o demás posibilidades).

Gracias a este método, el sistema cuenta con una flexibilidad añadida, ya que varios agentes de diagnóstico pueden solicitar las creencias necesarias a un solo agente de creencia que esté especializado en un dominio concreto de conocimiento experto, disminuyendo así considerablemente el número de mensajes intercambiados entre diferentes dominios.

Comienza recibiendo una solicitud de creencia y, al igual que el agente de diagnóstico, inicia un proceso de diagnóstico con dicha información. Una vez iniciado el proceso de diagnóstico, busca las mejores acciones que puede ejecutar en un momento dado y las realiza. Cuando termina un diagnóstico, ya sea por superar el umbral de alguna hipótesis o por haber realizado todas las acciones posibles para alcanzar una conclusión válida, el agente de diagnóstico envía un informe del resultado al agente solicitante.

En la Tabla 34 se muestra el modelo BDI de este agente. El agente es consciente de los agentes con los que interactúa, como el agente solicitante o los agentes de creencia u observación cuyos servicios pueden ser solicitados por el agente de diagnóstico, y de las diferentes hipótesis y acciones con las que interactúa durante la operación.

Tiene dos objetivos principales, atender solicitudes de creencia que puedan recibirse y obtener las creencias más acertadas posibles. Para ello, tiene una serie de acciones que puede realizar para obtener información sobre sus hipótesis.

Una vez terminado el proceso de diagnóstico, el agente alcanza su primer objetivo enviando un informe del resultado al agente solicitante.

Modelo BDI	Agente de creencia
Creencias	• Agente de diagnóstico • Agentes de creencia • Agentes de observación • Conjunto de hipótesis • Conjunto de posibles acciones
Objetivos	• Atender solicitudes de creencia • Alcanzar diagnóstico fiable
Planes	• Solicitar creencia • Solicitar observación • Realizar observación • Enviar informe de diagnóstico

TABLA 34: MODELO BDI - AGENTE DE CREENCIA

En la Ilustración 26 se muestra el diagrama de actividad que tiene un agente de creencia.

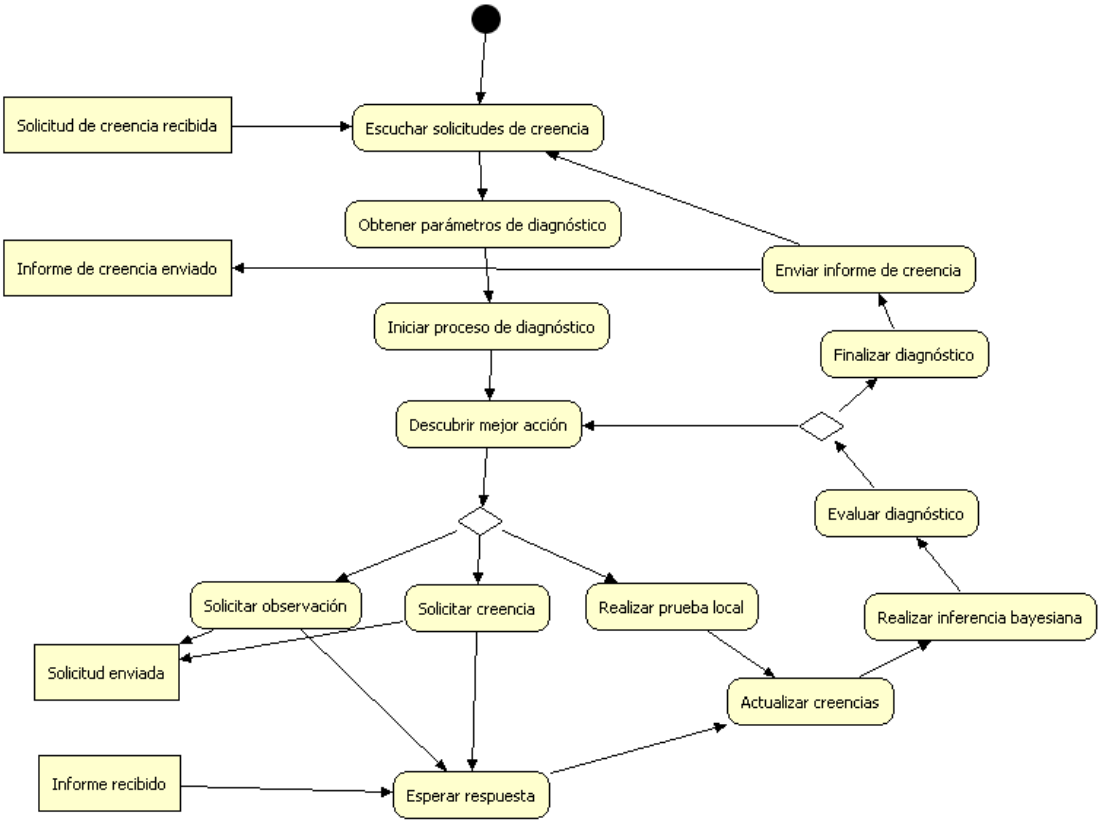

ILUSTRACIÓN 26: DIAGRAMA DE ACTIVIDAD - AGENTE DE CREENCIA

- En el estado inicial, comienza escuchando los mensajes de solicitud de creencia de un agente de un agente de diagnóstico o de otro agente de creencia.
- Una vez recibido un mensaje de solicitud de creencia, obtiene los parámetros que contiene el mensaje para poder decidir qué acción tomar dada la información en ese instante.
- Se inicia el proceso de diagnóstico.
- En este instante se decide por la mejor acción para tomar que dependerá tanto del coste como de la región en la que se quiere actuar.
- Si decide solicitar una observación o una creencia a un agente remoto, envía la solicitud correspondiente y espera la respuesta.
- Si decide realizar una prueba local, la ejecuta y guarda los datos obtenidos dentro de la operación.
- Una vez realizada la acción y obtenida nueva información, se actualiza la base de creencias.
- Se realiza inferencia bayesiana para intentar alcanzar un diagnóstico fiable.
- Con las hipótesis actualizadas gracias a la inferencia, se evalúa el estado actual del diagnóstico.
- Si aún no se ha superado el umbral de ninguna hipótesis, se vuelve a buscar la mejor acción que aún esté pendiente de realizar.

- Si se ha superado un umbral o no quedan más acciones que realizar, se finaliza el diagnóstico.
- Una vez finalizado, se genera un mensaje de informe de creencia.
- Tras enviar el mensaje, se vuelve al estado inicial en el que espera nuevas solicitudes de creencia.

El diagrama de secuencia que representa las interacciones de un agente de creencia con su entorno está representado en la Ilustración 27. El agente de creencia interactúa con el resto de agente mediante mensajes FIPA-ACL en los cuales viaja información asociada apropiada.

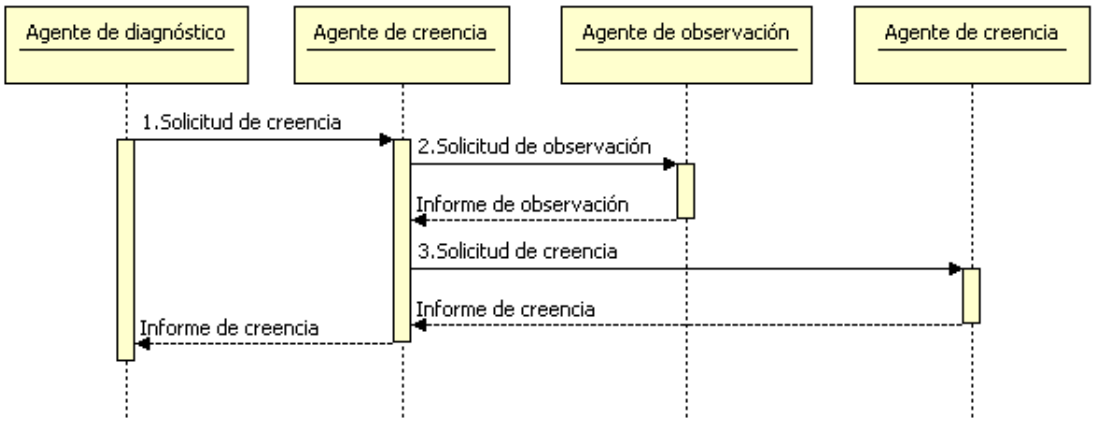

ILUSTRACIÓN 27: DIAGRAMA DE SECUENCIA - AGENTE DE CREENCIA

Profundizando más en el diseño del agente, se muestra en la Ilustración 28 un diagrama de componentes donde se exponen los módulos usados por este agente.

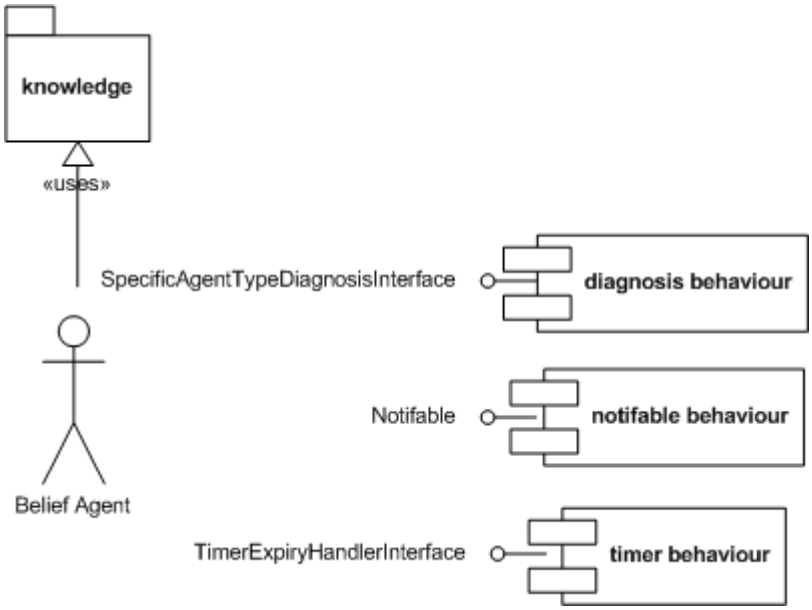

ILUSTRACIÓN 28: DIAGRAMA DE COMPONENTES - AGENTE DE CREENCIA

En este diagrama se observa que el agente de creencia usa los mismo recursos que el agente de observación: "*knowledge*" para interactuar con la ontología o los otros agentes; "*diagnosis behaviour*" para instanciar el proceso de diagnóstico; "*notifable behaviour*" para recibir notificaciones y "*timer behaviour*" para liberar recursos del dispositivo.

6.3.2.5 DISEÑO DE AGENTE DE CONOCIMIENTO

El agente de conocimiento ofrece, al administrador del sistema de diagnóstico, un modo de actualizar el conocimiento de cualquiera de los agentes que tengan una red bayesiana para realizar inferencia y así diagnosticar. Este mecanismo está diseñado para ser fácilmente adaptable a un sistema de auto-aprendizaje que generaría nuevas redes bayesianas basadas en datos recolectados de antiguos diagnósticos.

Este tipo de agente está a la espera de una orden del administrador del sistema. Cuando éste proporciona una nueva red bayesiana, informa a los agentes interesados en el conocimiento proporcionado. Dicha red debe estar apropiadamente configurada para que el sistema pueda obtener de ella la información necesaria para desplegar correctamente el conocimiento mediante la operación para distribuir conocimiento (ver sección 6.3.1.3).

En la Tabla 35 se muestra el modelo BDI que representa el comportamiento de un agente de conocimiento. Este tipo de agente tiene conocimiento sobre los diferentes agentes a los que puede notificar la aparición de nuevos conocimientos a través de los servicios de suscripción de la plataforma de agentes (ver Anexo I: Plataforma de agentes – JADE) y también el conocimiento que es capaz de transmitir.

Comienza escuchando las actualizaciones de conocimiento por parte del administrador del sistema. Tras recibirlo, parsea adecuadamente el conocimiento al dominio de la ontología (ver Anexo III: Adquisición de conocimiento – Genie & SMILE) y comienza el proceso necesario para desplegar adecuadamente el conocimiento. Para cumplir este objetivo, primero busca qué agentes están interesados en el conocimiento que el administrador ha suministrado al agente de conocimiento y los notifica con la operación adecuada.

Modelo BDI	Agente de conocimiento
Creencias	• Agentes destinatarios • Conocimiento adquirido
Objetivos	• Atender peticiones del administrador • Desplegar conocimiento
Planes	• Parsear conocimiento • Buscar agentes destinatarios

• Distribuir conocimiento

TABLA 35: MODELO BDI - AGENTE DE CONOCIMIENTO

En la Ilustración 29 se presenta el diagrama de actividad que el agente de conocimiento presenta durante su ejecución.

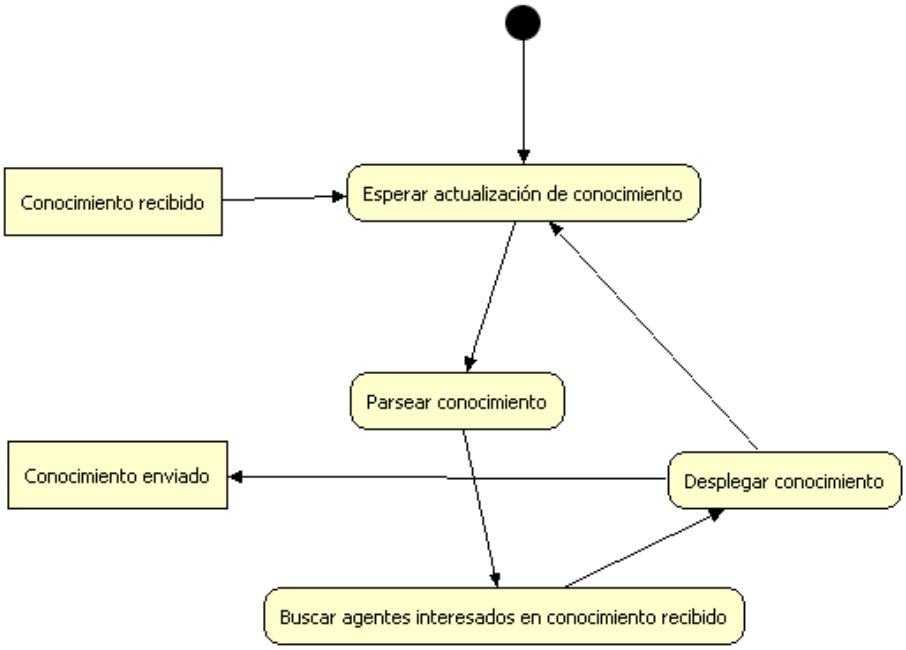

ILUSTRACIÓN 29: DIAGRAMA DE ACTIVIDAD - AGENTE DE CONOCIMIENTO

- En el estado inicial, un agente de conocimiento espera la acción del administrador del sistema
- Cuando recibe un nuevo conocimiento, procede a parsearlo de tal modo que se amolde a la ontología usada por el sistema (ver sección 6.3.1).
- Con la ontología en un estado adecuado, el agente busca a los agentes interesados en el conocimiento recibido.
- Con los destinatarios fijados, se procede al envío del nuevo conocimiento.
- Y se vuelve al estado inicial para esperar las órdenes del administrador.

Para terminar de exponer la funcionalidad del agente de conocimiento, en la Ilustración 30 se presenta la interacción del agente con su entorno. Con el administrador del sistema interactúa mediante una interfaz gráfica que le permite a éste seleccionar que archivo de conocimiento (ver Anexo III: Adquisición de conocimiento – Genie & SMILE) quiere distribuir en un instante dado y con el resto de agentes se comunica mediante mensajes FIPA-ACL.

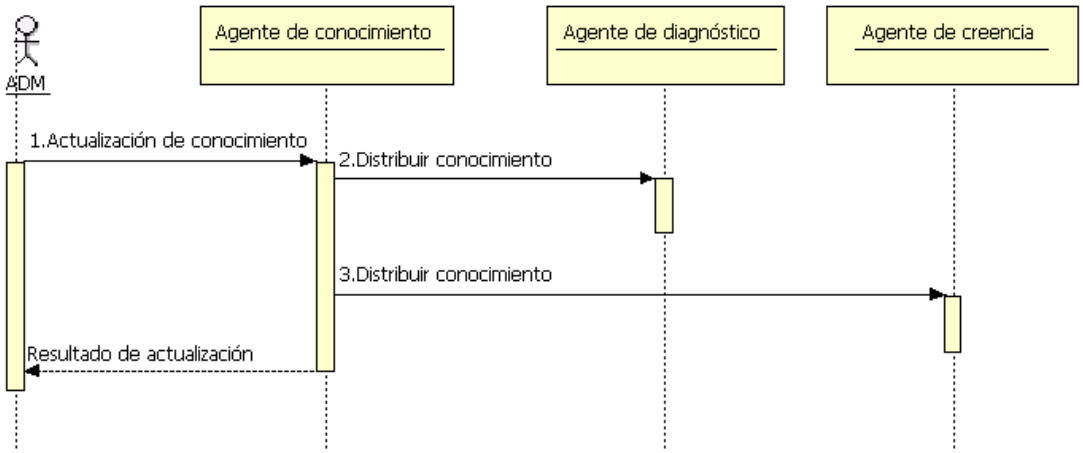

ILUSTRACIÓN 30: DIAGRAMA DE SECUENCIA - AGENTE DE CONOCIMIENTO

Profundizando más en el diseño del agente, se muestra en la Ilustración 31 un diagrama de componentes donde se exponen los módulos usados por este agente.

En este diagrama se observa como el agente de conocimiento usa el recurso "*knowledge*" para crear instancias de la ontología y también los mensajes necesarios para enviar las diferentes notificaciones.

También implementa un interfaz que permite usar el componente "*notifier behaviour*" para notificar apropiadamente el conocimiento sólo a aquellos agentes que estén interesados en ese tipo de conocimiento.

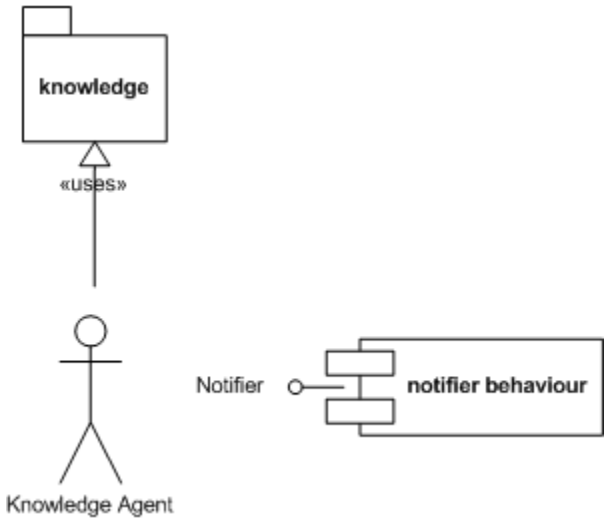

ILUSTRACIÓN 31: DIAGRAMA DE COMPONENTES - AGENTE DE CONOCIMIENTO

6.3.3 INSTANCIAS DE AGENTES EN EL CASO DE ESTUDIO

En esta sección se quiere presentar cómo funciona el sistema de diagnóstico de fallos en una conexión extremo a extremo en la que se da una compartición de archivos multimedia en tiempo real. Para ello, se exponen las diferentes ubicaciones dónde se ejecutan los agentes y las instancias de los diferentes tipos dentro del caso de estudio.

Dentro del escenario de la Ilustración 32, se presentan diferentes puntos donde podrían desplegarse diferentes grupos de agentes.

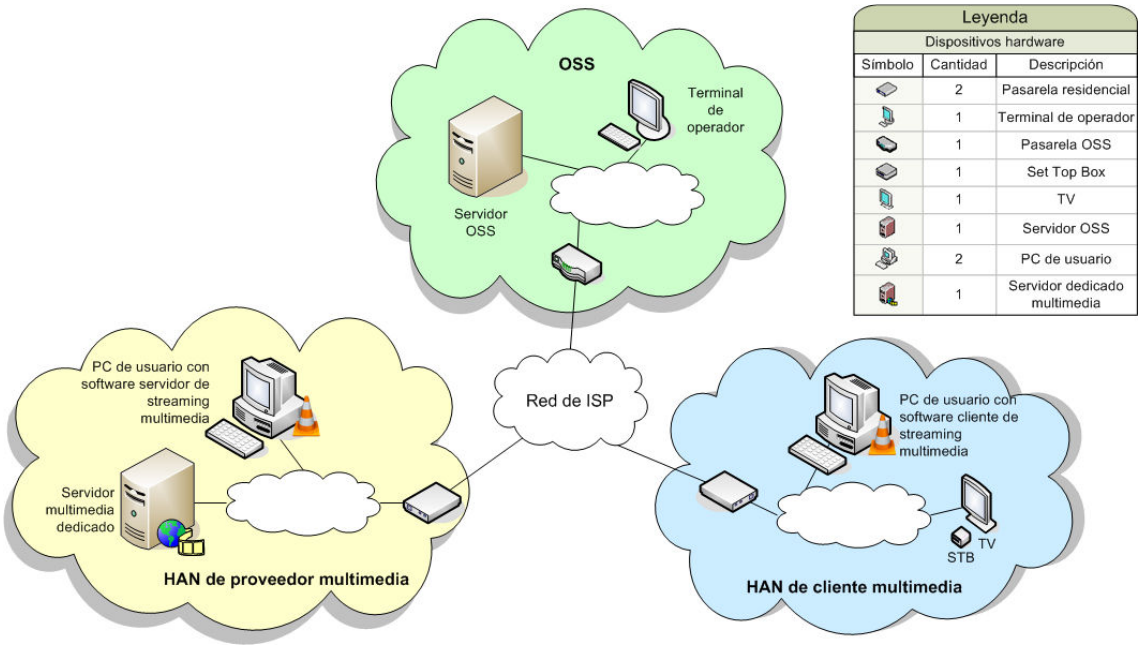

ILUSTRACIÓN 32: ESCENARIO DE IMPLANTACIÓN DE AGENTES

Dadas las propiedades de la plataforma de agentes que se usa (ver Anexo I: Plataforma de agentes – JADE), los agentes pueden ser desplegados en cualquier máquina que tenga el entorno de ejecución de java, ya sea en su versión estándar (J2SE) o en su versión móvil (J2ME).

