Doctoral Dissertation

Service-Oriented Multi-Agent Architecture for a Cognitive Surveillance System

Submitted in partial fulfilment of the requirements
of University of Castilla-La Mancha
for the degree of Doctor of Philosophy
(Computer Science).

November, 2009

Author

David Vallejo Fernández

Supervisors

Dr. Luis Jiménez Linares - Dr. Carlos González Morcillo

Tesis Doctoral

Arquitectura Multi-Agente basada en Servicios para un Sistema de Vigilancia Cognitivo

Presentada como parte de los requisitos necesarios
para optar al grado académico de
Doctor en Informática

Noviembre, 2009

Autor

David Vallejo Fernández
Ingeniero en Informática

Directores

Dr. Luis Jiménez Linares - Dr. Carlos González Morcillo

Autor:
DAVID VALLEJO FERNÁNDEZ

Directores:
DR. LUIS JIMÉNEZ LINARES - DR. CARLOS GONZÁLEZ MORCILLO

Este documento fue maquetado con LaTeX. Imágenes generadas con Blender/Yafray, compuestas con Gimp y OpenOffice.

Abstract

Intelligent Surveillance makes use of Artificial Intelligence techniques in order to monitor environments whose analysis is becoming more and more complex. Two of the main factors that capture this complexity are the high number of sensors deployed in the environment, usually security cameras, and the need of dealing with different aspects or events of interest simultaneously (behaviour analysis, crowd detection, identification of suspicious objects, etc).

Most of the existing surveillance systems solve specific problems in well-limited environments, but they lack of scalability and flexibility when they are reused in similar surveillance tasks or in environments that are different from those that were initially designed for.

To overcome these limitations, the adopted solution should address two key challenges. On the one hand, the use of a surveillance model based on expert knowledge that is flexible enough for dealing with different events of interest is needed. On the other hand, an architecture that gives support to this kind of models and facilitates the development and deployment of intelligent surveillance systems should be designed. Within this context, the combination of this two goals would mean a good approximation to the design and development of intelligent surveillance systems.

This thesis discusses the architecture conceived to deploy this kind of systems by means of a set of autonomous agents that cooperate to carry out the monitoring of complex environments. This multi-agent architecture is inspired by a formal model of normality analysis used to define the knowledge needed for analysing general-purpose surveillance concepts.

Several parts of the proposed architecture have been generalised to design and develop an agent platform that complies with the set of FIPA standards. This framework is conceived to facilitate the development of multi-agent systems and is supported by a modern object-oriented middleware, which allows the design of agent-based solutions in a wide variety of programming languages, operating systems and hardware platforms.

Resumen

La Vigilancia Inteligente hace uso de técnicas de Inteligencia Artificial para monitorizar entornos cuyo análisis resulta cada vez más complejo. Dos de los principales factores que determinan esta complejidad son el gran número de sensores desplegados en el entorno a vigilar, normalmente cámaras de seguridad, y la necesidad de tratar con distintos aspectos o amenazas (análisis de comportamientos, detección de tumultos, identificación de objetos sospechosos...) de manera simultánea.

La mayoría de los sistemas de vigilancia existentes en la actualidad solucionan problemas específicos en entornos bien delimitados, pero carecen de escalabilidad y de flexibilidad a la hora de reutilizarse en problemas de vigilancia similares o en entornos diferentes para los que fueron diseñados.

Para solventar esta limitación, la solución adoptada debería tener en cuenta dos aspectos fundamentales. En primer lugar, sería necesario el uso de un modelo de vigilancia basado en conocimiento experto que fuese lo suficientemente flexible como para tratar diferentes conceptos o amenazas. Por otra parte, habría que diseñar una arquitectura que diera soporte a este tipo de modelos y que facilitara el desarrollo y el despliegue de sistemas de vigilancia inteligente. En este contexto, la consolidación de estos dos objetivos supondría una buena aproximación para diseñar y desarrollar sistemas de vigilancia inteligente.