Las posibles configuraciones de la arquitectura multi-agente, dada la topología del escenario, se muestran en las siguientes subsecciones en las que se presentan las diferentes ubicaciones en las que podría ser ejecutado un agente y las ventajas e inconvenientes de cada una.

6.3.3.1 INSTANCIAS DE AGENTES DE INTERFAZ

Este tipo de agente se comunica con los actores DAL y REP (ver Tabla 6: Diccionario de actores); por lo tanto, su ubicación dependerá de dónde resida la plataforma OSGi en la que se ejecuten estos dos sistemas externos.

Las diferentes ubicaciones en las que podría residir el agente de interfaz serían las mostradas en la Tabla 36 .

Ubicación de agente de interfaz	Ventajas	Inconvenientes
Dispositivo cliente de contenido multimedia	- Detección de fallos más rápida	- Necesidad de un agente de interfaz por cliente - No reside plataforma OSGi
Pasarela residencial del cliente	- Detección de fallos más rápida	- Agregación de varios clientes en un solo agente de interfaz - Reside en plataforma OSGi

TABLA 36: UBICACIÓN DE AGENTE DE INTERFAZ

Dadas las ventajas e inconvenientes mostrados para cada ubicación, la seleccionada para este escenario concreto ha sido la segunda de ellas, la pasarela residencial del cliente.

La otra de las opciones, los diferentes dispositivos cliente, conllevaría un trabajo adicional para recibir y enviar los diferentes eventos desde o hacia la plataforma OSGi, además de la necesidad adicional de recursos que pueden ser escasos en muchos de los distintos dispositivos multimedia.

La presencia del agente de interfaz dentro de la plataforma de servicios OSGi permite tener un acceso mucho más simple y eficiente a los sistemas externos con los que éste interactúa. Además se optimizan los recursos del sistema, ya que se reutiliza el mismo agente para todos los dispositivos que usen esa misma pasarela residencial para consumir los diferentes recursos multimedia.

En resumen, en el escenario hay un agente de interfaz que reside en la pasarela residencial de la red cliente (consumidor de recursos multimedia) teniendo acceso a los diferentes servicios de la plataforma OSGi que se ejecuta en la misma pasarela.

6.3.3.2 INSTANCIAS DE AGENTE DE DIAGNÓSTICO

Un agente de diagnóstico interactúa con el resto de tipos de agentes del sistema a través del sistema de mensajería de la plataforma de agentes (ver Ilustración 21: Diagrama de secuencia - Agente de diagnóstico). Dado que la solicitud de diagnóstico proviene de un agente de interfaz que residirá dentro de la red local del cliente consumidor de contenido multimedia, las posibles ubicaciones dentro del escenario son las presentadas en la Tabla 37.

Ubicación de agente de diagnóstico	Ventajas	Inconvenientes
Red del proveedor de conexión	- Agregación de diagnósticos de diferentes clientes en un solo agente	- Mayor trasiego de mensajes para hacer pruebas locales - Imposibilidad de diagnóstico antes problemas de conectividad
Pasarela residencial del cliente	- Posibilidad de diagnóstico de fallos de conectividad -Menor trasiego de mensajes para hacer pruebas locales	- Necesidad de un agente de diagnóstico por cada pasarela residencial

TABLA 37: UBICACIÓN DE AGENTE DE DIAGNÓSTICO

Con el análisis presentado en dicha tabla, se ha optado por ubicar este tipo de agente en la pasarela residencial.

Esta opción permite obtener diagnósticos aún teniendo problemas de conectividad externa y reduce el número de mensajes que se transmiten a la red externa del proveedor de la conexión. No obstante, optar por esta ubicación obliga a desplegar un agente de diagnóstico por cada pasarela residencial.

En cambio, la posibilidad de desplegar un agente de diagnóstico común para varios clientes dentro de la red del proveedor de conexión permite una mayor eficiencia de recursos de procesamiento, aunque una pérdida en la eficiencia del envío de mensajes, ya que se deberían solicitar las pruebas locales de manera remota (a través de mensajes). No obstante, estos dos hechos no son del todo determinantes, el mayor inconveniente es la pérdida de capacidad de diagnóstico de fallos ante la pérdida de conectividad interna.

En resumen, en el escenario hay un agente de interfaz que reside en la pasarela residencial de la red cliente (consumidor de recursos multimedia) con el fin de diagnosticar fallos de conectividad.

6.3.3.3 INSTANCIAS DE AGENTES DE CREENCIA

Un agente de creencia se comunica tanto con agentes de diagnóstico como con otros agentes de creencia o de observación. Su función es disminuir la carga que tiene el agente de diagnóstico durante dicha operación, permitiendo englobar bajo un mismo agente de creencia la parte de diagnóstico que se dé en una región con ciertas singulares, como pueden ser áreas físicamente distantes (reduciendo el número de mensajes intercambiados) o áreas con diferentes restricciones de acceso (por ejemplo, redes privadas). Con estas posibilidades del agente de creencia, la presencia de un

agente podría ser útil dentro del escenario en las ubicaciones presentadas en la Tabla 38.

Ubicación de agente de creencia	Ventajas	Inconvenientes
Red del proveedor de conexión	- Delegación de diagnóstico	- Necesidad de recursos adicionales en los sistemas de soporte de operación
Pasarela residencial del proveedor	- Delegación de diagnóstico	- Necesidad de recursos adicionales en la pasarela residencial

TABLA 38: UBICACIÓN DE AGENTE DE CREENCIA

Las dos ubicaciones posibles presentan las mismas ventajas e inconvenientes y dado que la posibilidad de distribuir el diagnóstico delegando parte de éste en diferentes agentes de creencia hace despreciable el coste de despliegue y operación.

En el escenario se despliegan agentes de creencia en ambas ubicaciones. Este despliegue ofrece la posibilidad de delegar la parte del diagnóstico concerniente a la red de la operadora en un agente y la parte concerniente a la red del proveedor de contenidos multimedia a otro agente.

6.3.3.4 INSTANCIAS DE AGENTES DE OBSERVACIÓN

Un agente de observación puede interactuar tanto con agentes de diagnóstico como con agentes de creencia. Pero éste no es el principal criterio de decisión a la hora de discernir la ubicación de un agente de observación, dicho criterio está basado en qué y dónde se va a realizar la prueba que el agente ejecuta.

Las diferentes pruebas realizadas pueden comprobar el estado tanto de dispositivos como de servicios o sistemas externos, por lo tanto, se presenta la posibilidad de desplegar agentes de observación en las ubicaciones mostradas en la Tabla 39.

Ubicación de agente de observación	Ventajas	Inconvenientes
Dispositivo monitorizado	- Pruebas más críticas	- Dispositivos con baja capacidad de procesamiento
Pasarela residencial	- Pruebas más restringidas - Servicios de gestión residentes en la pasarela residencial	- Necesidad de recursos adicionales en la pasarela residencial - Necesidad de interfaz con el dispositivo o servicio a monitorizar

TABLA 39: UBICACIÓN DE AGENTE DE OBSERVACIÓN

La opción de desplegar agentes de observación en el mismo dispositivo que se quiere monitorizar sería la más adecuada en el caso de que el dispositivo ofrezca los requisitos necesarios para soportar la plataforma de agentes. No obstante, la mayoría de los dispositivos para ajustar los costes de producción reducen al mínimo las características de sus equipos; lo cual imposibilita la opción de desplegar agentes en la mayoría de los dispositivos multimedia.

La otra opción que se presenta es desplegar agentes en la pasarela residencial y que dichos agentes tengan diferentes interfaces para interactuar con los dispositivos. Lo cual proporciona un acceso inmediato a la gestión de servicios, en el caso de estudio, al sistema de gestión de recursos multimedia (ver Tabla 6: Diccionario de actores).

6.3.3.5 INSTANCIAS DE AGENTES DE CONOCIMIENTO

El agente de conocimiento presenta una ventana de interacción con el administrador del sistema y una vía abierta a futuros procesos de aprendizaje para las redes bayesianas (ya que ofrece el mecanismo de despliegue de conocimiento bajo petición a los agentes interesados en dicho conocimiento). Con estas características y teniendo en cuenta que los algoritmos de aprendizaje conllevan un proceso costoso de ejecutar en el sentido de requisitos de computación, la ubicación más viable de despliegue de un agente de conocimiento es en la red del proveedor de servicio (en el OSS).

Ubicación de agente de conocimiento	Ventajas	Inconvenientes
Red de proveedor de conexión	- Capacidad suficiente de procesamiento para algoritmos de aprendizaje - Agregación de funcionalidad para diferentes clientes en un solo agente de conocimiento	- Mecanismo de notificación de conocimiento no tolerante a desconexión
Pasarela residencial	- Aprendizaje local con tolerancia a desconexión	- Recursos escasos para procesar algoritmos de aprendizaje - Necesidad de un agente de conocimiento por pasarela residencial

TABLA 40: UBICACIÓN DE AGENTE DE CONOCIMIENTO

La posibilidad de desplegar un agente de conocimiento en los sistemas de soporte de operación de la operadora que ofrece el servicio de diagnóstico de fallos ofrece unas ventajas que superan claramente los inconvenientes que presentan.

La primera de estas es la abundancia de recursos en dichos sistemas de soporte que permitirían la actualización de una versión mejorada del agente de conocimiento que permitiera recolectar datos y ejecutar algoritmos de aprendizaje.

La segunda ventaja es a su vez su inconveniente. A la vez que tener un agente de conocimiento desplegado en un punto neutral de la red permite usar sus mecanismos para actualizar conocimiento en cualquier agente del sistema de diagnóstico, también tiene el inconveniente de perder esta capacidad en el caso de falta de conexión.

La alternativa a esta ubicación de despliegue es instanciar un agente de conocimiento por cada pasarela residencial. Esta opción permitiría en una futura versión del agente de conocimiento realizar aprendizaje de manera local. No obstante, los requisitos de la pasarela residencial crecerían desmesuradamente, tanto para almacenar datos como para ejecutar algoritmos de aprendizaje.

Finalmente, la opción tomada en el caso de estudio es el despliegue de un agente de conocimiento común para todo el sistema de diagnóstico en los sistemas de soporte de operación de la operadora de comunicaciones que ofrece la conexión.

7. PLAN DE DESPLIEGUE

7.1 INTRODUCCIÓN

Este capítulo presenta el modo de desplegar el sistema en el escenario, para lo cual es un requisito indispensable una descripción detallada de los recursos necesarios. Con el fin de presentar de manera clara dichos recursos para la implantación del sistema de diagnóstico, en las secciones siguientes se exponen los recursos hardware y software necesarios en el escenario.

7.2 RECURSOS NECESARIOS

En la Ilustración 33 se presenta un diagrama UML de despliegue en el que se muestran los equipos necesarios para desplegar el sistema de diagnóstico presentado en el capítulo 6. En dicho diagrama se presentan también las restricciones o requisitos que deben cumplir los diferentes dispositivos.

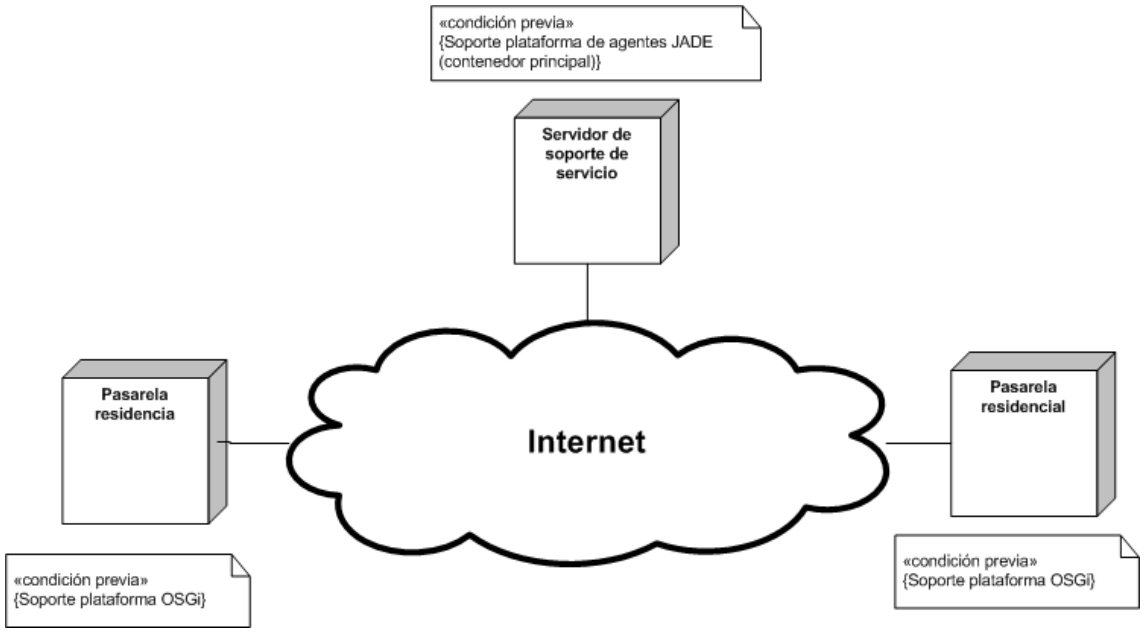

ILUSTRACIÓN 33: DIAGRAMA DE DESPLIEGUE

7.2.1 RECURSOS HARDWARE

Además de los dispositivos presentados como nodos en la Ilustración 33, se requieren también elementos que permitan la conexión entre los diferentes dispositivos, en el caso de estudio, se daría la necesidad de una conexión a internet por parte de los dos usuarios que comparten recursos multimedia (cliente y proveedor).

Recurso	Cantidad	Descripción
Pasarela residencial	1 / usuario	La pasarela residencial debe ser de altas prestaciones para poder dar soporte a la plataforma de servicios OSGi y al sistema de diagnóstico.
Servidor de soporte de servicio	1 / región	El servidor de soporte es necesario para dar infraestructura a la plataforma de agentes.
Infraestructura de comunicación	1	La infraestructura de comunicaciones es necesaria para dar conexión entre los diferentes equipos.

TABLA 41: PLAN DE DESPLIEGUE - RECURSOS HARDWARE

Teniendo en cuenta la información presentada en la Tabla 41, se debe puntualizar la posibilidad de aumentar el número de servidores de soporte si el número de diagnóstico es elevado. Así, desplegando diferentes servidores en regiones especificas, se puede aumentar la escalabilidad y el rendimiento del sistema.

Otro punto a tener en cuenta es que la infraestructura de comunicación no tiene ningún requisito de tecnología de acceso, mientras la conectividad entre las diferentes partes siga activa, puede ser ADSL, fibra óptica, acceso inalámbrico o cualquier otra.

7.2.2 RECURSOS SOFTWARE

Para desplegar debidamente el software desarrollado se necesitan una serie de recursos software, como las plataformas de soporte sobre las que se han desarrollado los diferentes elementos del sistema.

Recurso	Ubicación	Descripción
Plataforma de servicios OSGi	Pasarela residencial	Esta plataforma ofrece el soporte para la interacción con los diferentes actores con el sistema de diagnóstico. Además presenta muchas facilidades para el administrador en el despliegue del software.
Plataforma de agentes	Servidor de soporte / Pasarela residencial	Esta plataforma es necesaria para conseguir la comunicación entre los diferentes agentes del sistema.
Servidor HTTP	Servidor de soporte	Este servidor sirve como repositorio de *bundles* para desplegar en las pasarelas residenciales.
Bundles OSGi	Pasarela residencial	Estos *bundles* representan a algunos de los actores del sistema y son necesarios para que el sistema pueda interactuar con estos.

TABLA 42: PLAN DE DESPLIEGUE - RECURSOS SOFTWARE

Los sistemas expuestos en la Tabla 42 son genéricos en el sentido que no se habla en concreto de ninguna distribución de software sino de sistemas o herramientas.

Las distribuciones usadas en el desarrollo del sistema y las expuestas en la sección "Implantación del sistema" son las presentadas en la Tabla 43.

Sistema	Distribución
Plataforma de servicios OSGi	Apache Felix
Plataforma de agentes	JADE-OSGi/WADE
Servidor HTTP	Apache Tomcat

TABLA 43: PLAN DE DESPLIEGUE - DISTRIBUCIONES SOFTWARE

Los *bundles* OSGi son implementaciones de los diferentes sistemas con los que el sistema de diagnóstico va a interactuar durante la ejecución.

7.3 IMPLANTACIÓN DEL SISTEMA

En esta sección se presenta el proceso a seguir para realizar un correcto despliegue del sistema. Se expone en tres secciones que definen los pases básicos a seguir: instalación, configuración y arranque.

7.3.1 INSTALACIÓN

A continuación se muestra cómo instalar el software necesario para poder ejecutar el sistema. Se hace una distinción explicita entre el software necesario en la pasarela residencial y en el servidor de soporte de operación.

7.3.1.1 SERVIDOR DE SOPORTE DE OPERACIÓN

Dentro del servidor deben ejecutarse diferentes plataformas y servicios. La herramienta principal es la plataforma de agentes que dará soporte de contenedor principal al resto de ubicaciones de agentes (ver Anexo I: Plataforma de agentes – JADE); ya que gracias a este contenedor, el resto de agentes podrán encontrar los servicios que exportan otros y así poder alcanzar un diagnóstico coherente mediante una mutua colaboración. No obstante, en el servidor de soporte se utiliza la aplicación WADE que permite unas opciones de despliegue más convenientes para el sistema.

El otro servicio necesario es el servidor HTTP que se usa como repositorio a través del cual se puede gestionar el ciclo de vida de los diferentes *bundles* que se instalan en la pasarela residencial. Gracias a la gestión que ofrece la plataforma OSGi, las actualizaciones de nuevas versiones de software son prácticamente transparentes si se configuran adecuadamente los *bundles* del sistema.

Herramienta/Servicio	Descripción
WADE	Aplicación basada en la plataforma JADE que ofrece características apropiadas para el sistema.
Apache Tomcat	Servidor HTTP que hace la función de repositorio de software para desplegar o actualizar *bundles* dentro de la pasarela residencial.

TABLA 44: SOFTWARE EN EL SERVIDOR DE SOPORTE

No obstante, también debe apuntarse que el repositorio de *bundles* (servidor HTTP en este caso) no tiene porque residir en la misma máquina que la plataforma de agentes, pero se muestra así en el despliegue para simplificar el plan de despliegue. En el caso de que el sistema saturara el servidor de soporte, podría usarse un segundo servidor.

7.3.1.2 PASARELA RESIDENCIAL

Sobre la pasarela residencial del usuario debe correr principalmente la plataforma de servicios OSGi, en este caso en particular se presupone la instalación anterior de la plataforma Apache Felix (para más detalles sobre la instalación y uso de Apache Felix, ver Anexo II: Plataforma de servicios OSGi – Apache Felix).

Con la plataforma de servicios activa, el administrador del sistema sólo tiene que acceder a esta de manera remota para poder instalar los *bundles* necesarios para la ejecución del sistema mediante la consola de la plataforma.

En este punto se hace una distinción entre dos tipos de *bundles*, los necesarios para interactuar con los necesarios con los sistemas externos (no pertenecientes al sistema de diagnóstico) y los *bundles* propios del sistema de diagnóstico.

Bundle	Descripción
Jade-Osgi	Da soporte de a la plataforma de agentes dentro de OSGi ofreciendo un servicio de creación de agentes que es usado por el sistema de diagnóstico.
Apache Felix Remote Shell	Permite acceso remoto a la consola de la plataforma OSGi, ofreciendo así la posibilidad de gestionar todo el software a distancia.
Apache Felix EventAdmin	Ofrece un sistema de mensajería a través de eventos que es utilizado para interactuar con algunos actores del sistema.
Spring-Osgi	Ofrece una serie de librerías para la carga dinámica de *bundles* en tiempo de ejecución.
Device Adaptation Layer	Es uno de los actores del sistema. Notifica sobre la detección de fallos en un dispositivo.
Report Notifier	Es uno de los actores del sistema. Escucha los informes de diagnóstico para mostrarlos debidamente.
Multimedia Resource Manager	Es uno de los actores del sistema. Gestiona la compartición de recursos entre diferentes usuarios y tiene información sobre ellos. Se usa como sistema de consulta sobre dichos recursos.

TABLA 45: LISTADO DE BUNDLES EXTERNOS

Todos los *bundles* mostrados en la Tabla 45 y en la Tabla 46 están dentro del repositorio que se ofrece en el servidor HTTP dentro del servidor de soporte de operación.

Bundle	Descripción
Jade Agents Loader	Permite la inicialización de agentes de manera más sencilla a través de ficheros de configuración en tiempo de ejecución.
Fault Diagnosis	Contiene los agentes dentro del escenario.

TABLA 46: LISTADO DE BUNDLES DEL SISTEMA

El administrador del sistema debe tener acceso tanto al servidor como a la pasarela residencial. Para obtener más datos sobre la instalación de *bundles* dentro de la plataforma de servicios, véase el Anexo II: Plataforma de servicios OSGi – Apache Felix.

7.3.2 CONFIGURACIÓN

Esta sección muestra como configurar los diferentes sistemas para poder ejecutar el sistema de diagnóstico correctamente. La configuración se expone de manera genérica de manera que se conozca cómo configurar el sistema para otros posibles escenarios además del caso de estudio.

Para configurar la plataforma OSGi se deben modificar algunos archivos de configuración en el directorio correspondiente a ese mismo propósito (ver sección Anexo II: Plataforma de servicios OSGi – Apache Felix). Mientras que en el servidor de soporte se debe configurar la plataforma de agentes de modo que puedan comunicarse todos los agentes debidamente.

7.3.2.1 SERVIDOR DE SOPORTE DE OPERACIÓN

En el servidor de soporte de servicio se deben configurar tanto la plataforma de agentes como el servidor HTTP que servirá de repositorio.

A continuación se muestra la configuración de la plataforma de agentes, ya que el servidor Apache Tomcat puede funcionar en su configuración por defecto sin problemas.

Al igual que en la pasarela residencial, la plataforma de agentes se configura a través de archivos que se leen en el inicio de la aplicación. En la aplicación WADE tienen especialmente importancia tres ficheros que se muestran a continuación. No obstante, si se desea más información sobre la plataforma, véase el Anexo I: Plataforma de agentes – JADE / WADE.

El fichero más importante es *"main.properties"*. En él se fijan las configuraciones más importantes del contenedor principal de la plataforma de agentes.

```
#
# WADE Main Container property file.
#
local-port=Puerto en el que escuchará la plataforma los mensajes de
      otros contenedores

#
# WADE properties
#
cfa_bootdaemon_port=Puerto en el que escucha un proceso que arranca
      agentes bajo petición

agents=cfa:com.tilab.wade.cfa.ConfigurationAgent;raa:com.tilab.wade.ra
      a.RuntimeAllocatorAgent;gui:com.tilab.wade.tools.management.Mana
      gementAgent

services=jade.core.messaging.TopicManagementService;jade.core.mobility
      .AgentMobilityService;jade.core.event.NotificationService;jade.c
      ore.replication.MainReplicationService;jade.core.nodeMonitoring.
      UDPNodeMonitoringService

#
# JADE properties
#
platform-id=ID de la plataforma
gui=true
nomtp=true
jade_core_AgentContainerImpl_enablemonitor=true
jade_core_MainContainerImpl_replicatedagents=com.tilab.wade.cfa.Config
      urationAgent;jade.tools.rma.rma;com.tilab.wade.raa.RuntimeAlloca
      torAgent;gui:com.tilab.wade.cfa.ManagementAgent

jade_core_nodeMonitoring_UDPNodeMonitoringService_pingdelaylimit=30000
jade_core_nodeMonitoring_UDPNodeMonitoringService_unreachablelimit=360
      0001

# DF configuration
jade_domain_df_maxresult=5000
jade_domain_df_autocleanup=true
```

CÓDIGO 1: SERVIDOR DE SOPORTE - ARCHIVO DE CONFIGURACIÓN DE CONTENEDOR PRINCIPAL DE AGENTES

Una vez configurada el contenedor principal de la plataforma, se debe configurar el conjunto de agentes que se quiere desplegar en el servidor de soporte. Para ello se deben modificar dos archivos: *"types.xml"* (Código 2) y *"_target.xml"* (Código 3). En el primero se especifican los diferentes tipos de agente y en el segundo, las instancias concretas de dichos tipos que se desean iniciar.

```
<platform>
      <agentRoles>
            <agentRole description="My Agent Type 1">
            </agentRole>
            <agentRole description="My Agent Type 2">
            </agentRole>
```

```
      </agentRoles>
      <agentTypes>
            <agentType description="My Agent"
            className="agent.MyAgentType1" role="My Agent Type 1">
                  <properties>
                        <property name = "MyParameter "
                  value="MyValue" />
                  </properties>
            </agentType>
      </agentTypes>
</platform>
```

CÓDIGO 2: SERVIDOR DE SOPORTE - ARCHIVO DE TIPOS DE AGENTES

```
<platform description="Descripción de la plataforma" name="ID de
plataforma">
  <hosts>
    <host name="Máquina en la que se vayan a ejecutar los agentes">
      <containers>
        <container name="MyContainer" jadeProfile="monitored">
         <agents>
             <agent name="MyAgent" type="MyType">
                  <parameters>
                          <parameter key="MyParameter">
                                  <value>MyValue</value>
                          </parameter>
                  </parameters>
             </agent>
          </agents>
        </container>
      </containers>
    </host>
  </hosts>
</platform>
```

CÓDIGO 3: SERVIDOR DE SOPORTE - ARCHIVO DE INSTANCIAS DE AGENTES

7.3.2.2 PASARELA RESIDENCIAL

En la pasarela residencial debe ejecutarse la plataforma de servicios y por lo tanto se deben configurar los diferentes *bundles* usados por el sistema.

La plataforma de servicios se configura a través de un directorio específico para este fin (ver Anexo II: Plataforma de servicios OSGi – Apache Felix). A través de diferentes archivos, los *bundles* pueden configurarse en el arranque. A continuación se muestra cómo configurar los diferentes archivos necesarios para ejecutar el sistema de diagnóstico.

Para configurar la plataforma de agentes en la pasarela residencial, se debe configurar el archivo *"system.properties"* dentro del directorio de configuración como se muestra a continuación.

```
#
#     JADE OSGI configuration properties
#
jade.container=true
```

```
jade.host=”IP o nombre de la máquina en la que reside el contenedor
      principal”
jade.port=”Puerto en el que escucha el contenedor principal”
jade.platform-id=”ID de la plataforma”
jade.container-name=”Nombre del contenedor de agentes”
jade.services=jade.core.nodeMonitoring.UDPNodeMonitoringService;jade.c
      ore.messaging.TopicManagementService;jade.core.messaging.Messagi
      ngService;jade.core.event.NotificationService
jade.nomtp=true
jade.jade_core_AgentContainerImpl_enablemonitor=true
jade.jade_core_nodeMonitoring_UDPNodeMonitoringService_pingdelay=2000
jade.jade_core_nodeMonitoring_UDPNodeMonitoringService_pingdelaylimit=
      30000
jade.jade_core_nodeMonitoring_UDPNodeMonitoringService_unreachablelimi
      t=3600000
java.util.logging.config.file=./conf/logging.properties
```

CÓDIGO 4: PASARELA RESIDENCIAL - ARCHIVO DE CONFIGURACIÓN DE PLATAFORMA JADE

Para obtener unas trazas de la ejecución del sistema más limpias, se usa una configuración específica que se define en el archivo *“logging.properties”*.

```
#
# Logging properties
#
handlers=java.util.logging.ConsoleHandler,
      java.util.logging.FileHandler
java.util.logging.ConsoleHandler.level = ALL
java.util.logging.ConsoleHandler.formatter=java.util.logging.SimpleFor
      matter
java.util.logging.FileHandler.level = ALL
java.util.logging.FileHandler.pattern= ./log/JADElogs%u.log
java.util.logging.FileHandler.limit=50000
java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormat
      ter
jade.core.level = FINE
```

CÓDIGO 5: PASARELA RESIDENCIAL - ARCHIVO DE CONFIGURACIÓN DE TRAZAS

Con estos dos archivos de configuración, la plataforma de agentes queda completamente operativa y lista para comenzar a arrancar agentes sobre ella.

A continuación se debe configurar apropiadamente el *bundle* “*Jade Agent Loader*” que se encarga de arrancar en tiempo de ejecución los diferentes agentes que se deban desplegar en la pasarela residencial. Para ello, en el archivo de configuración *“agents.xml”* se debe especificar qué agentes arrancar y sus respectivos parámetros como se muestra a continuación.

```
<agents>
      <agent name="Nombre del agente" bundle="Bundle en el que
      encontrar el agente" className="Clase de implementación del
      agente (incluyendo jerarquía de paquetes)">
            <parameters>
                  <parameter key="Nombre del parámetro"
                  value="Valor del parámetro"/>
            </parameters>
```

```
      </agent>
      <!--EJEMPLO DE AGENTE-->
      <agent name="–" bundle="bundle1" className="agent.myAgent">
            <parameters>
                  <parameter key="FileNumber" value="23" />
                  <parameter key="LineNumber" value="12" />
            </parameters>
      </agent>
</agents>
```

CÓDIGO 6: PASARELA RESIDENCIAL - ARCHIVO DE CONFIGURACIÓN DE AGENTES

7.3.3 ARRANQUE

En esta sección se presenta la manera de arrancar el sistema cuando ya está correctamente configurado. Para ello se deben seguir unos determinados pasos en un orden concreto.

En primer lugar se debe iniciar la parte del sistema que se ejecuta en el servidor de soporte de operación y, una vez iniciado, se deben iniciar los *bundles* apropiados en las diferentes pasarelas residenciales.

7.3.3.1 SERVIDOR DE SOPORTE DE OPERACIÓN

Para iniciar el sistema, una vez configurados los ficheros descritos en la sección de configuración, se deben seguir los siguientes pasos en el orden descrito.

1. Bajo el directorio de la aplicación WADE, se debe ejecutar el script *"./startBootDaemon.sh"* el cual inicia un proceso que inicializará agentes bajo petición.
2. En ese mismo directorio, se debe ejecutar el script *"./startMain.sh"* que inicia el contenedor principal de la plataforma, además de la interfaz gráfica de la aplicación WADE.
3. Una vez mostrada dicha interfaz, se deben iniciar los agentes descrito en el fichero de configuración *"_target.xml"*; para ello sólo se debe pulsar el botón "*Start*" en la interfaz gráfica de WADE o configurar la aplicación para que instancie los agentes automáticamente.

Con estos tres simples pasos, el servidor de soporte de operación está preparado para la ejecución del sistema.

7.3.3.2 PASARELA RESIDENCIAL

Para iniciar la parte del sistema que se ejecuta en la pasarela residencial, es crucial iniciar los *bundles* necesarios en un orden determinado mostrado a continuación.

En este punto se deben hacer algunas consideraciones iniciales:

- En cada plataforma de servicios OSGi (o en otras palabras, en cada pasarela residencial), cada *bundle* puede tener un identificador de *bundle* propio ya que pueden tener diferentes servicios ejecutándose en la plataforma.
- La explicación que se expone a continuación presupone que se usa la distribución Apache Felix y que la plataforma de servicios ya está iniciada.

Tras estas aclaraciones, en la siguiente lista se expone el orden en el que deben iniciarse los *bundles* que deben haber sido configurados (como se muestra en la respectiva sección).

1. Iniciar el *bundle* "Jade-Osgi", que toma su configuración del fichero de configuración *"system.properties"*.
2. Iniciar el *bundle* "*Jade Agent Loader*", que inicia las instancias de agentes descritas en el fichero *"agents.xml"*. Para iniciar este *bundle* se debe tener instalado también el *bundle* de "*Fault Diagnosis*" que contiene las implementaciones del código de los agentes del sistema.

Con estos pasos y el resto de actores del sistema iniciados apropiadamente, el sistema de diagnóstico de fallos está listo para comenzar su ejecución con cada evento de detección de fallos que reciba.

8. PLAN DE PRUEBAS

8.1 INTRODUCCIÓN

El plan de pruebas tiene la función de verificar el correcto funcionamiento del sistema desarrollado comprobando que se cumplen los diferentes requisitos del funcionamiento.

Para facilitar la comprensión del plan de pruebas usado en el proyecto, este capítulo se estructura presentando, en primer lugar, la descripción de la maqueta que se ha utilizado para realizar los casos de prueba y, en segundo lugar, la descripción de dichos casos de prueba y los resultados obtenidos.

8.2 MAQUETA DE PRUEBAS

Dada la complejidad del escenario y la dificultad que presenta el acceso a distintos tipos de dispositivos requeridos, se ha construido una maqueta que simula el escenario real permitiendo así una administración completa de todos los sistemas que intervienen en el escenario.

8.2.1 DEFINICIÓN DE LA MAQUETA

Para lograr la reproducción de un escenario en el que se ofrezca un servicio de compartición de video, deben cumplirse la mayoría de los requisitos presentados en la sección 7.2. No obstante, algunos de los requisitos pueden cumplirse parcialmente y se necesitarán algunos otros requisitos adicionales para simular diferentes situaciones.

En la Ilustración 34 se muestra un diagrama de la maqueta de pruebas.

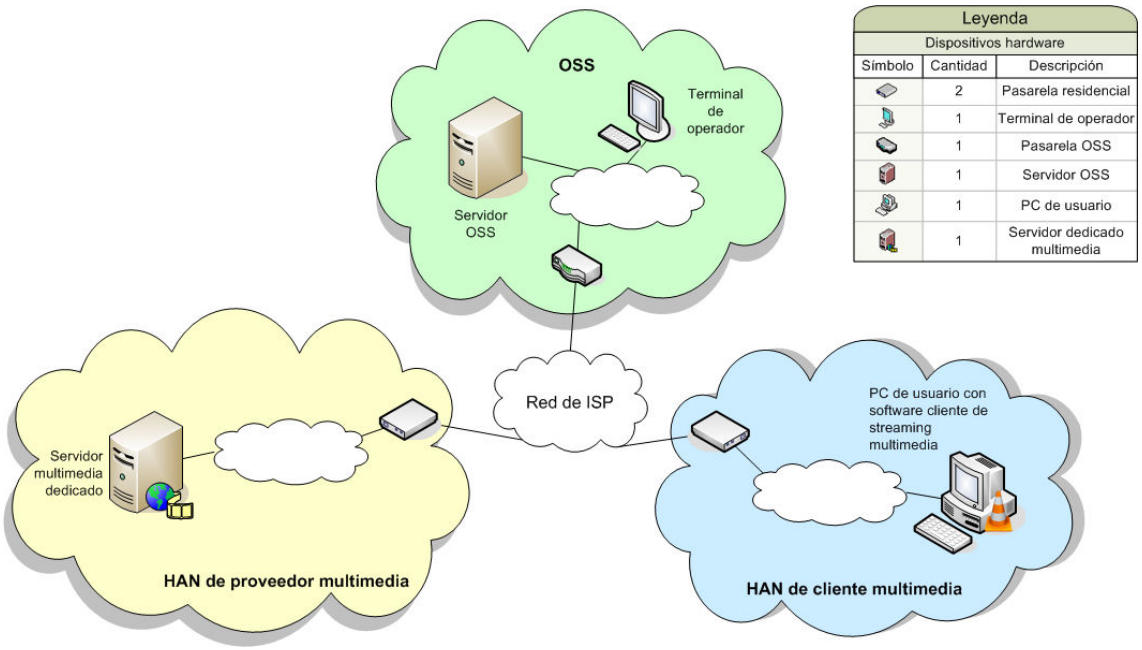

ILUSTRACIÓN 34: MAQUETA DE PRUEBAS

8.2.1.1 PASARELA RESIDENCIAL

Dada la dificultad que conlleva encontrar una pasarela residencial de un coste asequible, se ha optado tener dos pasarelas residenciales simuladas.

Para simular una pasarela residencial se necesitan varios dispositivos con unas determinadas características como se muestra en la Tabla 47.

Dispositivo	Descripción
Ordenador	Ordenador con capacidades básicas y dos interfaces de red.
Switch	Dispositivo de interconexión de redes que opera en la capa 2 del modelo OSI.
Cable Ethernet	Cable que permite la conexión entre el ordenador y el switch.

TABLA 47: DISPOSITIVOS DE PASARELA RESIDENCIAL SIMULADA

Para construir la pasarela residencial simulada a partir de estos elementos, sólo se ha conectado el cable Ethernet a una de las interfaces de red del ordenador y a una de las entradas del *switch*, como se muestra en la Ilustración 35.

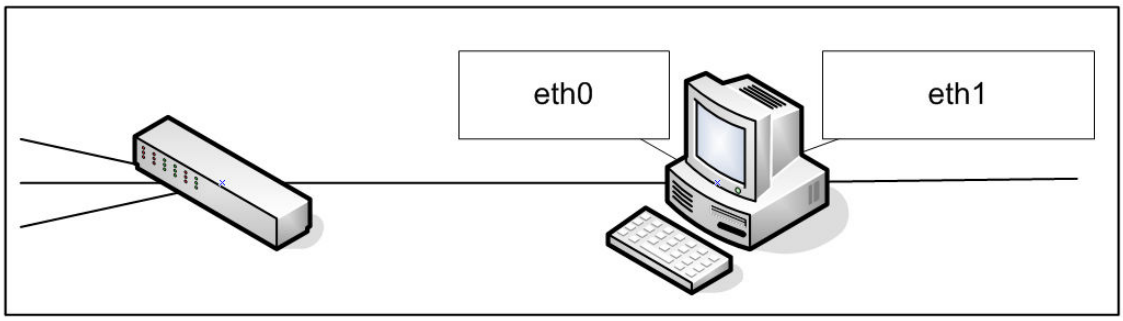

ILUSTRACIÓN 35: CONFIGURACIÓN DE LA PASARELA RESIDENCIAL SIMULADA

El software usado en el ordenador que contiene la capacidad de proceso de la pasarela residencial es, además del especificado en la sección "Recursos software" del Plan de despliegue, una serie de paquetes del núcleo del sistema operativo Linux que ofrecen la posibilidad de enrutar paquetes como lo haría una pasarela residencial.

Obviamente, además de esta capacidad de enrutado, también se usa como soporte para la plataforma OSGi, en la que se ejecutan parte de los agentes del sistema.

8.2.1.2 SERVIDOR DE SOPORTE DE OPERACIÓN

Como se muestra en el Plan de despliegue, el servidor de soporte de operación requiere cierto software para el despliegue del sistema.

8.2.1.3 CLIENTE MULTIMEDIA

El equipo consumidor de los recursos multimedia consta del sistema de reproducción multimedia "*VideoLAN VLC*". Se ha optado por este sistema porque es el que uno de los actores del sistema, el DAL (ver Tabla 6: Diccionario de actores) soporta este reproductor.

8.2.1.4 SERVIDOR MULTIMEDIA

El equipo servidor multimedia consta de un servidor RTSP llamado "Live555".

8.2.1.5 CONECTIVIDAD

En la conectividad del sistema se ha realizado una simplificación. Dada la dificultad de obtener conexión ADSL (con todos los equipos que ello conlleva), se ha realizado la maqueta de una misma red con conexiones Ethernet.

8.2.2 CONFIGURACIÓN DE LA MAQUETA

En esta sección se presenta una vista general de la configuración de los diferentes dispositivos de la maqueta, la configuración de los actores del sistema (sistemas externos) y las instancias de los agentes que se ejecutan en los diferentes dispositivos. Si se desea un mayor detalle sobre esta información, consultar el "Anexo V: Ficheros de configuración de la maqueta de pruebas".

8.2.2.1 CONFIGURACIÓN DE DISPOSITIVOS

La configuración más importante a tener en cuenta por parte de los dispositivos es la ausencia de NAT en las pasarelas residenciales, dado que la aplicación de compartición de contenidos multimedia presenta problemas al atravesar un dispositivo con esta capacidad de enmascarar direcciones IP.

De este modo, las pasarelas residenciales presentan una configuración de "Puente de red" permitiendo así la interconexión de redes locales.

8.2.2.2 CONFIGURACIÓN DE ACTORES

Los actores del sistema que a su vez son sistemas externos están implementados como componentes (*bundles*) de la plataforma de servicios OSGi. Como tales, estos sistemas están configurados en la maqueta apropiadamente mediante sus diferentes ficheros de configuración.

8.2.2.3 CONFIGURACIÓN DE AGENTES

Dentro de la maqueta se han configurado una serie de actores que diagnostican fallos en un servicio de compartición de recursos multimedia entre dos usuarios.

A continuación se muestran diversas tablas en las que se pueden observar las diferentes instancias de agentes que se ejecutan en cada dispositivo.

Pasarela residencial de cliente	Descripción
Agente de interfaz OSGi	Agente encargado de interactuar con diferentes actores del sistema. Sirve de interfaz con la plataforma de servicios OSGi a través del servicio de eventos.
Agente de diagnóstico	Agente de diagnóstico encargado de dirigir el proceso.
Agente de conectividad de cliente	Agente de observación que realiza diferentes pruebas bajo petición. Realiza pruebas sobre la conectividad del escenario.

TABLA 48: AGENTES EN LA PASARELA RESIDENCIAL DE CLIENTE

Pasarela residencial de proveedor	Descripción
Agente de creencia de proveedor	Agente de creencia en el que se delega parte del diagnóstico en la parte del proveedor de contenidos multimedia.
Agente de conectividad de proveedor	Agente de observación que realiza diferentes pruebas bajo petición. Realiza pruebas sobre la conectividad del escenario.
Agente de búsqueda de servicio	Agente de observación que realiza una prueba sobre la configuración del servicio, los recursos compartidos y los permisos de los usuarios.
Agente de acceso local al servidor	Agente de observación que realiza una prueba local sobre el servidor multimedia para comprobar su estado.

TABLA 49: AGENTES EN LA PASARELA RESIDENCIAL DE PROVEEDOR

Servidor de soporte de operación	Descripción
Agente de creencia de ISP	Agente de creencia en el que se delega parte del diagnóstico en la parte de la operadora de servicio.
Agente de conectividad de ISP	Agente de observación que realiza diferentes pruebas bajo petición. Realiza pruebas sobre la conectividad del escenario.
Agente de conocimiento	Agente de conocimiento que ofrece la capacidad de desplegar conocimiento a los diferentes agentes.

TABLA 50: AGENTES EN EL SERVIDOR DE SOPORTE DE OPERACIÓN

8.3 CASOS DE PRUEBA

En esta sección se muestran las pruebas realizadas sobre el sistema para intentar asegurar el uso.

En primera instancias se presenta el plan de pruebas que se ha realizado sobre el sistema y, en segunda, un análisis sobre los requisitos para revisar si se han conseguido y qué grado se ha hecho.

En las pruebas de diseño se sigue el modelo de diseño en V. Este modelo promueve la práctica de probar, en primer lugar, los módulos de manera independiente; en segundo, la integración entre los diferentes módulos; en tercero, el comportamiento del sistema completo y, por último, la aceptación del sistema por los actores del sistema.

En el "Anexo V: Ficheros de configuración de la maqueta de pruebas" se adjunta un listado de los de los ficheros de configuración concretos de la maqueta de pruebas con la que se han realizado las siguiente pruebas.