En la presente tesis se plantea una arquitectura concebida para desplegar este tipo de sistemas mediante un conjunto de agentes autónomos, los cuales cooperan para llevar a cabo la monitorización de entornos complejos. Dicha arquitectura multi-agente está inspirada por un modelo formal de análisis de normalidad, usado para definir el conocimiento necesario para estudiar conceptos de vigilancia de propósito general.

Ciertas partes del diseño de esta arquitectura se han generalizado para diseñar y desarrollar una plataforma de agentes definida en base al conjunto de estándares de FIPA. Este *framework* facilita el desarrollo de sistemas multi-agente y está soportado por un moderno *middleware* de comunicaciones orientado a objetos, lo cual posibilita la creación de soluciones basadas en agentes en una gran variedad de lenguajes de programación, sistemas operativos y plataformas *hardware*.

A mi familia, especialmente a
mis abuelos Ángeles y Amancio.
A Alba, Juana y Paco.

Acknowledgement
Agradecimientos

Acknowledgement

First, I would like to thank Dr. Paolo Remagnino for his support and direction during my stays at the Digital Imaging Research Centre (Kingston University). I am looking forward to sharing new ideas and discussions with you.

I would also like to thank people at Software Competence Center Hagenberg, especially Dr. Gerhard Weiss (now at Maastricht University) and Dr. Bernhard Moser, who provided me with new perspectives and valuable support.

Agradecimientos

En primer lugar, quiero expresar mi agradecimiento a los directores de esta tesis, los doctores Luis Jiménez Linares y Carlos González Morcillo, por su ayuda, dirección y apoyo en estos últimos, mis primeros, años de investigación.

A todos los miembros del grupo de investigación Oreto, especialmente a Cayetano, Vanesa, Javier, Raúl, Manuel, Juan Moreno, Nines, Luis Rodríguez, José Carlos, Roberto; y a los que nos dejaron hace poco, Iñaki, Blankito y Loren, por crear un inmejorable ambiente de trabajo.

Gracias a la Universidad de Castilla-La Mancha por la financiación de mis estancias de investigación en la Universidad de Kingston (Londres) y en el *Software Competence Center Hagenberg* (Hagenberg, Austria), las cuales han servido para mejorar la calidad de este trabajo. Gracias a la Escuela Superior de Informática de Ciudad Real y al Departamento de Tecnologías y Sistemas de Información por proporcionarme todos los recursos necesarios para el desarrollo de la presente tesis. Este trabajo también ha sido posible gracias a la financiación de los proyectos PBC-06-0064, PAC-06-0141 y Hesperia (*Homeland Security*: tecnologías para la seguridad integral en espacios públicos e infraestructuras), proyecto CENIT.

A Luis Jiménez, por introducirme en este mundo y darme su confianza desde el primer momento. A Carlos González, por *despejar* este primer tramo del camino cuya meta estoy a punto de alcanzar y por ser mi referente en el ámbito académico. A Javier Albusac, por ser mi compañero de fatigas y brindarme su habilidad a la hora de resolver problemas. Mi gratitud al Dr. José Jesús Castro Sánchez, por su geniales consejos y su ayuda en la formalización de la arquitectura multi-agente. A Paco Moya, por su soporte en el ámbito de los sistemas distribuidos y por facilitarnos la vida a los que usamos Debian e ICE. A Blankito y a Iñaki, por su soporte en el desarrollo y despliegue de las pruebas de la plataforma multi-agente.

A mis padres, por su confianza y sacrificio, sin los que no habría tenido la oportunidad de haber llegado tan lejos.

A Alba, por su compresión y apoyo sin límites.