8.3.1 PRUEBAS DE DISEÑO

8.3.1.1 PRUEBAS UNITARIAS

Las pruebas unitarias realizadas han probado que diferentes módulos comunes para ciertos agentes presentan la funcionalidad esperada. Los módulos probados han sido los siguientes:

Módulo	Prueba	Resultado
Gestión de conocimiento	Se ha probado que presenta el comportamiento esperado con respecto a la interacción de los agentes con la ontología.	Admitido
Acciones de los agentes	Se ha probado que se gestionan correctamente las diferentes acciones que puede realizar un agente.	Admitido
Gestión de eventos OSGi	Se ha probado que se manejan apropiadamente los eventos procedentes de la plataforma OSGi.	Admitido
Gestión de la red bayesiana	Se ha probado que se realiza inferencia y se manejan los diferentes eventos de adición o sustracción de evidencias correctamente.	Admitido
Pruebas realizadas por los agentes	Se ha probado el comportamiento de las pruebas que los diferentes agentes realizan durante un diagnóstico para validar el resultado.	Admitido

TABLA 51: PRUEBAS UNITARIAS

En esta fase de pruebas unitarias se podrían haber probado los diferentes agentes a través de diferentes agentes de prueba, pero esto habría conllevado un esfuerzo adicional ya que, aunque estos agentes no habrían tenido la misma funcionalidad que los agentes finales, deberían manejar diferentes tipos de mensajes.

Finalmente, se ha probado la interacción entre los agentes con los agentes definitivos como pruebas de integración y de sistema con la finalidad de ahorrar esfuerzo en la generación de las pruebas. La interacción entre agentes, aún sin ser explícita, se prueba con profundidad en la mayoría de las pruebas posteriores.

8.3.1.2 PRUEBAS DE INTEGRACIÓN

Las pruebas de integración realizadas han constado de la interacción de los agentes entre sí para comprobar que intercambian los diferentes mensajes adecuadamente.

Prueba	Descripción	Resultado
Integración de agentes que intervienen en el proceso de diagnóstico	Para ello se han usado diferentes *bundles* OSGi de prueba que iniciaban diagnósticos manualmente para comprobar que se desencadena el diagnóstico correctamente.	Admitido
Integración de agentes que intervienen en el despliegue de nuevo conocimiento	Para ello se ha desplegado un nuevo fichero de conocimiento en el agente de conocimiento.	Admitido

TABLA 52: PRUEBAS DE INTEGRACIÓN

En estas pruebas además se han probado la integración de los módulos mencionados en la sección anterior que han pasado sus pruebas unitarias con los agentes que los usan durante las pruebas realizadas.

8.3.1.3 PRUEBAS DE SISTEMA

Las pruebas del sistema han consistido en simular los sistemas externos con los que el sistema debe interactuar y probar la funcionalidad del sistema de diagnóstico completo. Para ello, se han construido componentes de prueba de la plataforma OSGi que simulan los actores del sistema (ver Tabla 6: Diccionario de actores).

Componente	Descripción
DAL	Genera eventos en la plataforma OSGi con el mismo formato que los debe generar el actor real para informar de un fallo hipotético.
MNG	Permite interactuar con una información limitada sobre recursos compartidos de prueba.
REP	Escucha eventos de informe de diagnóstico generados por el sistema de

	diagnóstico para comprobar que el formato es correcto.

TABLA 53: ACTORES DE PRUEBA

Tras enumerar los componentes de prueba que se han implementado, en la siguiente tabla se muestran las pruebas realizadas.

Prueba	Descripción	Resultado
Diagnóstico de fallos	Prueba múltiple que se repitió con una gran variedad de parámetros de entrada al sistema de diagnóstico y de configuraciones de la maqueta diferentes. También sirvió para mejorar el comportamiento de la red bayesiana usada para el diagnóstico.	Admitido
Tolerancia a fallos durante la ejecución	Prueba múltiple consistente en detener los contenedores de la plataforma de agentes representando desconexión de la parte de la red en la que reside cada uno de ellos.	Rechazado parcialmente: Dependiendo de si se han realizado diagnósticos con anterioridad, detener el contenedor principal de la plataforma de agentes es un punto crítico del sistema que conlleva al cese de las comunicaciones entre agentes.
Tolerancia a fallos en el arranque	Prueba múltiple similar a la anterior pero en el arranque del sistema.	Rechazado parcialmente: Si no se arrancan ciertos agentes, el diagnóstico se ejecuta sin ellos, lo que lleva a un diagnóstico menos fiable; pero mientras que el contenedor principal esté activo, el sistema sigue activo.
Inicio de contenedores en orden alterno	Prueba que arranca los contenedores de las pasarelas residenciales antes del contenedor principal en el servidor de soporte de operación.	Rechazado: Si los contenedores secundarios se inician antes que el principal, la plataforma de agentes falla; ya que no puede encontrar algunos servicios básicos para los agentes.
Reinicio de agentes durante la ejecución	Prueba múltiple que detiene contenedores (sus respectivos agentes) y los inicia de nuevo.	Rechazado parcialmente: Siempre que el contenedor que muera no sea el contenedor principal, la plataforma soporta reinicios de agentes.
Actualización de conocimiento	Prueba múltiple que despliega conocimiento a diferentes	Admitido

	agentes interesados en diferente conocimiento.

TABLA 54: PRUEBAS DE SISTEMA

8.3.1.4 PRUEBAS DE ACEPTACIÓN

Las pruebas de aceptación realizadas sobre el sistema han consistido en la interacción con los actores finales del escenario, validando así que se cumplen los diferentes requisitos obtenidos del análisis del problema.

Para realizar las pruebas, se han desinstalado los componentes de prueba correspondientes a los diferentes actores y se han instalado los componentes finales progresivamente.

Además se realizan pruebas probando los diferentes casos de uso del sistema:

- Diagnóstico de un síntoma
- Actualización de conocimiento

Prueba	Descripción	Resultado
Instalación de actor: MNG	Prueba el sistema de diagnóstico cambiando componente de prueba por el final.	Admitido
Instalación de actor: DAL	Prueba el sistema de diagnóstico cambiando componente de prueba por el final configurándolo debidamente con el cliente de recursos multimedia presente en la maqueta de pruebas.	Admitido
Instalación de actor: REP	Prueba el sistema de diagnóstico cambiando componente de prueba por el final.	Admitido
Diagnóstico de fallos	Prueba múltiple del sistema de diagnóstico con todos los actores del sistema instalados y diferentes situaciones.	Admitido
Actualización de conocimiento	Prueba múltiple que despliega conocimiento a diferentes agentes interesados en diferente conocimiento con todos los	Admitido

actores del sistema instalados.

TABLA 55: PRUEBAS DE ACEPTACIÓN

8.3.2 PRUEBAS DE REQUISITOS

En esta sección se presenta un resumen del estado de los requisitos y en qué medida se ha logrado cumplir dicho requisito. Si se desea consultar una descripción más detallada de los requisitos u otros datos, véase la sección 5.5.

8.3.2.1 REQUISITOS FUNCIONALES

Identificador	Nombre	Prioridad	Resultado	Comentario
RF-1	Determinación de la causa de fallo de servicio	Esencial	Logrado	El sistema determina la causa más probable modelada en la red bayesiana.
RF-2	Inicio automático de diagnóstico	Esencial	Logrado	El inicio del diagnóstico es automático ya que ninguna persona debe lanzarlo.
RF-3	Causas posibles	Alta	Logrado	Se diagnostican las causas referenciadas en el requisito.
RF-4	Informe de diagnóstico	Esencial	Logrado	Se envía el informe de diagnóstico al interesado.
RF-5	Acceso al estado de los diferentes dispositivos	Alta	Logrado de modo parcial	Se accede a algunos de los dispositivos, no a todos los presentes en la maqueta.
RF-6	Actualización de conocimiento	Esencial	Logrado	Se actualiza correctamente el conocimiento de un agente.
RF-7	Despliegue del conocimiento en un solo punto del sistema	Media	Logrado	Sólo se debe desplegar el nuevo conocimiento en el agente de conocimiento, el resto del proceso es automático.
RF-8	Despliegue de conocimiento transparente	Media	Logrado	El despliegue del conocimiento es transparente para el administrador. No tiene que decir a qué agente va dirigido el conocimiento.

TABLA 56: PRUEBAS DE REQUISITOS FUNCIONALES

8.3.2.2 REQUISITOS NO FUNCIONALES

Identificador	Nombre	Prioridad	Resultado	Comentario
RNF-1	Escalabilidad	Alta	Logrado	El sistema delega el diagnóstico en diferentes puntos para distribuir la carga del diagnóstico.
RNF-2	Estabilidad	Alta	Logrado	Las pruebas realizadas han demostrado que el sistema es estable durante periodos largos de tiempo.
RNF-3	Sistema descentralizado	Alta	Logrado	El sistema está distribuido en los diferentes contenedores y sus agentes que residen en máquinas diferentes.
RNF-4	Sistema ligero	Media	Logrado de modo parcial	Los agentes necesitan un entorno de ejecución JAVA que no soportan muchos dispositivos con recursos limitados.
RF-5	Sistema de fácil despliegue	Alta	Logrado	Gracias al uso de la plataforma de servicios OSGi, el despliegue del sistema en las redes de usuario final es extremadamente sencillo.
RF-6	Sistema de fácil gestión	Media	Logrado	Gracias al uso de la plataforma de servicios OSGi, la gestión del sistema es sencilla y manejable.
RF-7	Rendimiento	Alta	Logrado	En las pruebas realizadas, los diagnósticos cumplen esta restricción de tiempo. No obstante, dependiendo de la complejidad del diagnóstico y de las pruebas realizadas, el tiempo puede variar.

TABLA 57: PRUEBAS DE REQUISITOS NO FUNCIONALES

9. MANUAL DEL DESARROLLADOR

9.1 INTRODUCCIÓN

Este capítulo tiene la finalidad de presentar el proyecto de cara a un desarrollador que continúe con el trabajo realizado hasta el momento.

Así, este manual pretende ser un punto de referencia inicial y de consulta en futuras iteraciones en las que nuevos desarrolladores se unan al proyecto con el consecuente desconocimiento del sistema de diagnóstico multi-agente presente en el proyecto.

En una de las siguientes secciones se presentan la nomenclatura y las directivas de diseño que se han seguido en la implementación del proyecto con el objetivo de mantener el mismo estándar en los futuros desarrollos.

También se presenta en otra sección la organización del código fuente para facilitar la interacción con éste. Se describe la jerarquía de paquetes y lo que representa cada uno de ellos, la información que contiene y las diferentes funciones que contiene.

9.2 NOMENCLATURA

En esta sección se presenta la nomenclatura seguida en el código en los rasgos más importantes desde el punto de vista del desarrollador para facilitar la compresión de éste.

En la siguiente lista se presentan los puntos más importantes a tener en cuenta cuando se desea observar el código:

- El nombre de todas las clases que implementan un agente del sistema se compone del nombre de éste seguido de la palabra "*Agent*", por ejemplo: *DiagnosisAgent, BeliefAgent, ObservationAgent*, etc.
- El nombre de los interfaces del sistema constan del nombre de dicho interfaz seguido de la palabra "*Interface*" o tienen el nombre de un adjetivo: *DiagnosisInterface, Notifable*, etc.
- El nombre de todas las clases que implementan tipos de excepciones está compuesto por el nombre de la excepción seguido de la palabra "*Exception*", por ejemplo: *GenieParserException, UnexpectedMessageException*, etc.

9.3 ORGANIZACIÓN DEL CÓDIGO FUENTE

Para facilitar la familiarización con el código fuente y la organización interna del proyecto, esta sección presenta una descripción de los directorios y paquetes que se usan en el proyecto.

Para comenzar, se debe tener en cuenta que el proyecto se estructura como dos proyectos software. El primero de ellos contiene la parte común que el sistema de diagnóstico tendría para ser adaptado a otro escenario y el segundo contiene las particularidades necesarias del escenario presentado.

9.3.1 DIAGNÓSTICO GENÉRICO

El primer proyecto software, llamado "Kowgar", contiene un archivo ANT (build.xml) que facilita la compilación del código y la generación de la librería correspondiente que importa el proyecto específico del escenario. Además contiene los siguientes directorios:

- **"lib":** carpeta en la que residen todas las librerías necesarias y la generada por el fichero ANT.
- **"src":** carpeta en la que reside el código fuente.

En la raíz de la carpeta de código fuente, existen dos paquetes con los siguientes contenidos:

- **"dat.ontology":** donde residen los ficheros de la ontología generados por Protégé para poder ser fácilmente editada.
- **"kowgar":** donde reside todo el código de los agentes, procesos de diagnóstico, etc.

Dentro del paquete "kowgar", se encuentra la parte más importante y compleja del código del proyecto. Para facilitar la lectura se presenta a continuación una lista con sublistas a su vez para presentar los subpaquetes.

- **"agent":** paquete en el que reside todo el código de los agentes, como definiciones comunes de constantes o los diferentes mecanismos que usan para suscribirse a diferentes servicios, configurar sus trazas o registrarse correctamente.
    - **"common":** paquete en el que se encuentra código usado por todos los agentes.
        - **"logging":** paquete que contiene el código necesario para configurar que las trazas de cada agente aparezcan en un fichero diferente para facilitar su manejo.
        - **"registration":** paquete que contiene el código para registrar correctamente los servicios que los agentes ofrecen o buscan.
        - **"topicSubscription":** paquete que contiene el código del servicio de suscripción a diferentes áreas de interés.
            - **"client":** paquete con el código cliente de este servicio.
            - **"server":** paquete con el código servidor de este servicio.
    - **"diagnosisAgent":** paquete con el código del agente de diagnóstico.
    - **"beliefAgent":** paquete con el código del agente de creencia.
    - **"observationAgent":** paquete con el código de un agente de observación genérico, es decir, no tiene ninguna prueba en su código.
    - **"knowledgeAgent":** paquete con el código del agente de conocimiento.
- **"behaviour":** paquete con el código de los diferentes comportamientos de los agentes.
    - **"notifier":** paquete con el código que permite a los agentes recibir nuevo conocimiento a través del mecanismo de suscripción a diferentes áreas de interés.
    - **"timer":** paquete que contiene el código de los diferentes temporizadores que usan los agentes para descatalogar observaciones antiguas o gestionar diagnósticos antiguos.
    - **"diagnosis":** paquete que contiene todo el código sobre el diagnóstico.

        - **"procedure":** paquete que contiene el código del proceso de diagnóstico, es decir, todo el proceso de solicitar y añadir observaciones o creencias, realizar inferencia bayesiana, etc.
        - **"actions":** paquete que gestiona las diferentes acciones que el agente puede llevar a cabo.
- **"knowledge":** paquete que contiene todo el código necesario para interactuar con el conocimiento que intercambian los agentes entre sí.
    - **"jade.ontology":** paquete con el código generado automáticamente por el *plug-in* de Protégé que contiene todos los beans para crear instancias de la ontología.
    - **"ontology":** paquete que contiene el código necesario para interactuar con las diferentes instancias de la ontología, parsear mensajes de entrada y salida, almacenar operaciones y traducir ficheros de conocimiento en formato XDSL a instancias de la ontología.
        - **"beans":** paquete que contiene diferentes clases de ayuda para la traducción de los ficheros de conocimiento de formato XDSL.
    - **"bayes":** paquete que contiene el código necesario para construir las redes bayesianas, interactuar con ellas, realizar inferencias, etc.
- **"exception":** paquete que contiene las diferentes excepciones que el sistema puede lanzar en diferentes situaciones.

9.3.2 DIAGNÓSTICO ESPECÍFICO DEL ESCENARIO

Tras presentar la parte común del código para otros posibles sistemas de diagnóstico, se presenta el proyecto que contiene la parte específica del escenario de diagnóstico de un servicio de compartición de recursos multimedia. Este proyecto software lleva el nombre de "Magneto".

Este proyecto también contiene un fichero ANT (build.xml) que facilita la generación de los diferentes elementos necesarios para desplegar el software correctamente (*bundles* OSGi, librerías, etc.). La organización de este proyecto es algo más compleja, ya que contiene el código necesario para generar los diferentes *bundles* OSGi que se requieren en el sistema. El sistema de directorios se muestra a continuación con el mismo formato que en el otro proyecto, presentando sublistas para los subpaquetes:

- **"lib":** directorio en el que residen las diferentes librerías necesarias y la generada por el fichero ANT.
- **"felix-dir":** directorio en el que residen los diferentes ficheros necesarios en el directorio de la plataforma OSGi, Apache Felix.
    - **"cfg":** directorio en el que residen los ficheros de configuración que deben estar en el directorio del mismo nombre de la plataforma OSGi.

  - **"var":** directorio en el que residen los diferentes ficheros de conocimiento que los agentes guardan en local tras cada actualización de conocimiento.
- **"bundles":** directorio en el que reside el contenido de los diferentes *bundles* para facilitar su gestión y su construcción. Es importante remarcar que en estos directorios no reside el código fuente, si no la estructura lista para generar los *bundles* correspondientes.
  - **"handler":** directorio en el que reside la estructura del *bundle* de prueba que recibe eventos de informe de diagnóstico.
  - **"publisher":** directorio en el que reside la estructura del *bundle* de prueba que envía eventos de notificación de fallo.
  - **"jadeAgentsLoaderBundle":** directorio en el que reside la estructura del *bundle* encargado de arrancar los diferentes agentes mediante el fichero configuración.
  - **"magnetoBundle":** directorio en el que reside la estructura del *bundle* que contiene todo el código del sistema de diagnóstico.
- **"deploy":** directorio en el que reside el código listo para desplegar.
  - **"bundle":** directorio en el que reside todos los *bundles* listos para ser instalados en la plataforma OSGi.
  - **"OSS Server":** directorio en el que reside la aplicación WADE lista para ser desplegada en el servidor de soporte de operación.
- **"src":** directorio en el que reside el código fuente.

Una vez presentada la estructura del proyecto software, se presenta la organización del código fuente por paquetes.

- **"magneto":** paquete que contiene el código necesario para la particularización del sistema de diagnóstico al escenario.
  - **"agent":** paquete que contiene el código de los agentes específicos del escenario.
    - **"interfaceAgentImpl":** paquete que contiene el código del agente de interfaz que interactúa con la plataforma OSGi.
    - **"observationAgentImpl":** paquete que contiene el código de los diferentes agentes de observación que se usan en el escanario.
    - **"common":** paquete que contiene las diferentes definiciones comunes a los diferentes agentes.
  - **"osgiEventIF":** paquete que contiene el código que permite interactuar con facilidad los eventos de entrada y de salida de la plataforma OSGi.
    - **"eventParser":** paquete que contiene el código que parsea con facilidad los diferentes eventos.

        - **“beans”:** paquete que contiene diferentes beans útiles en el proceso de parseo de los eventos.
- **“dat”:** paquete que contiene diferentes datos sobre el conocimiento que se despliega sobre los agentes.
    - **“bayes”:** paquete que contiene los ficheros XDSL.
    - **“ontology”:** paquete que contiene los ficheros SL, con instancias de la ontología (obtenidas tras parsear los diferentes archivos XDSL).
- **“bundle”:** paquete que contiene el código de los *bundles* que no tienen una relación con el sistema de diagnóstico.
    - **“loader”:** paquete que contiene el código del *bundle* “jadeAgentsLoaderBundle”.
    - **“test”:** paquete que contiene el código de los diferentes *bundles* de prueba.
        - **“handler”:** paquete que contiene el código del *bundle* de prueba “handler”.
        - **“publisher”:** paquete que contiene el código del *bundle* de prueba “publisher”.

10. CONCLUSIONES Y TRABAJO FUTURO

10.1 INTRODUCCIÓN

En este capítulo se presentan las conclusiones alcanzadas tras su desarrollo y los trabajos futuros que podrían llevarse a cabo para mejorar el sistema y añadir nuevas funcionales de interés en ámbito general aunque incluyendo una pequeña descripción de los puntos más importantes para conceptualizar mejor las características potenciales a añadir.

10.2 CONCLUSIONES

Tras el desarrollo del proyecto, se ha alcanzado el objetivo propuesto a su inicio: el desarrollo de un sistema de diagnóstico distribuido usando redes bayesianas. Durante dicho desarrollo se han presentado diferentes aspectos a tener en cuenta, la mayoría de ellos se muestran en el siguiente listado:

- Las redes bayesianas son una potente herramienta que ofrece muchas posibilidades en casi cualquier entorno de aplicación. Esta herramienta permite modelar cualquier proceso mental que se componga de relaciones causales, incluso, con diagramas de influencia, puede llegarse a tomar decisiones de manera más intuitiva.
- El paradigma de programación orientada a agentes ofrece un amplio abanico de posibilidades para diseñar sistemas distribuidos. La modularidad que reside intrínsecamente en el concepto de los agentes es una ventaja en dichas arquitecturas.
- El uso de metodologías ágiles para el desarrollo de software permite orientar el esfuerzo a cuestiones claves del software mientras que otros aspectos pueden dejarse un poco más relegados al no ser urgentes y/o importantes. La carencia de tiempo puede llevar a, prácticamente, abandonar ciertos puntos que deberían llevarse a cabo con anterioridad.
- Las plataformas de código abierto que se encuentran en la comunidad son de gran utilidad en el desarrollo de pruebas conceptuales de sistemas. No obstante, si se desea tener un software en producción, muchas de estas herramientas tienen ciertas restricciones que puede obstaculizar algunas características del producto.
- Los diferentes lenguajes en los que se pueden expresar las ontologías, como OWL, ofrecen potentes métodos de inferencia sobre el conocimiento. No obstante, en sistemas restrictivos con respecto al rendimiento y con un intercambio de mensajes moderado puede sobrecargar el requisito de capacidad de proceso en ciertos puntos del sistema.
- La tecnología OSGi y el conjunto de especificaciones de las que está compuesta ofrecen un marco muy útil con respecto a la gestión del ciclo de vida del software y de la integración entre diferentes componentes.

10.3 TRABAJO FUTURO

En esta sección se muestran las líneas de desarrollo e investigación que pueden seguirse para mejorar el comportamiento y las características del sistema. En las siguientes subsecciones se muestra cada una de los diferentes aspectos a abordar con una breve explicación.

10.3.1 DESCUBRIMIENTO DE AGENTES POR REGIÓN

Esta característica permitiría el diagnóstico de regiones concretas. Esto es necesario cuando hay diferentes regiones con las mismas características dentro del escenario.

Por ejemplo, en el caso de estudio, se podría diagnosticar fallos en sesiones con múltiples usuarios proveedores o clientes. Así cuando un usuario tiene un fallo en una sesión con un proveedor concreto, el sistema debe diagnosticar problemas en la red de ese proveedor.

Para ello se necesita un identificador de red de área del hogar (o HAN, Home Area Network) que debe ser proporcionado por el sistema de gestión de recursos multimedia compartidos (ver Tabla 6: Diccionario de actores). Gracias a este identificador, los agentes pueden buscar dinámicamente a los agentes adecuados para buscar diagnosticar un caso concreto de fallo.

10.3.2 ALMACENAMIENTO DE DIAGNÓSTICOS EN BASE DE DATOS

Esta característica del sistema es necesaria para realizar un aprendizaje válido y eficiente sobre la red bayesiana usada en el sistema.

Obviamente, esta función requiere del diseño de una base de datos acorde con los datos manejados en una operación de diagnóstico del sistema.

Un posible enfoque del diseño de ésta es añadiendo unas nuevas acciones a la ontología, como "*StoreDiagnosis*" si se desea guardar los diagnósticos de uno en uno o "*StoreDiagnoses*" si se desean guardar en bloques. Para procesar estas acciones, un agente encargado de la interacción con la base de datos puede exportar tal servicio y recibir peticiones de almacenamiento de diagnósticos.

Mientras que la opción de guardar los diagnósticos uno por uno es más simple, la opción de guardar los diagnósticos en bloques sería algo más exigente en cuanto a requisitos del agente y reduciría la mensajería necesaria, aumentando así la escalabilidad del sistema.

10.3.3 VALIDACIÓN DE DIAGNÓSTICOS PASADOS

Una vez se tengan los diagnósticos almacenados en la base de datos correspondiente y como paso previo al aprendizaje, se deben validar los diagnóstico correctos y corregir los diagnóstico que hayan sido incorrectos (hayan dado un hipótesis de causa de fallo falsa).

Este proceso podría ser automático en la mayoría de los casos si el sistema también tuviera la característica de reparación automática de fallos, así el sistema diagnosticaría una causa de fallo, intentaría recuperarse y, tras ello, se evalúa de nuevo el estado del sistema diagnosticado; si el sistema ha alcanzado un estado válido,

el diagnóstico fue correcto; en el caso contrario, se intenta resolver la siguiente causa de fallo más probable y así sucesivamente hasta encontrar el correcto.

En el caso de que el sistema no tenga esta característica, esta validación debería realizarse de manera manual por un experto en el sistema o servicio diagnosticado.

10.3.4 APRENDIZAJE AUTOMÁTICO

Con las dos características anteriores integradas en el sistema, el sistema podría realizar un proceso de aprendizaje con los datos almacenados en la base de datos como correctos o incorrectos (siempre que se remarque que son incorrectos).

Este proceso de aprendizaje generaría una nueva red bayesiana que sería desplegada en el agente interesado en ella mediante el mecanismo que el sistema ya ofrece mediante el agente de conocimiento (ver secciones 5.4.3 y 6.3.2.5 para saber más sobre el proceso de actualización de conocimiento).

10.3.5 RECUPERACIÓN AUTOMÁTICA DEL SISTEMA DIAGNOSTICADO

Una vez diagnosticada una causa de fallo en el sistema o servicio, podrían existir diferentes agentes de recuperación encargados de ejecutar diferentes acciones mediante reglas o políticas predefinidas que tendrían la responsabilidad de llevar de nuevo al sistema a un estado estable y válido para su operación.

10.3.6 INTERFAZ GRÁFICA DE ADMINISTRADOR

Una interfaz gráfica para el administrador del sistema puede resultar de gran utilidad para que el administrador pueda acceder a la base de datos para observar diagnósticos o validarlos, gestionar en todo momento que agentes del sistema están activos, arrancar nuevos agentes, solicitar diagnósticos en cualquier instante, etc.

Una de las formas más sencillas y útiles de acceso para el operador sería a través de una plataforma WEB que le ofreciese los servicios anteriormente comentados.

10.3.7 ALGORITMO DE INTERCAMBIO DE CREENCIAS MEJORADO

Como se muestra en el Anexo IV: Intercambio de creencias en redes bayesianas, el algoritmo de intercambio tiene actualmente ciertas restricciones en la construcción de los diferentes *clusters*, ya que no se soporta el intercambio de nodos “hijo” de una estructura en V si no residen todos los padres en el mismo *cluster*.

ANEXO I: PLATAFORMA DE AGENTES – JADE / WADE

INTRODUCCIÓN

Este capítulo presenta las características más importantes de JADE y WADE con la intención de ofrecer una visión global de la arquitectura de la plataforma.

La programación orientada a agentes (*Agent-Oriented Programming* /AOP) es un nuevo paradigma software relativamente nuevo que lleva conceptos de teorías de inteligencia artificial a la corriente actual de sistemas distribuidos. AOP modela esencialmente una aplicación como una colección de componentes llamados agentes que se caracterizan, entre otras cosas, por su autonomía y su capacidad para comunicarse.

Desde el punto de vista de la autonomía, estos agentes pueden manejar independientemente tareas complejas o prolongadas en el tiempo. Y, desde el punto de vista comunicativo, ellos pueden interactuar con otras entidades para ayudar a conseguir sus propias metas o metas de otros.

El modelo arquitectónico de una aplicación orientada a agentes es, de manera intrínseca, "peer-to-peer"; ya que cualquier agente es capaz de iniciar una comunicación con cualquier otro agente o ser receptor de una comunicación entrante originada por otro agente en cualquier momento.

PLATAFORMA DE AGENTES JADE

VISIÓN GLOBAL

JADE es una plataforma de agentes completamente distribuida con una infraestructura flexible que permite una fácil extensión a través de módulos. Este *framework* facilita el desarrollo de aplicaciones completamente basadas en agentes, es decir, de un entorno de ejecución que implementa las características del ciclo de vida que requieren los agentes.

ARQUITECTURA DE JADE

Una plataforma JADE está compuesta por contenedores de agentes que pueden estar distribuidos sobre la red. Los agentes viven en los contenedores los cuales son procesos JAVA que provee el entorno JADE y los servicios necesarios para alojar y ejecutar a los diferentes agentes. Hay que puntualizar que la plataforma tiene un contenedor especial, llamado "Contenedor principal", que representa el punto de autosuficiencia de la plataforma: es el primer contenedor que es iniciado y el resto de contenedores deben afiliarse a éste registrándose en él.

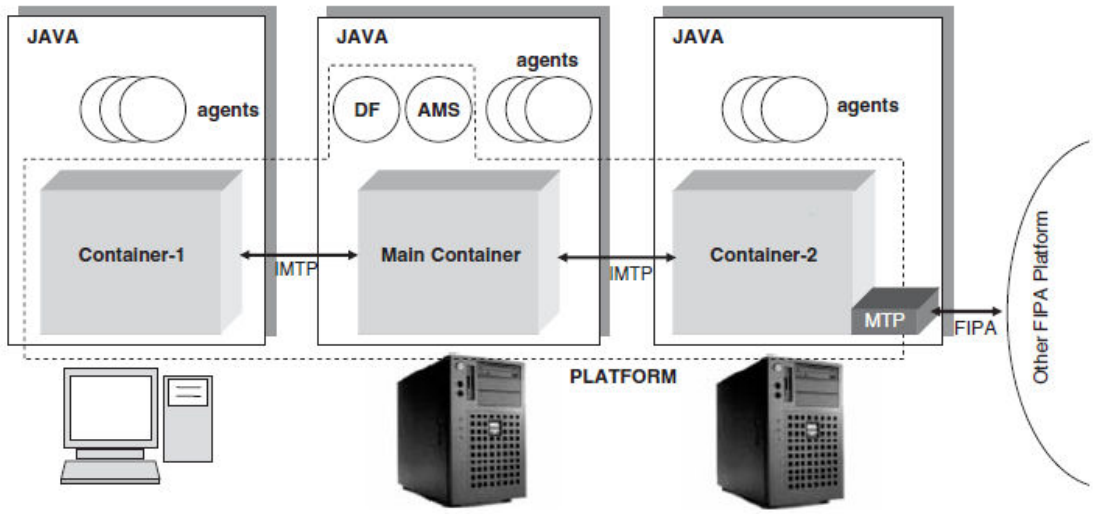

ILUSTRACIÓN 36: ARQUITECTURA DE PLATAFORMA JADE

Cuando se arranca un contenedor principal, dos agentes especiales son iniciados automáticamente. La función de estos agentes viene definida por el estándar FIPA de gestión de agentes. Estos agentes son:

- El *Agent Management System* (AMS o Sistema de Gestión de Agentes) es el agente que supervisa la plataforma global. Es el punto de contacto para todos los agentes que necesitan acceder a las "páginas blancas" de la plataforma para gestionar su ciclo de vida. Se requiere que cada agente se registre en el AMS para obtener un AID (*Agent Identifier*) válido.
- El *Directory Facilitator* (DF o Coordinador de directorio) es el agente que implementa el servicio de páginas amarillas. Este servicio es usado por cualquier agente que desee registrar sus servicios o buscar servicios disponibles ofrecidos por otros agentes. Este agente también acepta suscripciones de agentes que desean ser notificados cada vez que se produzca un registro o una modificación de servicio que coincida con algún criterio específico.

Además de estas, de estas características, la plataforma ofrece otros servicios interesantes como es el de comunicación externa con otras plataformas FIPA a través del protocolo MTP (*Message Transport Protocol*).

SOPORTE OSGI DE LA PLATAFORMA JADE

La plataforma JADE ofrece un *bundle* que se puede desplegar dentro de un entorno OSGi para ofrecer un nuevo servicio de creación de agentes. Esta distribución ofrece las mismas características de la plataforma de agentes JADE.

Se pueden iniciar contenedores (principales o no) con cualquier otro servicio de la plataforma.

BUNDLE JADE-OSGI

El *bundle* JADE-OSGi incluye un "*BundleActivator*", cumpliendo así la especificación OSGi que activa un contenedor JADE. Así, instalar JADE en un entorno OSGi es tan simple como instalar el *bundle* JADE-OSGi en la plataforma OSGi e iniciarlo.

Como en la mayoría de los casos, los contenedores necesitan unas opciones de configuración específicas, como por ejemplo si va a ser un contenedor principal y, si no lo va a ser, dónde reside dicho contenedor.

Todas estas opciones pueden ser especificadas en las propiedades del entorno OSGi cuyos nombres deben ser los mismos que en las opciones normales de configuración

de JADE, pero con el prefijo “jade.”; por ejemplo, “jade.main=true” para indicar si un contenedor debe ser contenedor principal.

Estas propiedades del entorno OSGi normalmente suelen estar especificadas en ficheros de configuración, no obstante, esto depende de la plataforma específica que se utilice.

Además de iniciar un contenedor, este *bundle* registra un servicio llamado *JadeRuntimeService* (JRS) en el registro de servicios del entorno OSGi. Este servicio puede ser usado por otros *bundles* dentro de la plataforma para crear agentes y controlar el ciclo de vida del contenedor local de agentes.

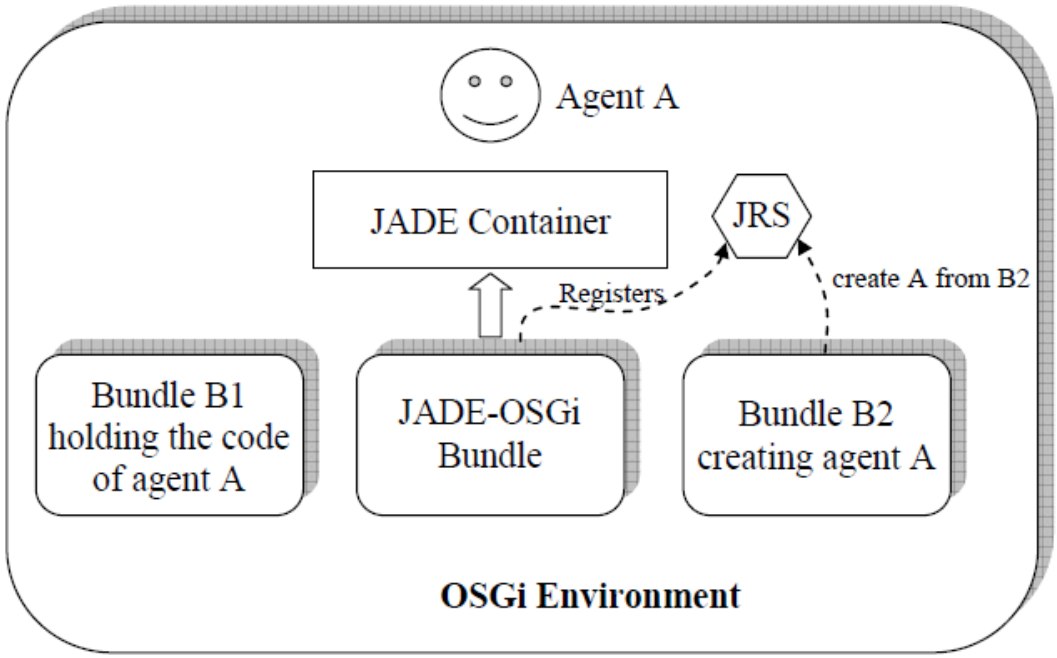

ILUSTRACIÓN 37: JADE EN ENTORNO OSGI

APLICACIÓN WADE

WADE (*Workflows and Agents Development Environment*) es una plataforma independiente construida sobre JADE. WADE añade a JADE el soporte para ejecutar tareas definidas siguiendo la metáfora de flujos de trabajo y una serie de mecanismos que ayudan a la gestión de la complejidad de un sistema distribuido con respecto a la administración y la gestión de fallos.

Los flujos de trabajo y la gestión distribuida son los dos elementos principales de la aplicación y, por lo tanto, son los puntos claves en la decisión de cómo WADE se adapta a las necesidades de la aplicación.

GESTIÓN DISTRIBUIDA DE AGENTES CON WADE

La distribución suele ser considerada una característica muy útil. Pero es evidente que la administración de un sistema distribuido es más compleja que en un sistema centralizado a menos que se dispongan de las herramientas apropiadas. Además, la probabilidad de fallos en dispositivos aumenta proporcionalmente con el número de equipos en los que el sistema está distribuido y, por lo tanto, se necesitan los mecanismos de recuperación apropiados para asegurar que la aplicación es tolerante a diferentes fallos en los dispositivos. Los sistemas de *clustering* suelen ser la mejor opción para estos propósitos, pero estos sistemas suelen ser bastante caros y difíciles de gestionar y configurar.

WADE planta cara a estos problemas ofreciendo diferentes mecanismos que facilitan la administración del sistema.

- Configurar fácilmente la aplicación, es decir, los diferentes contenedores y agentes.
- Activar o desactivar los componentes distribuidos de la aplicación a través de los equipos (gracias a los *Boot Daemons* y al *Configuration Agent* / CFA).
- Monitorización en tiempo de ejecución de eventos y condiciones críticas como el consumo de disco o de memoria.
- Desplegar nuevas lógicas del sistema en tiempo de ejecución sin parar el sistema.
- Recuperación automática de fallos en agentes o contenedores (gracias al *Controller Agents* / CA, uno por contenedor).

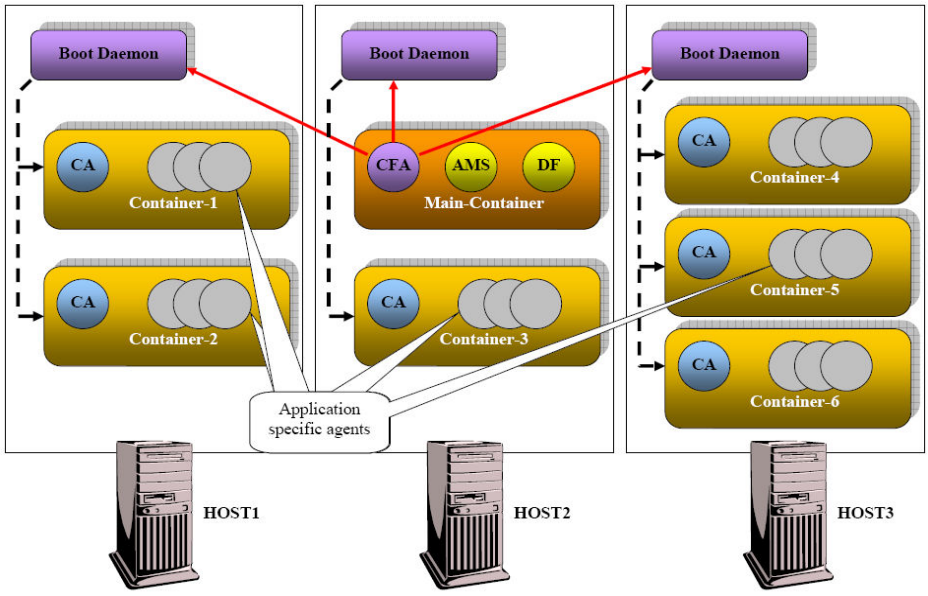

ILUSTRACIÓN 38: ARQUITECTURA WADE

ANEXO II: PLATAFORMA DE SERVICIOS OSGI – APACHE FELIX

INTRODUCCIÓN

La tecnología OSGi es un conjunto de especificaciones que define un sistema de componentes dinámicos para JAVA. Estas especificaciones permiten un modelo de desarrollo donde las aplicaciones están compuestas dinámicamente de muchos componentes reutilizables. Las especificaciones OSGi permiten a los diferentes componentes ocultar sus implementaciones a otros componentes y comunicarse a través de servicios.

OSGi es la primera tecnología que tiene éxito actualmente con un sistema de componentes que está resolviendo muchos problemas reales en el desarrollo de software. Con este enfoque de servicios, el código es más fácil de escribir y probar, la reusabilidad se incrementa, la construcción de sistemas se hace significativamente más simple, el despliegue es más gestionable, los fallos son detectados fácilmente y el entorno de ejecución ofrece una vista global de todo lo que se está ejecutando. Aún

después de todo esto, lo más importante es el modelo funciona cómo se ha probado en la gran adopción y uso en aplicaciones tan populares como Eclipse y Spring.

Como conclusión, las especificaciones OSGi proveen un modelo basado en componente maduro y completo. Así, convertir sistemas monolíticos al modelo de OSGi casi siempre ofrece grandes mejoras en el proceso completo de desarrollo de software.

MODELO DE CAPAS OSGI

La arquitectura OSGi consta de un modelo de capas que consta de los estratos presentados en la Tabla 58.

Capa	Descripción
Bundles	Los *bundles* son componentes OSGi hechos por los desarrolladores.
Servicios	La capa de servicios conecta diferentes *bundles* de manera dinámica ofreciendo un modelo de publicación-búsqueda-conexión a través de clásicos objetos JAVA.
Ciclo de vida	El API para instalar, iniciar, detener, actualizar y desinstalar *bundles*.
Módulos	La capa que define como un *bundle* puede importar o exportar código.
Seguridad	La capa que maneja los aspectos de seguridad.
Entorno de ejecución	Define qué métodos y clases están disponibles en una plataforma específica.

TABLA 58: MODELO DE CAPAS OSGI

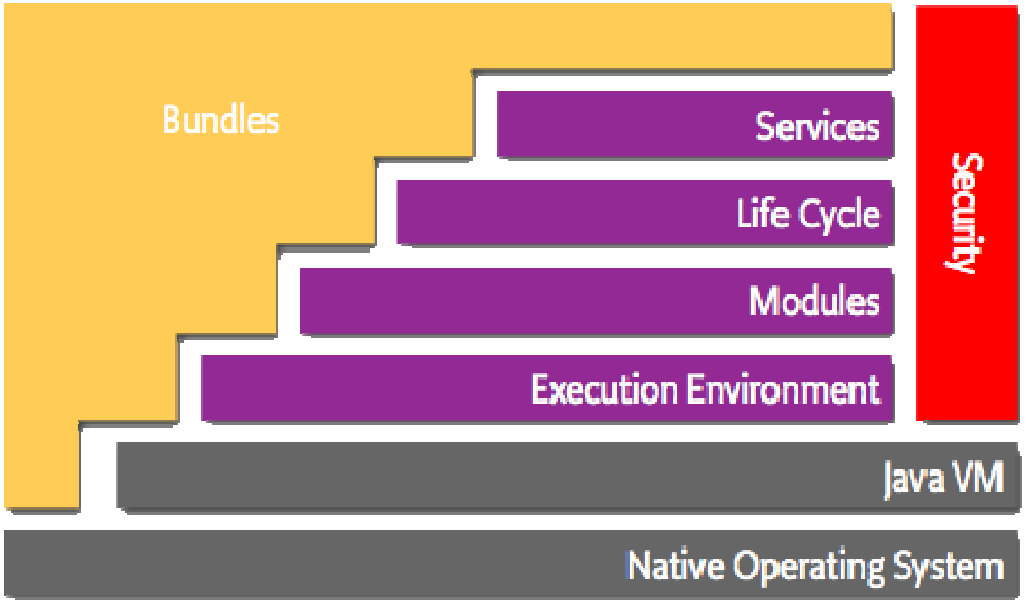

ILUSTRACIÓN 39: MODELO DE CAPAS OSGI

PLATAFORMA APACHE FELIX

En este apartado se presenta una visión general de cómo funciona esta plataforma que implementa la especificación de la plataforma de servicios OSGi *Release* 4.