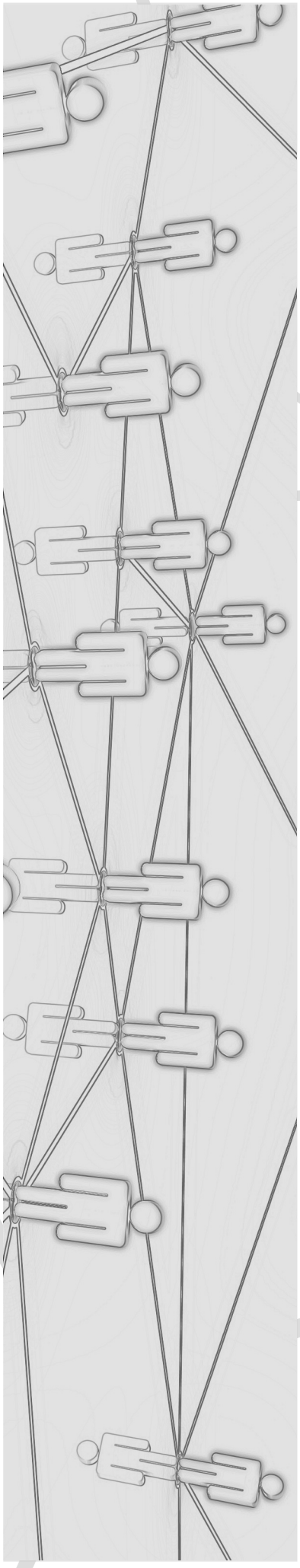
Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción



Capítulo 1
Introducción

En este primer capítulo se establecen los objetivos que se pretenden alcanzar con el desarrollo de la presente tesis doctoral, estableciendo el marco de trabajo en el que se ha realizado este trabajo de investigación. La hipótesis que se plantea es la definición de una arquitectura que permita el despliegue de sistemas de vigilancia inteligente de acuerdo a las necesidades de monitorización y análisis de entornos generales. En este contexto, resulta especialmente importante la necesidad de una arquitectura flexible y escalable que dé soporte a modelos de conocimiento experto aplicados al ámbito de la vigilancia inteligente.

"Todas las obras de arte deben empezar...
por el final."

Edgar Allan Poe

1 Introducción

1.1. Antecedentes

La **Vigilancia Inteligente** se puede entender como la aplicación de técnicas, algoritmos y métodos de Inteligencia Artificial (IA) con el objetivo de desarrollar sistemas de seguridad avanzados que lleven a cabo las tareas de vigilancia y monitorización realizadas tradicionalmente por operadores humanos. En este ámbito, la videovigilancia representa un tema de investigación muy activo, teniendo como principales metas la detección, el reconocimiento y el seguimiento de ciertos objetos a partir de secuencias de imágenes y, desde una perspectiva más general, el entendimiento y la descripción de los comportamientos de dichos objetos (ver figura 1.1). El **objetivo** de la vigilancia inteligente no es desplegar cámaras para que sean supervisadas por empleados de seguridad, sino automatizar (con un determinado grado de robustez y eficiencia) las tareas de las que se compone el proceso de vigilancia tanto como sea posible.

Actualmente, dos de los principales **retos** de los sistemas de vigilancia inteligente son la integración de grandes cantidades de datos recogidas por los dispositivos de vigilancia, típicamente cámaras de seguridad, y la aplicación de modelos basados en conocimiento experto para proporcionar una vigilancia más avanzada.

En este contexto, la meta que se persigue consiste en avanzar el estado del arte de los sistemas de vigilancia tradicionales, en los que el personal de seguridad supervisa la actividad de monitores de televisión para controlar un entorno [VR05]. Los principales **problemas** de esta dependencia respecto al operador humano son la fatiga y el cansancio derivados de la observación continua de este tipo de dispositivos de vigilancia [Smi04], sobre todo debido a la gran cantidad de éstos, los cuales superan la capacidad de los operadores humanos a la hora de monitorizarlos. En este ámbito, el uso de técnicas