INSTALACIÓN Y ARRANQUE

Para comenzar, la instalación de la plataforma es tan simple como copiar el directorio raíz del proyecto en la ruta deseada. Y tras esto, en el directorio de Apache Felix, sólo se debe ejecutar un comando para iniciar la plataforma OSGi:

```
java –jar bin/felix.jar
```

CÓDIGO 7: ARRANQUE DE FELIX

El lanzador de este entorno de trabajo inicia el entorno propiamente dicho e instala todos los *bundles* que estén dentro del directorio "*bundle*" bajo el directorio actual. Por defecto, Apache Felix tiene una simple consola de texto para interactuar con la plataforma. Para obtener otras interfaces para el administrador del sistema, existen otros *bundles* disponibles para instalar.

CONSOLA DE LA PLATAFORMA

El modo principal de interactuar con la plataforma es a través de su consola. Por defecto, la consola de Felix está implementada como un servicio OSGi que ofrece al usuario una simple interfaz en modo texto. Después de instalar la plataforma, esta interfaz puede modificarse con otros servicios que ofrecen, entre otras cosas, consolas remotas o interfaces web.

Para gestionar el ciclo de vida de los *bundles* que se quieran instalar en la plataforma, la consola ofrece diferentes comandos, entre los que se destacan los mostrados en la Tabla 59.

Comando	Argumentos	Descripción
ps	Ninguno	Ofrece un listado de los *bundles* instalados en la plataforma y el estado en el que están, por ejemplo, ejecutándose, instalados o parados tras haberse ejecutado.
install	URL o ruta del *bundle*	Instala un *bundle* dado dentro de la plataforma. Para

		ello comprueba todas las dependencias para la correcta instalación del componente.
start	ID de *bundle*	Inicia el componente seleccionado.
stop	ID de *bundle*	Detiene el componente seleccionado.
update	ID de *bundle*	Actualiza el componente seleccionado buscando una nueva versión en la misma ruta en la que se instaló éste.
uninstall	ID de *bundle*	Desinstala el componente seleccionado.

TABLA 59: COMANDOS DE GESTIÓN DE CICLO DE VIDA

Para mostrar un ejemplo y así facilitar la comprensión de estos comandos, se presenta un ejemplo de su uso dentro de la consola.

```
-> install http://myserver/example.jar
-> ps
START LEVEL 1
   ID   State         Level  Name
[   0] [Active     ] [    0] System Bundle (2.0.0)
[   1] [Active     ] [    1] Shell Service (1.4.0)
[   2] [Active     ] [    1] Shell TUI (1.4.0)
[   3] [Active     ] [    1] Bundle Repository (1.4.0)
[   4] [Installed  ] [    1] Bundle Example (1.0.0)
-> start 4
Hello from Example Bundle.
-> stop 4
Bye bye from Example Bundle.
-> update 4
-> ps
START LEVEL 1
   ID   State         Level  Name
[   0] [Active     ] [    0] System Bundle (2.0.0)
[   1] [Active     ] [    1] Shell Service (1.4.0)
[   2] [Active     ] [    1] Shell TUI (1.4.0)
[   3] [Active     ] [    1] Bundle Repository (1.4.0)
[   4] [Installed  ] [    1] Bundle Example (1.0.1)
-> start 4
Hello from Example Bundle (new version).
-> stop 4
Bye bye from the new version of Example Bundle.
-> uninstall 4
-> ps
START LEVEL 1
   ID   State         Level  Name
[   0] [Active     ] [    0] System Bundle (2.0.0)
[   1] [Active     ] [    1] Shell Service (1.4.0)
[   2] [Active     ] [    1] Shell TUI (1.4.0)
[   3] [Active     ] [    1] Bundle Repository (1.4.0)
```

CÓDIGO 8: GESTIÓN DE CICLO DE VIDA DE OSGI

CONFIGURACIÓN DE LA PLATAFORMA

La plataforma permite ser configurada a través de un fichero de propiedades. El lanzador de la plataforma Felix usa estos ficheros para configurar ciertos aspectos de su comportamiento. Lee los ficheros "conf/config.properties" y "conf/system.properties" (estos ficheros usan la sintaxis estándar de propiedades de JAVA).

Una vez configurada la plataforma se plantea la cuestión de cómo un *bundle* instalado puede tomar parámetros de configuración en tiempo de ejecución.; ya que la única opción que ofrece la plataforma es iniciar el *bundle* (comando "start") sin pasar ningún parámetro.

Este dilema se puede resolver de dos formas, añadiendo dichos parámetros en los ficheros de propiedades propios de la plataforma OSGi (Apache Felix) o usando archivos de configuración propios del *bundle* concreto y leer dichos ficheros en el arranque del *bundle*.

CREACIÓN DE BUNDLES OSGI

La creación de componentes que se ejecutan en una plataforma OSGi es tan simple como crear una clase que implemente el interfaz "*BundleActivator*" que requiere que la clase tenga dos métodos, uno con el que arrancar el componente y otro con el que pararlo.

```
package example;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;


public class Activator implements BundleActivator
{

    /**
     * Implements BundleActivator.start().
     * @param context the framework context for the bundle.
    **/
    public void start(BundleContext context)
    {
        System.out.println("Hello from Example Bundle.");
    }


    /**
     * Implements BundleActivator.stop().
     * @param context the framework context for the bundle.
    **/
    public void stop(BundleContext context)
```

```
    {
        System.out.println("Bye bye from Example Bundle.");
    }
}
```

CÓDIGO 9: EJEMPLO DE BUNDLEACTIVATOR

Además de esta clase que la plataforma usará en la gestión del ciclo de vida de este *bundle* o componente, un *bundle* debé incluir un fichero "manisfest.mf" en el que se especifican las dependencias con otros componentes o con la plataforma.

```
Bundle-Name: Service listener example
Bundle-Description: A bundle that displays messages
Bundle-Vendor: Apache Felix
Bundle-Version: 1.0.0
Bundle-Activator: example.Activator
Import-Package: org.osgi.framework
```

CÓDIGO 10: EJEMPLO DE FICHERO MANIFEST

ANEXO III: ADQUISICIÓN DE CONOCIMIENTO – GENIE & SMILE

INTRODUCCIÓN

Este anexo tiene el objetivo de mostrar la manera de generar nuevo conocimiento para el sistema de diagnóstico multi-agente desarrollado. Para ello se utiliza la herramienta Genie, que a su vez usa la librería SMILE para manejar redes bayesianas, diagramas de influencia, etc. Dicha herramienta es un entorno desarrollado para la construcción de modelos gráficos teóricos de decisión.

El objetivo del proceso de adquisición de conocimiento es generar el conocimiento necesario, en otras palabras, generar instancias de la ontología del sistema para que un agente dado conozca, entre otras cosas, qué tareas puede llevar a cabo durante un diagnóstico.

En el proceso de adquisición de conocimiento se aprovecha la capacidad, que ofrece Genie, de insertar información adicional a la red y a los diferentes nodos a través de

propiedades del modelo de datos en formato XDSL, un formato de etiquetas basado en XML.

En las siguientes secciones se presentan todos los pasos a seguir para la correcta construcción de un fichero XDSL, el cual contiene, al final del proceso, un conocimiento que se puede desplegar a los agentes a través de un agente de conocimiento.

CONFIGURACIÓN DE FICHERO DE CONOCIMIENTO

En esta sección se muestran la información que se debe insertar en el fichero XDSL además de diferentes capturas de pantalla de la interfaz gráfica de Genie, con la finalidad de aumentar la claridad de la explicación.

No obstante, debe anotarse que se podría generar este fichero de cualquier manera siempre y cuando tenga el mismo formato final. Se podría generar con un procesador de texto plano; no obstante, se descarta esta posibilidad dada la complejidad que conllevaría.

CREACIÓN DE LA RED BAYESIANA

El primer paso a seguir, y uno de los más importantes, es la creación de la red bayesiana. La interfaz gráfica de Genie proporciona un fácil manejo de los nodos y los arcos de una red bayesiana.

Gracias a las opciones de insertar nodos y arcos, la creación de la estructura de la red es muy sencilla.

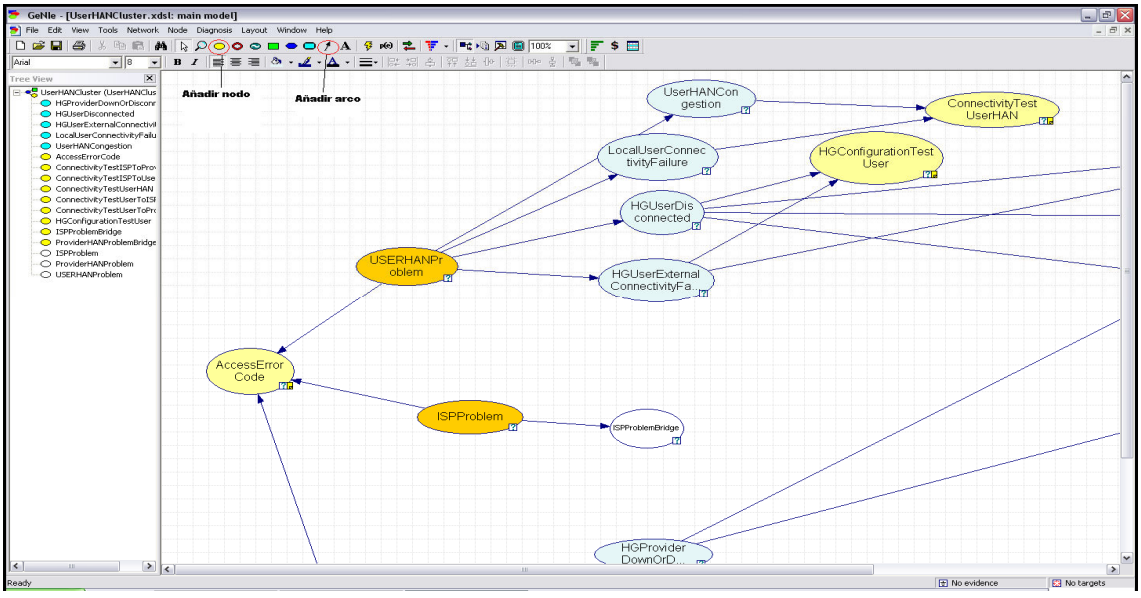

ILUSTRACIÓN 40: CAPTURA DE PANTALLA - PANTALLA PRINCIPAL

Una vez obtenido la estructura deseada, se deben insertar algunos datos más en los nodos. Para ello se utiliza la hoja de propiedades de nodo, accesible a través del menú "Node"->"Node Properties" al tener un nodo seleccionado.

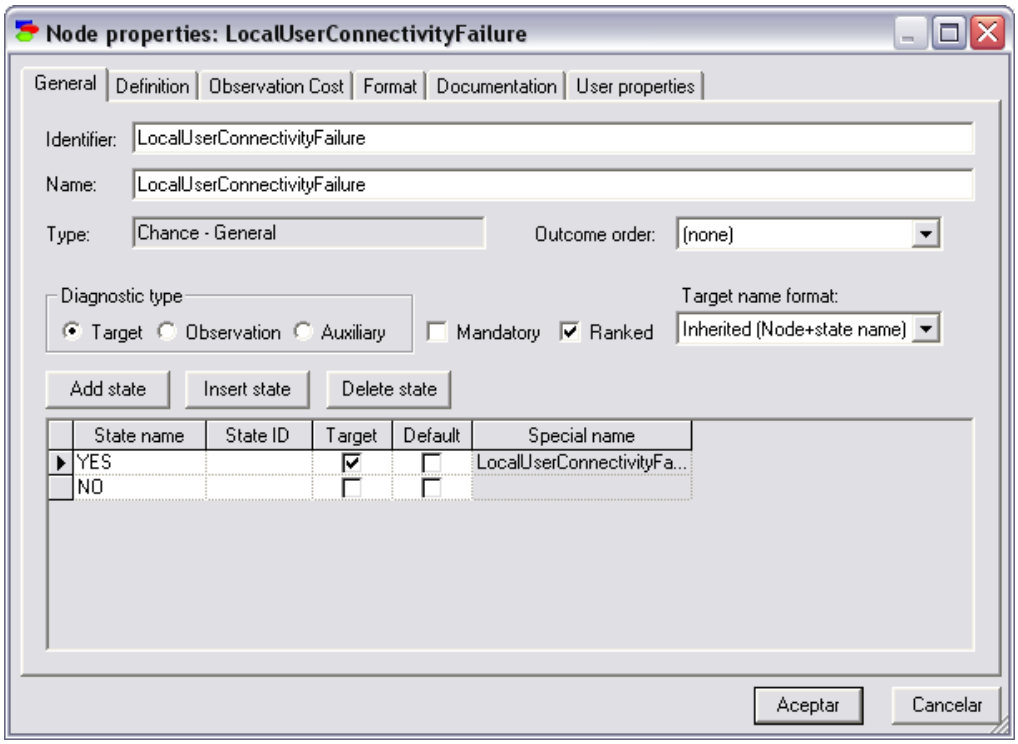

ILUSTRACIÓN 41: CAPTURA DE PANTALLA - HOJA DE PROPIEDADES DE NODO - GENERAL

En la pestaña "General" dentro de las propiedades de nodo, se debe especificar:

- su nombre y su identificador en los dos primeros campos ("*Name*" e "*Identifier*"),
- el tipo de nodo que es: hipótesis, observación o auxiliar, en el campo "*Diagnostic type*". Además, si es un nodo hipótesis, se debe marcar el estado que se quiere alcanzar como "*Target*" dentro de la tabla de estados.
- Con los botones "*Add state*" o "*Insert state*", se deben añadir todos los estados deseados para el nodo.

Es recomendable hacer esta configuración en todos los nodos antes de continuar con el siguiente paso, ya que si no podrían modificarse diferentes tablas de probabilidad y se debería repetir parte del trabajo.

En la pestaña "*Definition*", se define la tabla de probabilidad condicional del nodo en cuestión.

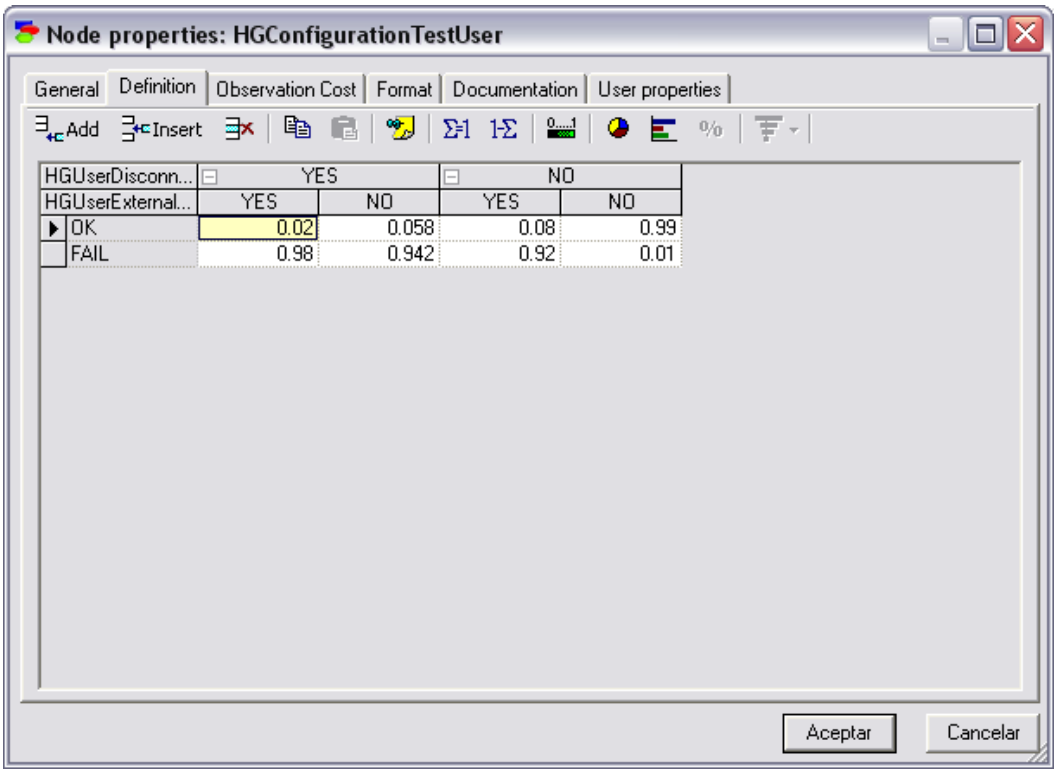

ILUSTRACIÓN 42: CAPTURA DE PANTALLA - HOJA DE PROPIEDADES DE NODO – DEFINITION

En la tabla presente en esta pestaña de la hoja, se deben insertar todos los valores correspondientes. Teniendo en cuenta que los valores de las filas son los estados del propio nodo y los valores de las columnas son los estados de los nodos padre del nodo en cuestión, se debe insertar en cada elemento de la tabla la probabilidad de que se dé el estado del nodo una vez dados los estados de esa columna en los nodos padre.

Para terminar con la creación de la red bayesiana, se deben insertar algunos parámetros más en los diferentes nodos de observación. Los parámetros necesarios para ejecutar una prueba se especifican en la pestaña "*User properties*".

Esta pestaña permite insertar información adicional mediante un sistema de propiedades con el sistema "clave-valor". Con los botones "*Add…*", "*Edit…*" y "*Delete*" se pueden añadir, editar o borrar propiedades en el nodo.

Para añadir los parámetros necesarios en la ejecución de la prueba de la que se obtendrá la observación de un nodo observación, se deben insertar una nueva propiedad con la clave "*parameters*" y, en su campo valor, la lista de parámetros que necesita el sistema separados por el signo ";".

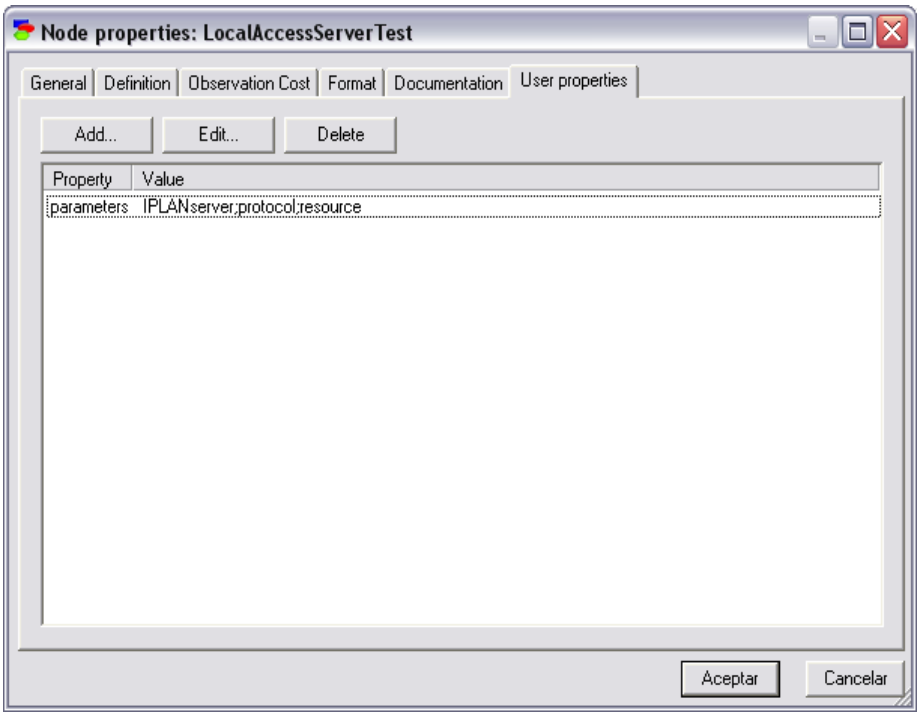

ILUSTRACIÓN 43: CAPTURA DE PANTALLA - HOJA DE PROPIEDADES DE NODO - PROPIEDADES DE USUARIO

PROPIEDADES DE LA RED BAYESIANA

Una vez se tiene la red bayesiana creada correctamente, se debe añadir cierta información concerniente a la red completa. Para ello se usa la hoja de propiedades de red a la cual se accede a través del menú "*Network*" -> "*Network Properties*".

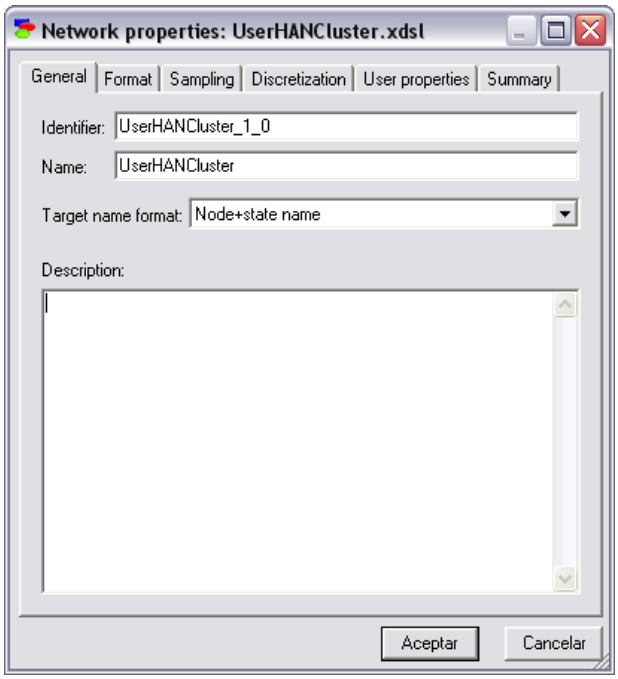

ILUSTRACIÓN 44: CAPTURA DE PANTALLA - HOJA DE PROPIEDADES DE RED – GENERAL

En la pestaña "*General*" se deben añadir:

- el nombre (en el campo "*Name*"),
- el identificador (en el campo "*Identifier*"). En este último se debe añadir también la versión de la red bayesiana.
- Además de esto, es importante fijar aquí, en el campo "*Target name format*", la opción "*Node+state name*"; ya que éste es el formato que soporta el sistema de agentes, el que se maneja en la ontología. A partir de aquí este formato se extiende a todos los nodos de la red bayesiana.