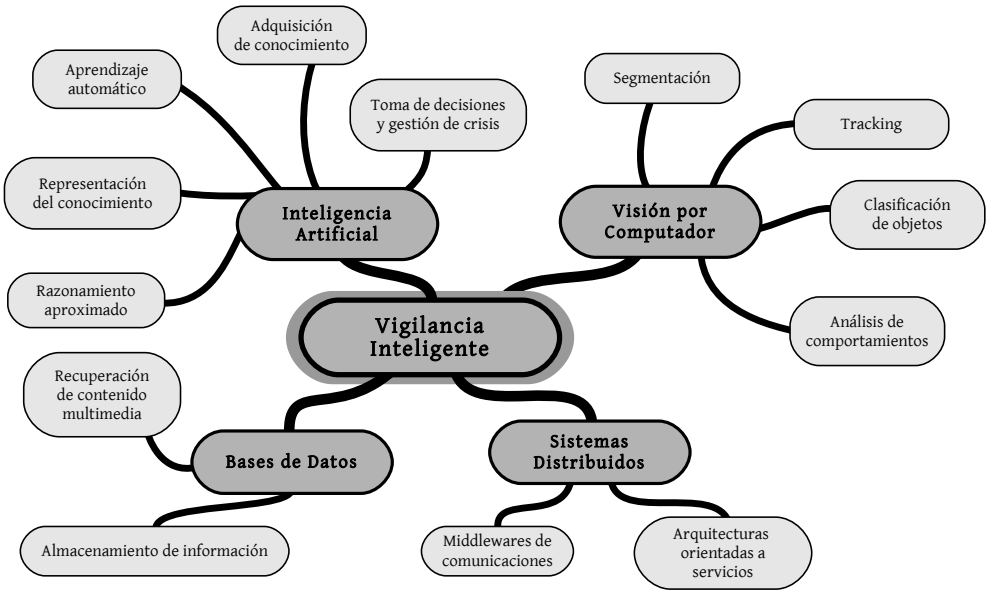


Figura 1.1: Principales áreas de investigación y técnicas relacionadas con la vigilancia inteligente.

de vigilancia inteligente puede solucionar este tipo de problemas, reduciendo la dependencia respecto al personal de seguridad y minimizando los costes.

En la actualidad, existen sistemas artificiales capaces de inferir comportamientos anómalos [HHD00], identificar objetos sospechosos o perdidos [SR00], analizar trayectorias de objetos [HXT04] o detectar aglomeraciones de personas [VBL+05], entre otras tareas (ver [VV05] para una revisión más detallada).

Normalmente, los sistemas de vigilancia están basados en arquitecturas multi-capa. Las capas de más bajo nivel llevan a cabo tareas asociadas al procesamiento de la información obtenida del entorno. Por otra parte, las de más alto nivel realizan el análisis inteligente de la escena, sirviendo como sistemas de apoyo a la decisión cuando se detectan situaciones peligrosas o anómalas. En la figura 1.2 se muestra la arquitectura abstracta de un sistema de vigilancia multi-capa y se resumen las funciones típicas de cada una de las capas que la componen.

Desde hace unos años, la **evolución** de los sistemas de vigilancia inteligentes ha venido determinada por dos factores fundamentales [VV05]:

- La distribución física de las fuentes de obtención de información, hecho que implica el uso de mecanismos de procesamiento distribuido y fusión de información.
- Las demandas de la sociedad para tener una mayor seguridad en entornos públicos y privados, lo cual requiere el uso de técnicas y métodos

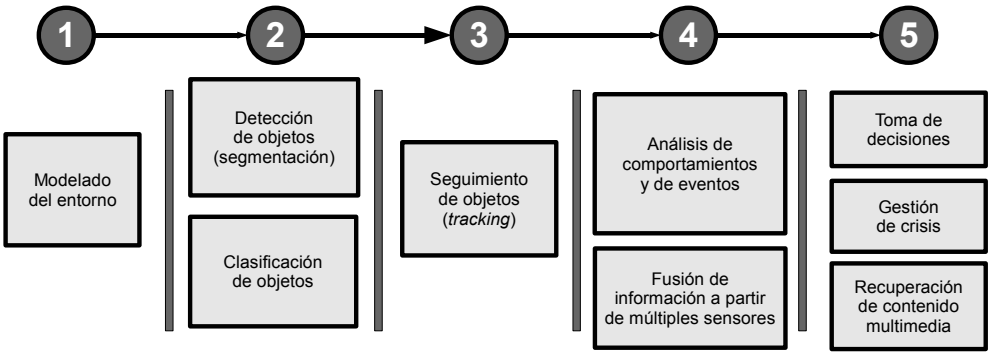


Figura 1.2: Arquitectura abstracta de un sistema de vigilancia inteligente.

de Inteligencia Artificial (sobre todo en las capas de más alto nivel) para diseñar soluciones más sofisticadas que proporcionen información más relevante en el proceso de vigilancia.

De este modo, la tendencia actual a la hora de crear un sistema de vigilancia inteligente consiste en afrontar el diseño de un sistema distribuido basado en el conocimiento, de manera que la información del entorno se utiliza para generar conocimiento, y dicho conocimiento se utiliza para proporcionar una vigilancia más avanzada.

Hasta ahora, la principal línea de investigación en este contexto ha estado basada en ofrecer soluciones para problemas específicos en entornos particulares. Estas aproximaciones proporcionan sistemas de vigilancia inteligentes que obtienen buenos resultados a la hora de monitorizar un entorno determinado para una amenaza o cuestión particular. No obstante, dichos sistemas suelen carecer de flexibilidad y escalabilidad a la hora de desplegarlos en entornos de vigilancia más generales, los cuales requieren el análisis de una mayor cantidad de objetos móviles, conceptos o elementos. Para solucionar estas limitaciones se han de tener en cuenta dos aspectos fundamentales:

1. La adopción de un **modelo** de vigilancia general, escalable e independiente del dominio de aplicación. Este modelo ha de permitir el despliegue de módulos de análisis desde distintos puntos de vista, es decir, teniendo en cuenta diversos conceptos o eventos de interés (análisis de trayectorias, interpretación de comportamientos, etc).

2. El diseño y el desarrollo de una **arquitectura** flexible que dé soporte a este tipo de modelos. Dicha arquitectura ha de especificar aquellos componentes necesarios para proporcionar una solución integral y ofrecer los mecanismos necesarios (comunicación, configuración, despliegue, etc) para llevar a cabo la vigilancia inteligente.

1.2. Objetivos de la investigación

A partir de las consideraciones expuestas en la sección anterior, y teniendo en cuenta la evolución de los sistemas de vigilancia inteligentes, el objetivo global de esta tesis consiste en definir una arquitectura que permita el despliegue de sistemas de vigilancia inteligentes de acuerdo a las necesidades de monitorización y análisis de entornos generales. Uno de los principales aspectos a tener en cuenta es la escalabilidad, para posibilitar el análisis de distintos conceptos o eventos y el despliegue de los elementos del sistema de acuerdo a los requisitos de vigilancia.

A continuación se enumeran los objetivos que se pretenden alcanzar mediante la elaboración de esta tesis:

- Llevar a cabo una revisión del **estado del arte** de las principales áreas de investigación vinculadas al desarrollo de una arquitectura para desplegar sistemas de vigilancia inteligentes.

  a) Estudiar la evolución de los sistemas de vigilancia y los enfoques existentes para abordar el diseño y desarrollo de una arquitectura de las características previamente mencionadas.

  b) Revisar las contribuciones más relevantes de las áreas científicas vinculadas con la vigilancia inteligente, como la Inteligencia Artificial Distribuida, la Computación Orientada a Servicios o los *middlewares* de comunicaciones.

- Proponer una **arquitectura genérica** para el despliegue de sistemas de vigilancia inteligente que ofrezca soporte para el uso de técnicas de Inteligencia Artificial, así como los mecanismos de comunicación y coordinación necesarios para proporcionar una vigilancia avanzada.

  a) Definir el grado de adecuación del uso de paradigmas Multi-Agente en el ámbito de los sistemas de vigilancia.

  b) Hacer uso de estándares y de técnicas clásicas de Ingeniería del Software para garantizar la interoperabilidad del sistema.

  c) Facilitar los mecanismos de escalabilidad y flexibilidad necesarios para construir sistemas de vigilancia que se puedan aplicar a una gran variedad de escenarios, permitiendo el uso de modelos de vigilancia de propósito general.

  d) Construir una arquitectura robusta que garantice aspectos relevantes en un sistema de vigilancia, como por ejemplo la tolerancia a fallos, la integridad de los datos o la seguridad del sistema.

  e) Abstraer los componentes y mecanismos generales de la arquitectura con el objetivo de reutilizarlos en otros dominios de aplicación en los que la tecnología de agentes pueda ser beneficiosa.