Una vez fijados estos datos, se debe proceder a la definición de las diferentes tareas que puede llevar a cabo el agente que recibirá este conocimiento dentro de un proceso de diagnóstico.

Dentro de la pestaña "*User properties*" se ofrece la misma posibilidad que en los nodos de la red, es decir, la posibilidad de insertar información adicional a través de propiedades en el formato "clave-valor".

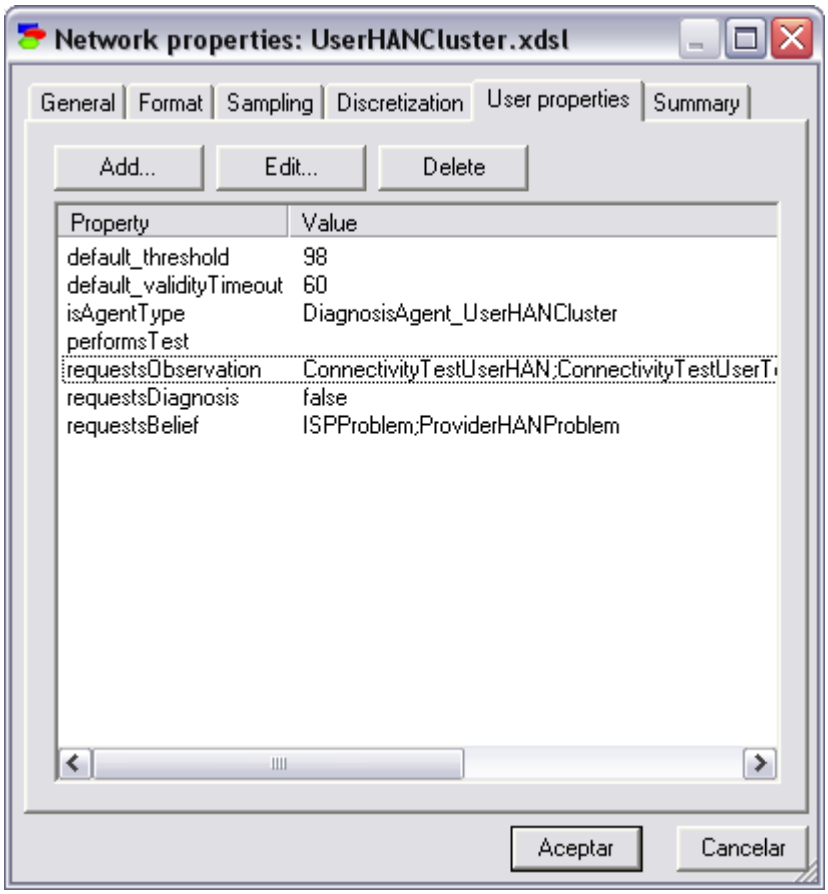

ILUSTRACIÓN 45: CAPTURA DE PANTALLA - HOJA DE PROPIEDADES DE RED - PROPIEDADES DE USUARIO

Los datos necesarios en estas propiedades son los siguientes:

- "*default_threshold*", cuyo valor debe ser el porcentaje al que se desea detener el diagnóstico. Es decir, cuando un nodo hipótesis alcanza ese valor en su estado objetivo, el diagnóstico termina ya que se ha alcanzado una conclusión válida.
- "*default_validityTimeout*", cuyo valor representa el tiempo (en segundos) que debe pasar para que una observación se considere obsoleta y no añada información correcta para el proceso de diagnóstico. Si se ha superado este tiempo desde que se realizó la prueba, la observación se descarta.
- "*isAgentType*", cuyo valor debe contener el nombre del agente y el nombre de la red (o escenario) en el formato: "*Agente*"_"*Red*".
- "*requestsDiagnosis*", cuyo valor debe ser "true" o "false" si puede o no puede solicitar diagnósticos.
- "*requestsObservation*", cuyo valor debe ser una lista de todas las observaciones que el agente puede solicitar separados con el signo ";". En el excepcionalmente de que el agente pueda solicitar todos los nodos observación de la red, puede insertarse el valor "ALL" para representar este hecho.
- "*requestsBelief*", funciona igual que la propiedad anterior pero con los nodos de creencia.
- "*performsTest*", funciona igual que los anteriores pero con las observaciones que se pueden puede realizar de manera local en el mismo agente que realiza el diagnóstico.

Con toda esta información añadida al fichero de conocimiento, éste ya está listo para ser distribuido debidamente a través del mecanismo ofrecido por el agente de conocimiento.

ANEXO IV: INTERCAMBIO DE CREENCIAS EN REDES BAYESIANAS

INTRODUCCIÓN

Este anexo tiene el objetivo de mostrar diferentes aspectos del método de intercambio de creencias que se usa en el sistema de diagnóstico, entre ellos, en qué consiste, por qué se usa y cuándo se usa.

Antes de continuar, se debe tener en cuenta que, para comprender plenamente este método, se deben tener unos conocimientos previos básicos sobre las redes bayesianas y la inferencia sobre estas.

Una red bayesiana puede llegar a tener un número elevado de nodos dependiendo del nivel de detalle que se quiera alcanzar en el conocimiento representado en la red. En este caso, puede ser muy interesante el disponer de la capacidad de distribuir la inferencia bayesiana en diferentes dispositivos. Ya sea para distribuir el peso

computacional de dicha inferencia o porque se desee realizar una parte de la inferencia en una región física o un dispositivo específico.

En el caso de una red bayesiana diseñada para el diagnóstico distribuido, como es el caso de estudio que se presenta en el proyecto, puede resultar muy interesante tener la posibilidad de realizar parte del diagnóstico en diferentes puntos del sistema a diagnosticar (suponiendo que se intenta diagnosticar un sistema distribuido, ya que carece de sentido el hecho de diagnosticar de manera distribuida un sistema monolítico o centralizado).

FUNDAMENTOS MATEMÁTICOS

En esta sección se muestran los fundamentos necesarios para el intercambio de creencias entre dos redes bayesianas. Para comenzar, se necesita presentar una serie de conceptos que deben quedar asentados para comprender el mecanismo de intercambio de creencias.

CONCEPTOS PRELIMINARES

Dentro del mecanismo de intercambio de creencias se usan una serie de conceptos que se exponen a continuación.

- Un arco es una relación entre dos nodos, padre e hijo. En la representación de una red bayesiana, el arco se representa como una flecha que nace en un nodo y muere en otro.

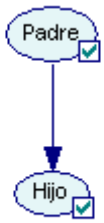

ILUSTRACIÓN 46: CONCEPTO DE ARCO

Cada rol está definido por la dirección de la flecha. El nodo padre es el nodo en el que nace la flecha y el nodo hijo es el nodo en el que muere o hacia el que apunta ésta.

- Un nodo “hijo” puede ser “padre” de otro nodo a su vez.

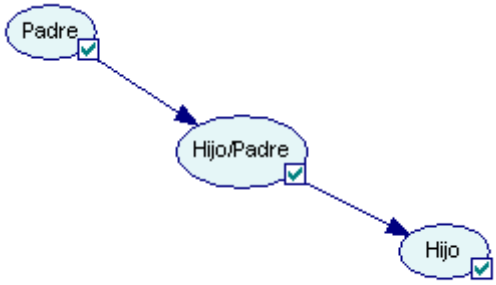

ILUSTRACIÓN 47: CONCEPTO DE NODO PADRE/HIJO

- Un nodo "hijo" puede tener varios padres, formándose así una "estructura en V".

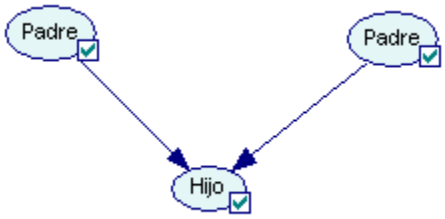

ILUSTRACIÓN 48: CONCEPTO DE ESTRUCTURA EN V

En una estructura en V, el nodo hijo puede tener tantos padres como se desee. No obstante, se debe tener en cuenta que cuantos más padres posea un nodo, más compleja será su CPT o Tabla de Probabilidad Condicional. Esto conlleva a tener una inferencia más costosa computacionalmente y un diseño más complejo ya que las tablas internas crecen exponencialmente con cada padre que se añada al nodo hijo.

Además de esto, es muy importante puntualizar el hecho de que en una estructura en V se generan más relaciones causales que las que se muestran explícitamente mediante los arcos existentes. Al existir explícitamente una distribución de probabilidad condicional entre el nodo hijo con respecto a todos los padres, a su vez se crea una relación implícita entre todos los padres.

Debido a este hecho, aunque el hijo no varíe (por ejemplo, porque haya una evidencia fijada sobre ese nodo), si un padre varía su estado, el resto de padres también se modificarán en mayor o menor medida, dependiendo de la relación implícita que exista en la distribución condicional explícita que existe en la tabla de probabilidad condicional del nodo hijo.

- Un *cluster* es un subgrupo de nodos de una red bayesiana que a su vez componen otra red bayesiana. La función de estos subgrupos es distribuir la

red bayesiana original en diferentes puntos y así conseguir una inferencia distribuida. Estos *clusters* se comunican a través de los diferentes nodos duplicados.

- Un nodo duplicado es un nodo que está al mismo tiempo en dos *clusters* diferentes. De este modo, las relaciones causales que hay en la red bayesiana original se mantienen en el conjunto de *clusters*.

Estos conceptos son básicos para la comprensión del algoritmo de intercambio de creencias ya que son referenciadas constantemente. No obstante, antes de continuar con la exposición del algoritmo, se deben mostrar algunas restricciones en la selección de los *clusters* o los subgrupos en los que se divide una red bayesiana.

RESTRICCIONES EN LA CONSTRUCCIÓN DE CLUSTERS

En el proceso de diseño de una red bayesiana que se desee distribuir a través del método presentando en este anexo, se deben tener en cuenta diferentes restricciones.

- Una estructura en V presente en la red bayesiana original debe conservarse íntegramente en uno de los *clusters* en los que se descomponga ésta. Ya que de no ser así, las relaciones existentes en la red original variarían y esto podría llevar a una serie de despropósitos si no se maneja con cuidado.
- Si el nodo que se desea duplicar o compartir entre dos *clusters* tiene un solo padre en la red bayesiana original, se puede duplicar conservando la coherencia entre los dos *clusters*, es decir, conservando el estado *a priori* del nodo. Para conseguir esto, su CPT debe modificarse. En el *cluster* en el que continúe siendo nodo hijo, se mantendrá intacta. Sin embargo, en el otro *cluster*, en el cual se queda huérfano, su CPT pasará a ser la de un nodo padre.

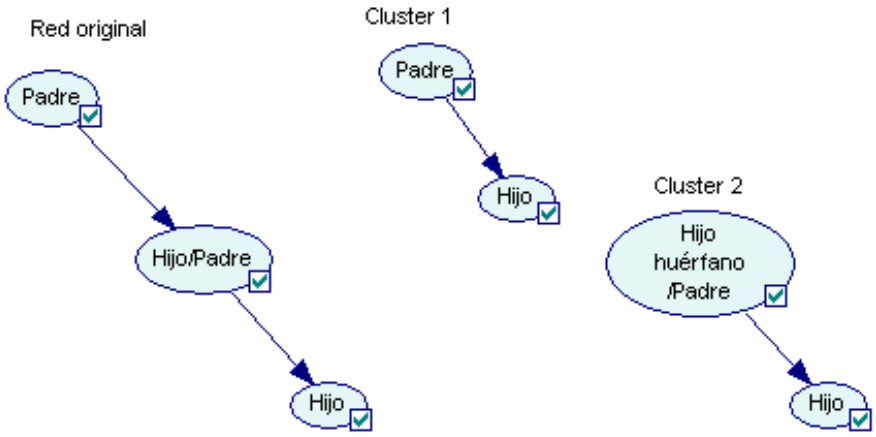

ILUSTRACIÓN 49: NODO COMPARTIDO - NODO HIJO HUÉRFANO

Si se cumplen estas restricciones en la construcción de los diferentes *clusters*, se puede ejecutar el algoritmo de intercambio de creencias sobre estos.

ALGORITMO

Si se desea distribuir una red bayesiana, es evidente que se debe simular el comportamiento de ésta. Para ello, todos los nodos y relaciones causales existentes en la red original se deben mantener, tanto en su estructura como en su estado, en los diferentes *clusters* en los que se divida la red.

Para alcanzar este propósito, hay que tener en cuenta un punto clave que es cómo realiza la inferencia una red bayesiana. Aunque existen múltiples algoritmos de inferencia, todos ellos tienen una característica común, funcionan añadiendo una evidencia y alcanzando estados estables.

Tras este paso, la red está de nuevo lista para recibir más evidencias.

El algoritmo usado funciona fijando creencias en diferentes nodos en lugar de fijar evidencias, lo cual proporciona una flexibilidad superior a la de una red bayesiana en la que sólo se pueden añadir evidencias, es decir, hechos certeros y no creencias sobre un hecho.

Para ello, existe un método sencillo de fijar un estado deseado en un nodo. No obstante, este método no es trivial. Añadiendo un nodo hijo (nodo auxiliar) a cualquier otro nodo (nodo duplicado dentro de un *cluster*), se obtiene el control sobre este nodo. Modificando la tabla de probabilidad condicional del nodo auxiliar se puede alcanzar el valor deseado en el nodo duplicado.

Para la implementación de este método de actualización de nodos, sólo se necesita conocer el valor anterior del nodo a actualizar (el que el nodo tiene en el momento en el que se desee actualizar el estado) y el valor que se desea alcanzar después de la actualización.

Así aplicando el teorema de Bayes, se puede conocer la tabla de probabilidad condicional del nodo auxiliar para fijar un valor deseado en el nodo compartido.

$$P(Auxiliar = Estado\ de\ actualización/Estado\ compartido \quad = EstadoX) = \frac{Pdeseada(Estado\ compartido = EstadoX)}{Pactual(Estado\ compartido = EstadoX)} \cdot Paux$$

Tras conseguir la CPT del nodo auxiliar, se fija el estado de actualización en este nodo como una evidencia más y, tras realizar la inferencia, el nodo compartido tiene el valor deseado.

EJEMPLO DE USO

Para facilitar la comprensión, se presenta un ejemplo simple.

ILUSTRACIÓN 50: RED ORIGINAL DE EJEMPLO

Para simplificar más el ejemplo, se va a suponer que todos los nodos sólo tienen dos estados: Estado"Nodo"0 y Estado"Nodo"1.

Las tablas de probabilidad de cada nodo son las presentadas a continuación.

A/B	EstadoB0	EstadoB1
EstadoA0	0.9	0.3
EstadoA1	0.1	0.7

TABLA 60: CPT DE A DADO B

B/C	EstadoC0	EstadoC1
EstadoB0	0.2	0.6
EstadoB1	0.8	0.4

TABLA 61: CPT DE B DADO C

C	
EstadoC0	0.3
EstadoC1	0.7

TABLA 62: CPT DE C

Estas CPT's dan unos estados a priori (sin añadir ninguna evidencia a la red):

Estado / Nodo	A	B	C
Estado"Nodo"0	0.59	0.48	0.3
Estado"Nodo"1	0.41	0.52	0.7

TABLA 63: TABLA DE ESTADOS A PRIORI

Teniendo una red bayesiana como la presentada en la Ilustración 50, se desea distribuirla en dos *clusters* para realizar inferencia en dos dispositivos diferentes conjuntamente.

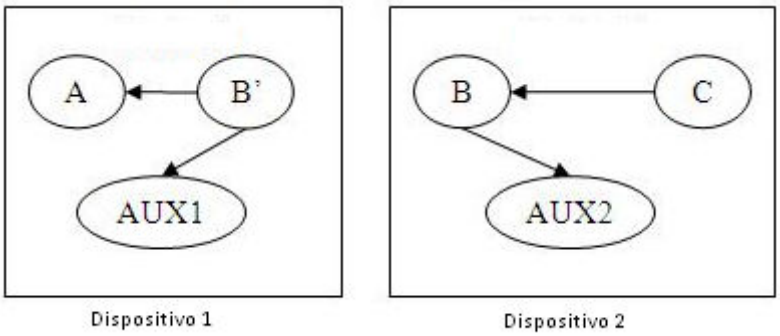

ILUSTRACIÓN 51: CLUSTERS DE EJEMPLO

Con este cambio se presentan las siguientes CPT's:

- Dispositivo 1

A/B'	**EstadoB0**	**EstadoB1**
EstadoA0	0.9	0.3
EstadoA1	0.1	0.7

TABLA 64: CPT DE A DADO B - CLUSTER DISPOSITIVO 1

B'	
EstadoB0	0.48
EstadoB1	0.52

TABLA 65: CPT DE B - CLUSTER DISPOSITIVO 1

- Dispositivo 2

B/C	**EstadoC0**	**EstadoC1**
EstadoB0	0.2	0.6
EstadoB1	0.8	0.4

TABLA 66: CPT DE B DADO C - CLUSTER DISPOSITIVO 2

C	
EstadoC0	0.3
EstadoC1	0.7

TABLA 67: CPT DE C - CLUSTER DISPOSITIVO 2

Manteniendo así la coherencia de la red, ya que el nodo B y B' tienen el mismo estado en los dos *clusters*.

Para terminar, falta presentar las CPT's de los nodos auxiliares. Estas tablas se modifican dinámicamente en cada intercambio de creencias; así es indiferente el valor inicial de estas. Lo más sencillo es mantener todos los valores equiprobables, es decir,

entropía máxima. En este caso, todas las celdas de sus tablas de probabilidad son iguales.

AUX/B	EstadoB0	EstadoB1
TriggerState	0.5	0.5
NonTriggerState	0.5	0.5

TABLA 68: CPT DE LOS NODOS AUXILIARES

De los dos estados de estos nodos, sólo interesa el que se usa para propagar la creencia hacia el nodo compartido ("*TriggerState"*). El estado restante es obligatorio para mantener la coherencia dentro de la tabla, ya que todas las probabilidades del nodo auxiliar dado un estado del nodo compartido deben sumar la unidad.

Una vez presentada la situación inicial de los *clusters*, se procede a fijar una evidencia en un nodo de la red. Por ejemplo, en el nodo A, se fija la evidencia de "A=EstadoA0" en el dispositivo 1. Tras realizar la inferencia correspondiente, el valor de los nodos es el siguiente:

Estado/Nodo	A	B'
Estado"Nodo"0	1	0.73
Estado"Nodo"1	0	0.27

TABLA 69: TABLA DE ESTADOS TRAS INFERENCIA EN DISPOSITIVO 1

Ahora que la creencia sobre el nodo B en el dispositivo 1 ha cambiado, se debe propagar hasta el dispositivo 2. Una vez enviado el nuevo estado, es decir, la creencia sobre un nodo concreto, el dispositivo receptor comienza la actualización de la creencia.

Primero debe calcular los valores de la CPT del nodo auxiliar.

$$P(AUX2 = TriggeState/B = EstadoB0) = \frac{Pdeseada(B = EstadoB0)}{Pactual(B = EstadoB0)} \cdot Paux$$

$$P(AUX2 = TriggerState/B = EstadoB1) = \frac{Pdeseada(B = EstadoB1)}{Pactual(B = EstadoB1)} \cdot Paux$$

Dado que:

$$Pdeseada(B = EstadoB0) = 0.73$$

$$Pdeseada(B = EstadoB1) = 0.27$$

$$Pactual(B = EstadoB0) = 0.48$$

$$Pactual(B = EstadoB1) = 0.52$$

Hay que tener en cuenta que los valores del estado actual del nodo B se deben tomar con el nodo auxiliar libre del cualquier evidencia, es decir, no puede darse ni AUX2=*TriggerState* ni AUX2=*NonTriggerState*. Ya que esto influiría en el proceso de inferencia posterior llevando así a un estado incoherente en la red.

El siguiente paso es realizar los cálculos apropiados:

$$P(AUX2 = TriggerState/B = EstadoB0) = \frac{0.73}{0.48} \cdot 0.1 = 0.15208$$

$$P(AUX2 = TriggerState/B = EstadoB1) = \frac{0.27}{0.52} \cdot 0.1 = 0.05192$$

$$P(AUX2 = NonTriggerState/B = EstadoB0) = 1 - P(AUX2 = TriggerState/B = EstadoB0) = 0.84792$$

$$P(AUX2 = NonTriggerState/B = EstadoB1) = 1 - P(AUX2 = TriggerState/B = EstadoB1) = 0.94808$$

Consiguiendo una CPT para el nodo auxiliar:

AUX2/B	**EstadoB0**	**EstadoB1**
TriggerState	0.15208	0.05192
NonTriggerState	0.84792	0.94808

TABLA 70: CPT AUX2 TRAS LA ACTUALIZACIÓN DE CREENCIA – CLUSTER DISPOSITIVO 2

Una vez obtenida esta versión de la CPT, se fija una evidencia en este nodo auxiliar (AUX2 = *TriggerState*) y se realiza la correspondiente inferencia, obteniendo los siguientes valores en los nodos del *clusters* del dispositivo 2.

Estado/Nodo	**B**	**C**
Estado"Nodo"0	0.73	0.22
Estado"Nodo"1	0.27	0.78

TABLA 71: TABLA DE ESTADOS DE LOS NODOS TRAS LA INFERENCIA - CLUSTER DISPOSITIVO 2

Tras este proceso, los valores de los *clusters* en los que se dividió la red original son los mismos que se habrían obtenido con una red bayesiana centralizada.

Si a continuación se fijase una evidencia en C o otra diferente en A, el proceso a seguir sería el mismo usando el nodo auxiliar AUX1 o de nuevo AUX2 respectivamente. Como conclusión, los nodos auxiliares te permiten fijar un estado dado en un nodo del cual sean hijos.