- Validar las **contribuciones** de la arquitectura propuesta en problemas de vigilancia concretos y bien definidos.

a) Definir de manera detallada en qué consiste el problema de la vigilancia inteligente, con el objetivo de disponer de un marco de trabajo bien definido.

b) Estudiar cómo el despliegue de la arquitectura del sistema de vigilancia inteligente puede contribuir a la evolución de este tipo de sistemas.

c) Abordar la integración de agentes especializados en las tareas que componen la vigilancia inteligente.

d) Evaluar empíricamente el rendimiento y la escalabilidad de la arquitectura propuesta.

1.3. Marco de trabajo

El marco de trabajo de la presente tesis está comprendido por los siguientes **proyectos de I+D+i**, los cuales han sido financiados mediante convocatorias públicas. A continuación se resumen brevemente los objetivos principales de cada uno de dichos proyectos, haciendo especial hincapié en el proyecto Hesperia, debido a que ha sido el proyecto directamente relacionado con las líneas de investigación desarrolladas en este trabajo.

- **Arquitectura multi-agente para la monitarización de la normalidad en entornos vigilados**. Proyecto financiado por la Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha (Ref: PII1 C09-0137-6488). Investigador principal: Dr. Luis Jiménez. Duración del proyecto: Abril de 2009 a Marzo de 2012.
  El objetivo general de este proyecto es definir una arquitectura multiagente que dé soporte al proceso de vigilancia inteligente basado en un modelo de análisis de normalidad, estableciendo de manera simultánea los mecanismos de despliegue y comunicación que faciliten la implantación de sistemas de vigilancia inteligente.

- **Modelo para el análisis de la normalidad de eventos y conductas en entornos monitorizados, aplicación a la videovigilancia**. Proyecto financiado por el Ministerio de Ciencia e Innovación (Ref: TIN2009-14538-C02-02). Investigador principal: Dr. Luis Jiménez. Duración del proyecto: Abril de 2009 a Marzo de 2012.
  El principal objetivo de este proyecto es la definición y aplicación de un modelo de análisis de normalidad en entornos de monitorización complejos, haciendo uso principalmente de cámaras de seguridad.

- **Hesperia** (*Homeland security: tecnologías para la seguridad integral en espacios públicos e infraestructuras*). Proyecto CENIT financiado por el Ministerio de Industria, Turismo y Comercio de España y empresas del sector privado. Investigador responsable (grupo Oreto): Dr. Luis Jiménez. Duración del proyecto: Enero de 2006 a Diciembre 2009.

El objetivo general de este proyecto es el desarrollo de tecnologías para la creación de sistemas punteros de seguridad, vigilancia y control de operaciones en espacios públicos y privados. Dentro de este proyecto y en el ámbito de la vigilancia inteligente, el grupo Oreto participa tanto en lo relativo a la aplicación de técnicas de Inteligencia Artificial para el análisis de normalidad como en el desarrollo de una arquitectura que dé soporte a dicho análisis.
Las tecnologías del proyecto persiguen la gestión de la seguridad a dos niveles. Por un lado, en relación a la gestión de la seguridad y las operaciones de infraestructuras públicas especialmente sensibles, como subestaciones eléctricas, de gas, depósitos de agua o estaciones de telecomunicaciones. Por otro lado, se persigue el incremento sustancial de los niveles de seguridad de grandes espacios públicos, como aeropuertos, estaciones de ferrocarril, puertos, centros de ciudades (especialmente en zonas peatonales), centros comerciales, etc.

- **Sarasvati** (*Movilización del conocimiento procedente de fuentes de información diversas y heterogéneas*). Proyecto financiado por la Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha (Ref: PBC-06-0064). Investigador principal: Dr. Luis Jiménez. Duración del proyecto: Junio de 2006 a Diciembre de 2008.
  El objetivo general de este proyecto se resume en el establecimiento de una metodología que permita la agregación autónoma e inteligente de nuevos recursos dentro del conocimiento global del negocio.

- **E-Pactos** (*Herramientas para Comercio Electrónico: desarrollo de una plataforma abierta para permitir actividades de comercio electrónico, aplicaciones y agentes para su uso*). Proyecto financiado por la Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha (Ref: PAC-06-0141). Investigador principal: Dr. J.J. Castro. Duración del proyecto: Enero de 2006 a Diciembre de 2008.
  El objetivo de este proyecto de investigación fue generar el conocimiento y la tecnología que faciliten las actividades de comercio electrónico en un ámbito regional, permitiendo de este modo incrementar la competitividad de las empresas de Castilla-La Mancha. Para ello se planteó y desarrolló una arquitectura que permitiera la implementación de un mercado virtual al que puedieran conectarse y asociarse cualquier proveedor o comprador que así lo deseara.

- **Devlena** (Descripción de escenas de vídeo en lenguaje natural). Proyecto financiado por el Ministerio de Educación y Ciencia (Ref: TIN2007-62568). Investigador principal: Dr. J. Moreno. Duración del proyecto: Enero de 2007 a Agosto de 2009.
  El objetivo de este proyecto consitió en diseñar e implementar un sistema de visión cognitiva capaz de describir cualitativamente escenas de vídeo.

1.4. Estructura de la tesis

Este documento está estructurado en seis capítulos: Introducción, Estado del Arte, Arquitectura del Sistema de Vigilancia, Plataforma Multi-Agente de Propósito General, Resultados y, finalmente, Conclusiones y Trabajo Futuro. Además, dicho documento se complementa con dos anexos: Diseño de un Agente de Normalidad y Detalles de Implementación.

El **capítulo 2** consiste en un estudio del estado del arte de las principales áreas de investigación vinculadas a la vigilancia inteligente. En dicha capítulo, se profundiza en la evolución de los sistemas de vigilancia, haciendo especial hincapié su arquitectura como componente fundamental para dar soporte a sistemas de vigilancia avanzados.

Debido a que el uso de técnicas de Inteligencia Artificial es uno de los principales pilares de los sistemas de vigilancia inteligente, en este capítulo también se estudia cómo la tecnología de agentes, y en particular los Sistemas Multi-Agente, puede contribuir a la evolución de este tipo de sistemas. Este hecho se debe a que ciertas cualidades básicas de los agentes, como la autonomía y el control descentralizado, se ajustan perfectamente a las características deseables en un sistema de vigilancia inteligente.

Así mismo, en el capítulo 2 se lleva a cabo un estudio de los *middlewares* de comunicaciones como posible solución transversal para abstraer la heterogeneidad asociada a los componentes de un sistema de vigilancia. Por otra parte, también se discuten las arquitecturas orientadas a servicios, con el objetivo de manifestar las ventajas que este enfoque puede aportar a la vigilancia inteligente.

En el **capítulo 3** se estudia en profundidad la propuesta de arquitectura planteada en este trabajo como soporte al despliegue de sistemas escalables de vigilancia inteligente. En este capítulo se definen los agentes que la forman y los mecanismos de comunicación empleados para especificar las interacciones entre dichos agentes. También se profundiza en los procesos de vigilancia y de los pasos necesarios para utilizar la arquitectura en el proceso de despliegue de un sistema de vigilancia inteligente.

Las soluciones generales que se extraen a partir de la arquitectura multi-agente para la vigilancia inteligente han servido para diseñar y desarrollar la plataforma de agentes que se discute en el **capítulo 4**. Dicha plataforma ha sido diseñada siguiendo los principios marcados por el comité FIPA (*Foundation for Intelligent Physical Agents*) para el desarrollo de sistemas multi-agente.