ANEXO V: FICHEROS DE CONFIGURACIÓN DE LA MAQUETA DE PRUEBAS

INTRODUCCIÓN

Este anexo pretende ser una sección de referencia en lo concerniente a la maqueta de pruebas. En las siguientes secciones se muestran los archivos de configuración con respecto a los dispositivos en los que se despliegan.

Primero se muestra la configuración del servidor de soporte de operación, ya que en él se ejecuta el contenedor principal (básico para el funcionamiento del sistema).

En segundo lugar, la configuración de las pasarelas residenciales, diferenciando el de cliente y el de proveedor.

Como se comprobará a continuación, los archivos de configuración de las pasarelas residenciales y el servidor de soporte de operación son de diferente formato. Esto es debido al uso de JADE-OSGi en las pasarelas y de WADE en el servidor de operación.

FICHEROS DE CONFIGURACIÓN DEL SERVIDOR DE SOPORTE DE OPERACIÓN

A continuación se muestran los tres ficheros más importantes de la configuración de la plataforma WADE.

MAIN.PROPERTIES

Este fichero reside dentro del directorio "*cfg*" de la plataforma WADE.

En este fichero se especifica la configuración del contenedor principal de la plataforma de agentes.

```
#
# WADE Main Container property file
#
local-port= 2999

cfa_bootdaemon_port=9999

# Additional JADE arguments (see the JADE Administrator's Guide)

services=jade.core.messaging.TopicManagementService;jade.core.mobility
.AgentMobilityService;jade.core.event.NotificationService;jade.core.re
plication.MainReplicationService;jade.core.nodeMonitoring.UDPNodeMonit
oringService
platform-id=MAGNETO

gui=true

nomtp=true

jade_core_AgentContainerImpl_enablemonitor=true

jade_core_MainContainerImpl_replicatedagents=com.tilab.wade.cfa.Config
urationAgent;jade.tools.rma.rma;com.tilab.wade.raa.RuntimeAllocatorAge
nt;gui:com.tilab.wade.cfa.ManagementAgent

#configuration for the UDPNodeMonitoringService

jade_core_nodeMonitoring_UDPNodeMonitoringService_pingdelay=2000

jade_core_nodeMonitoring_UDPNodeMonitoringService_pingdelaylimit=30000

jade_core_nodeMonitoring_UDPNodeMonitoringService_unreachablelimit=360
0001

# DF configuration

jade_domain_df_maxresult=5000

jade_domain_df_autocleanup=true
```

CÓDIGO 11: FICHERO MAIN.PROPERTIES DEL SERVIDOR DE SOPORTE OPERACIÓN

TYPES.XML

Este fichero reside en el mismo directorio que el anterior, "*cfg*" dentro de la plataforma WADE.

En este fichero se especifican los diferentes tipos de agentes que serán usados más adelante en el siguiente fichero de configuración para representar instancias de estos tipos de agentes.

```
<platform>
      <agentRoles>
            </agentRole>
            <agentRole description="Knowledge Agent">
            </agentRole>
            <agentRole description="Belief Agent">
            </agentRole>
            <agentRole description="Observation Agent">
            </agentRole>
      </agentRoles>
      <agentTypes>
            <agentType description = "Knowledge Agent"
            className="kowgar.agent.KnowledgeAgent.KnowledgeAgent"
            role="Knowledge Agent">
                  <properties>
                        <property name = "KNOWLEDGE_FILE" value
                        ="Magneto.sl"/>
                  </properties>
            </agentType>
            <agentType description = "Belief Agent"
            className="kowgar.agent.BeliefAgent.BeliefJadeAgent"
            role="Belief Agent">
                  <properties>
                        <property name = "T_MAX_PROCESSING"
                        value="35000" />
                        <property  name  =  "T_WAITING_FOR_OBSERVATION"
                        value ="6000"/>
                        <property name = "T_WAITING_FOR_BELIEF" value
                        ="9000"/>
                        <property  name  =  "T_ONE_INFERENCE"  value=
                        "2000"/>
                  </properties>
            </agentType>
            <agentType description = "ISP Connectivity Agent"
      className="magneto.agent.observationAgentImpl.ConnectivityAgent"
            role="Observation Agent">
                  <properties>
                        <property name="PINGS" value="3"/>
                  </properties>
            </agentType>
      </agentTypes>
</platform>
```

CÓDIGO 12: FICHERO TYPES.XML DEL SERVIDOR DE SOPORTE DE OPERACIÓN

_TARGET.XML

Este fichero reside en una subcarpeta del directorio anterior, "*cfg/configuration*" de la plataforma WADE.

En este fichero se especifican las instancias concretas de los tipos de agentes presentes en el fichero anterior "types.xml".

```
<platform description="Application configuration for the MAGNETO
prototype"
      name="MAGNETO">
      <hosts>
      <host name="caritas">
      <containers>
            <container jadeProfile="monitored"
            name="Knowledge-Container">
                  <agents>
                  <agent name="Knowledge Agent"
                  type="Knowledge Agent">
                        <parameters>
                        <parameter key="TOPIC_SUBSCRIPTION">
                              <value>SCENE-TOPIC-SUBSCRIPTIONS</value>
                        </parameter>
                        </parameters>
                  </agent>
                  </agents>
            </container>
            <container jadeProfile="monitored" name="ISP-
            Container">
                  <agents>
                  <agent name="ISP Belief Agent"
                  type="Belief Agent">
                        <parameters>
                        <parameter key="AREA-LAN">
                              <value>ISP</value>
                        </parameter>
                        <parameter key="SCENARIO-ID">
                              <value>ISPCluster</value>
                        </parameter>
                        <parameter key="LOGGING_TO_FILE">
                              <value>YES</value>
                        </parameter>
                        <parameter key="LOGGING_LEVEL">
                              <value>ALL</value>
                        </parameter>
                        <parameter key="LIST_OF_BELIEVES">
                              <value>ISPProblem</value>
                        </parameter>
                        </parameters>
                  </agent>
                  <agent name="ISP Connectivity Agent" type="ISP
                  Connectivity Agent">
                        <parameters>
                        <parameter key="AREA-LAN">
                              <value>ISP</value>
                        </parameter>
                        <parameter key="COST">
                              <value>1</value>
```

```
                        </parameter>
                        <parameter key="TIME-COST">
                              <value>1</value>
                        </parameter>
                        <parameter key="LOGGING_TO_FILE">
                              <value>YES</value>
                        </parameter>
                        <parameter key="LOGGING_LEVEL">
                              <value>ALL</value>
                        </parameter>
                        <parameter key="LIST_OF_OBSERVATIONS">

      <value>ConnectivityTestISPToUser;ConnectivityTestISPToProvider</
value>
                        </parameter>
                        </parameters>
                  </agent>
                  </agents>
            </container>
      </containers>
      </host>
      </hosts>
</platform>
```

CÓDIGO 13: FICHERO _TARGET.XML DEL SERVIDOR DE SOPORTE DE OPERACIÓN

FICHEROS DE CONFIGURACIÓN DE PASARELA RESIDENCIAL

SYSTEM.PROPERTIES

Este fichero reside dentro del directorio de configuración de la plataforma OSGi.En este fichero se especifica la configuración del contenedor de agentes que se ejecuta en la pasarela.

```
#     JADE OSGI configuration properties
jade.container=true
jade.host=caritas
jade.port=2999
jade.platform-id=MAGNETO
jade.container-name=ClientHANContainer (pasarela de cliente)
jade.container-name=ProviderHANContainer (pasarela de proveedor)
jade.services=jade.core.nodeMonitoring.UDPNodeMonitoringService;jade.c
ore.messaging.TopicManagementService;jade.core.messaging.MessagingServ
ice;jade.core.event.NotificationService
jade.nomtp=true
jade.jade_core_AgentContainerImpl_enablemonitor=true
jade.jade_core_nodeMonitoring_UDPNodeMonitoringService_pingdelay=2000
jade.jade_core_nodeMonitoring_UDPNodeMonitoringService_pingdelaylimit=
30000
jade.jade_core_nodeMonitoring_UDPNodeMonitoringService_unreachablelimi
t=3600000
java.util.logging.config.file=./conf/logging.properties
```

CÓDIGO 14: FICHERO SYSTEM.PROPERTIES DE PASARELA RESIDENCIAL

LOGGING.PROPERTIES

Este fichero también reside en el directorio de configuración de la plataforma OSGi y contiene la configuración de las traza de los diferentes agentes.

```
handlers=java.util.logging.ConsoleHandler,
      java.util.logging.FileHandler
java.util.logging.ConsoleHandler.level = ALL
java.util.logging.ConsoleHandler.formatter=java.util.logging.SimpleFor
      matter
java.util.logging.FileHandler.level = ALL
java.util.logging.FileHandler.pattern= ./log/JADElogs%u.log
java.util.logging.FileHandler.limit=50000
java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormat
      ter
jade.core.level = FINE
```

CÓDIGO 15: FICHERO LOGGING.PROPERTIES DE LA PASARELA RESIDENCIAL

AGENTS.XML

Este fichero reside en el directorio de configuración de la plataforma OSGi y contiene las instancias de los agentes que deben ejecutarse con sus respectivas configuraciones.

Se muestran a continuación dos ficheros diferentes, uno para la configuración de la pasarela residencial de cliente y otro para la pasarela residencial de proveedor.

```
<agents>
      <agent name="OsgiIFAgent" bundle="magnetoBundle"
            className="magneto.agent.interfaceAgentImpl.OsgiIfAgent">
            <parameters>
                  <parameter key="AREA-LAN" value="CLIENTHAN"/>
                  <parameter key="LOGGING_TO_FILE" value="YES"/>
                  <parameter key="LOGGING_LEVEL" value="ALL"/>
                  <parameter key="PROVIDER_IPGW"
                        value="10.95.14.27"/>
                  <parameter key="ISPSERVER_IP"
                        value="10.95.3.110"/>
            </parameters>
      </agent>
      <agent name="DiagnosisVOD" bundle="magnetoBundle"

      className="kowgar.agent.diagnosisAgent.DiagnosisJadeAgent">
            <parameters>
                  <parameter key="AREA-LAN" value="CLIENTHAN"/>
                  <parameter key="SCENARIO-ID"
                        value="UserHANCluster"/>
                  <parameter    key="T_MAX_PROCESSING"    value="30000"
                        />
                  <parameter key="T_WAITING_FOR_OBSERVATION"
                        value="6000"/>
                  <parameter key="T_WAITING_FOR_BELIEF"
                        value="10000"/
                  <parameter key="T_ONE_INFERENCE" value="2000"/>
                  <parameter key="LOGGING_TO_FILE" value="YES"/>
                  <parameter key="LOGGING_LEVEL" value="ALL"/>
```

```
            </parameters>
      </agent>
      <agent name="Connectivity Agent User HAN"
            bundle="magnetoBundle"
className="magneto.agent.observationAgentImpl.ConnectivityAgent">
            <parameters>
                  <parameter key="AREA-LAN" value="USERHAN"/>
                  <parameter key="COST" value="1"/>
                  <parameter key="TIME_COST" value="1"/>
                  <parameter key="GET_IP_FROM_NAME" value="NO"/>
                  <parameter key="LOGGING_TO_FILE" value="YES"/>
                  <parameter key="LOGGING_LEVEL" value="ALL"/>
                  <parameter key="LIST_OF_OBSERVATIONS"
      value="ConnectivityTestUserHAN;ConnectivityTestUserToISP;Connect
      ivityTestUserToProvider"/>
                  <parameter key="PINGS" value="3"/>
            </parameters>
      </agent>
</agents>
```

CÓDIGO 16: FICHERO AGENTS.XML DE LA PASARELA RESIDENCIAL DE CLIENTE

```
<agents>
      <agent    name="Local    Access    Server    Observation    Agent"
      bundle="magnetoBundle"
      className="magneto.agent.observationAgentImpl.LocalAccessServerA
      gent">
            <parameters>
                  <parameter key="AREA-LAN" value="PROVIDERHAN"/>
                  <parameter key="COST" value="5"/>
                  <parameter key="TIME_COST" value="1"/>
                  <parameter key="GET_IP_FROM_NAME" value="NO"/>
                  <parameter key="LOGGING_TO_FILE" value="YES"/>
                  <parameter key="LOGGING_LEVEL" value="ALL"/>
            </parameters>
      </agent>
      <agent      name="Connectivity      Agent      Provider      HAN"
      bundle="magnetoBundle"
      className="magneto.agent.observationAgentImpl.ConnectivityAgent"
      >
            <parameters>
                  <parameter key="AREA-LAN" value="PROVIDERHAN"/>
                  <parameter key="COST" value="4"/>
                  <parameter key="TIME_COST" value="1"/>
                  <parameter key="GET_IP_FROM_NAME" value="NO"/>
                  <parameter key="LOGGING_TO_FILE" value="YES"/>
                  <parameter key="LOGGING_LEVEL" value="ALL"/>
                  <parameter key="LIST_OF_OBSERVATIONS"
                  value="ConnectivityTestProviderHAN"/>
                  <parameter key="PINGS" value="3"/>
            </parameters>
      </agent>
      <agent    name="LookupService    Agent"    bundle="magnetoBundle"
      className="magneto.agent.observationAgentImpl.ServiceLookupTestA
      gent">
            <parameters>
                  <parameter key="AREA-LAN" value="PROVIDERHAN"/>
                  <parameter key="LOGGING_TO_FILE" value="YES"/>
                  <parameter key="COST" value="3"/>
                  <parameter key="TIME_COST" value="1"/>
```

```
                  <parameter key="LOGGING_LEVEL" value="ALL"/>
            </parameters>
      </agent>
      <agent   name="Provider   Belief   Agent"   bundle="magnetoBundle"
      className="kowgar.agent.BeliefAgent.BeliefJadeAgent">
            <parameters>
                  <parameter key="SCENARIO-ID"
                  value="ProviderHANCluster"/>
                  <parameter key="AREA-LAN" value="ProviderHAN"/>
                  <parameter key="LOGGING_TO_FILE" value="YES"/>
                  <parameter key="COST" value="3"/>
                  <parameter key="TIME_COST" value="1"/>
                  <parameter key="LOGGING_LEVEL" value="ALL"/>
                  <parameter key="LIST_OF_BELIEVES"
                  value="ProviderHANProblem"/>
                  <parameter key="T_MAX_PROCESSING" value ="35000"/>
                  <parameter   key="T_WAITING_FOR_OBSERVATION"   value
                  ="6000"/>
                  <parameter key="T_WAITING_FOR_BELIEF" value
                  ="9000"/>
                  <parameter key="T_ONE_INFERENCE" value ="2000"/>
            </parameters>
      </agent>
</agents>
```

CÓDIGO 17: FICHERO AGENTS.XML DE LA PASARELA RESIDENCIAL DE PROVEEDOR

TRABAJOS CITADOS

Acid, Silvia, Luis M. de Campos, Juan M. Fernández-Luna, Susana Rodríguez, José María Rodríguez, y José Luis Salcedo. «A comparison of learning algoritms for Bayesian networks: a case study based on data from an emergency medical service.» *Artificial Intelligence in Medicine 30*, 2004: 215-232.

Akaike, Hirotugu. «A new look at the statistical model identification.» *IEEE Transactions on Automatic Control 19*, 1974: 716-723.

Andoutsopoulos, I., G. Sakkis, G. Paliouras, V. Karkaletsis, C.D. Spyropoulos, y P. Stamatopoulos. «Learning to filter spam e-mail: A comparison of Naive Bayesian and a Memory Based Approach.» *4th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2003: 1-13.

Babaoglu, Özalp, y Keith Marzullo. *Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms.* Laboratory for Computer Science, University of Bologna, Bologna, Italy: ACM Press/Addison-Wesley Publishing Co., New York, 1993.

Badonnel, R., R State, y O Festor. «Probabilistic Management of Ad-Hoc Networks.» *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS'06), Vancouver, Canada*, April 2006.

Cerquides, Jesús, y Ramón López de Màntaras. «Tractable Bayesian Learning of Tree Augmented Naive Bayes Classifiers.» *20th ICML*, 2003: 75-82.

Cheng, V.H.L., L.S. Crawford, y P.K. Menon. *Air Traffic Control Using Genetic Search Techniques.* Proceedings of the 14th IFORS Triennial Conference, 1999.

Collouris, Dollimore, y Klindberg. *Distributed Systems: Conceps and Design.* Addison-Wesley, 1994.

Cooper, G. F. *The computational complexity of probabilistic inference using Bayesian belief networks.* Medical Computer Science Group, Knowledge Systems Laboratory, Stanford University, Stanford, USA, 1990.

Cui, Li-Jie, y George Kuczera. «Optimizing Urban Water Supply Headworks using Probabilistic Search Methods.» *Journal of Water Resources Planning and Management* 129, nº 5 (2003): 380-387.

Dagum, P., y M. Luby. «Approximating probabilistic inference in Bayesian belief networks is NP-hard.» *Artificial Intelligence* 60 (1993): 141-153.

Dantu, Ram, y Prakash Kolan. «Detecting Spam in VoIP Networks.» *Steps to Reducing Unwanted Traffic on the Internet Workshop* (USENIX Association), 2005: 5-6.

Darwiche, Adnan. *Recursive Conditioning.* Los Angeles, USA: Computer Science Department, UCLA, 2000.

DSL Forum Technical Report. «TR-069 CPE WAN Management Protocol v1.1.» 2007.

Frints, Martijn, David Ortega Abad, y Pablo Arozarena Llopis. «Madeira: A peer-to-peer approach to network management.» *Comunicaciones de Telefónica I+D nº 38* (Dial), 2006: 219-232.

Granger, Millett, Max Henrion, y Mitchell Small. *Uncertainty: a guide to dealing with uncertainty in quantitative risk and policy analysis.* Cambridge: Cambridge University Press, 2007.

ITU-T Princ. *Principles for a Telecommunications Manager Networks.* Geneva, Switzerland: Recommendation M.3010, 2000.

ITU-T Recom. *Overview of TMN Recommendations.* Geneva, Switzerland: Recommendation M.3000, 2000.

Jennings, Brendan, y otros. «Towards Autonomic Management of Communications Networks.» *IEEE Communications Magazine* 45, nº 10 (2007): 112-121.

Jensen, F. V., S. L. Lauritzen, y K. G. Olesen. «Bayesian updating in causal probabilistic networks by local computations.» *Computational Statistic Quarterly*, 1990: 269-282.

Kandula, S, D Katabi, y J.P. Vasseur. *Shrink: A tool for failure diagnosis in IP networks.* MineNet Workshop, SIGCOMM, 2005.

Kjaerilff, Uffe B., y Anders L. Madsen. *Bayesian Networks and Influence Diagrams: A guide to construction and analysis.* Aalborg, Denmark: Springer, 2008.

Lauritzen, S. L., y D. J. Spiegelhalter. «Local computations with probabilities on graphical structures and their application to expert systems.» *Journal of the Royal Statistical Society*, nº 50 (1988): 157-224.

Leitner, Markus, Philipp Leitner, Martin Zach, Sandra Collins, y Claire Fahy. «Fault management based on peer-to-peer paradigms.» Tenth IFIP/IEEE International Sympsium on Integrated Network, Munich, Germany, 2007.

Liu, Rose F., y Rusmin Soetjipto. «Learning on Bayesian Networks.» *Techniques in Artificial Intelligence* (Massachusetts Institute of Technology), 2004.

Luckham, David C., y Brian Frasca. *Complex Event Processing in Distributed Systems.* Standfor University: Program Analysis and Verification Group, 1998.

Martin-Flatin, J. *Distributed event correlation and self-managed systems.* Properties in Complex Information Systems, Bertinoro, Italy: 1st International Workshop on Self, 2004.

Miller, Ed, Flemming Andreasen, y Glenn Russell. *The PacketCable Architecture.* IEEE Communications Magazine, 2001.

Mizoguchi, Riichiro, Eiichi Sunagawa, Kouji Kozaki, y Yoshinobu Kitamura. *The model of roles within an ontology development tool: Hozo.* The Institute of Scientific and Industrial Research, Osaka University, 2007.

Murch, Richard. *Autonomic Computing.* IBM Press, 2004.

Olmsted, Scott M. *On representing and solving decision problems.* Doctoral Thesis, Department of Engineering-Economic Systems, Stanford University, 1983.

OSGi Alliance. «OSGi Service Platform Core Specification Release 4, Version 4.2.» 2009.

Pearl, Judea. «Bayesian Networks: A model of self-activated memory for evidential reasoning.» *Proceedings, Cognitive Science Society*, 1985: 329-334.

Pearl, Judea, y J.H. Kim. «A computational model for causal and diagnostic reasoning in inference systems.» *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI)*, 1983: 190-193.

Pearl, Judea, y Morgan Kauffman. *Probabilistic Reasoning in Intelligent Systems: networks of plausible inference.* San Francisco, USA: Elsevierdirect, 1988.

Pencolé, Y. «A decentralized model-based diagnostic tool for complex systems.» *13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'01)*, 2001: 95-102.

Roure Alcobé, Josep. *Learning Bayesian Networks with an Approximated MDL Score.* Berlin, Germany: Springer, 2007.

Ruiz, Carolina. *Illustration of the K2 algorithm for Learning Bayes Net Structures.* Department of Computer Science, WPI, Massachusetts, USA: Spring, 2005.

Shachter, R. D. «Evaluating influence diagrams.» *Operations Research* 34, nº 6 (1986): 871-882.

Strassner, John C., Nazim Agoulmine, y Elyes Lehtihet. «FOCALE - A Novel Autonomic Networking Architecture.» *International Transactions on Systems, Science, and Applications Journal* 3, nº 1 (2007): 64-79.

Turcotte, D. L. «A fractal approach to probabilistic seismic hazard assessment.» *Tectonophysics* 167 (1989): 171-177.

Walton, Mary. *The Deming Management Method.* The Putnam Publishing Group, 1986.

Witten, Ian H., y Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* University of Waikato, Hamilton, New Zeland: Elsevier, 2005.