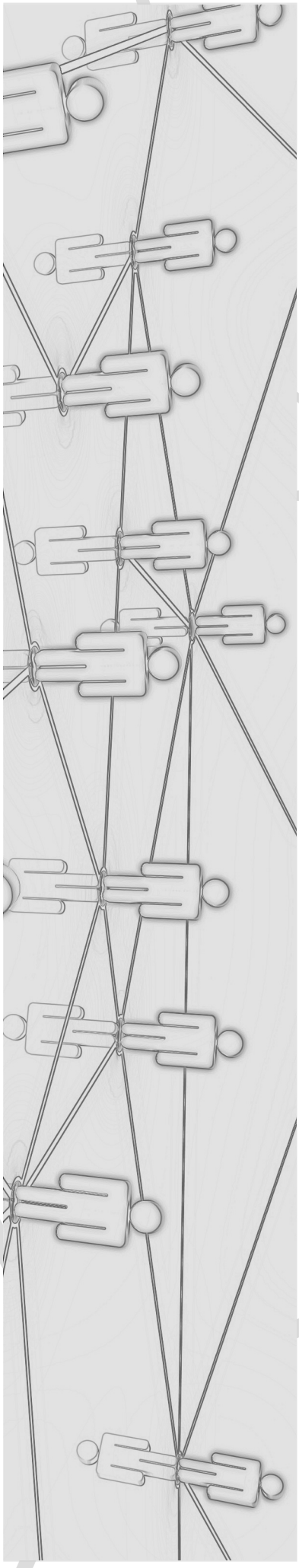
La evaluación y el análisis de resultados obtenidos a partir del uso de esta arquitectura se detallan en el **capítulo 5**, realizando el despliegue de la misma en un escenario de monitorización bien definido. En este mismo capítulo también se evalúa el rendimiento de la plataforma multi-agente de

propósito general y se estudia cómo desplegar un sistema multi-agente en un dominio de aplicación concreto. Finalmente, se referencian las contribuciones científicas más relevantes desarrolladas durante la realización de este trabajo

Finalmente, en el **capítulo 6** se discuten las conclusiones obtenidas al elaborar la presente tesis. Así mismo, en este capítulo también se plantean las principales líneas de investigación que se derivan de la realización del presente trabajo de investigación.

Capítulo 2

Estado del Arte



Capítulo 2
Estado del Arte

En este capítulo se lleva a cabo un estudio de los principales conceptos de la Inteligencia Artificial Distribuida, los Sistemas de Vigilancia Inteligente, las Arquitecturas Orientadas a Servicios y los *Middlewares* de Comunicaciones. Así mismo, se realiza un especial hincapié en las arquitecturas existentes diseñadas para soportar el desarrollo y despliegue de sistemas de vigilancia inteligente. En este contexto, se ha prestado especial importancia al uso de la tecnología de agentes en las distintas etapas funcionales en las que tradicionalmente se divide un sistema de vigilancia inteligente.

"El conocimiento es poder."
Sir Francis Bacon

2
Estado del Arte

2.1. Inteligencia Artificial Distribuida

2.1.1. Introducción

La Inteligencia Artificial Distribuida (*Distributed Artificial Intelligence (DAI)*) surgió como un sub-campo de la IA orientado al diseño y desarrollo de soluciones distribuidas para problemas complejos que requerían el uso de métodos de IA. Actualmente, los **Sistemas Multi-Agente** (SMA) han reemplazado a este área como el entorno en los que los agentes se comunican y, normalmente, cooperan para alcanzar un objetivo o meta común [Fer98].

Las principales líneas de investigación asociadas a este área se enumeran a continuación:

- Resolución de problemas de forma paralela, con el objetivo de modificar algunos conceptos clásicos de IA para que se puedan utilizar en sistemas multi-procesor y en *clusters* de computadores.
- Resolución de problemas de manera distribuida (*Distributed Problem Solving (DPS)*), donde el concepto de agente como entidad autónoma que puede comunicarse y cooperar con otros agentes se diseñó para abordar el desarrollo de sistemas DPS.
- Simulación basada en SMA (*Multi-Agent Based Simulation (MABS)*), como rama de la Inteligencia Artificial Distribuida encargada de construir los fundamentos necesarios para realizar tareas en distintos escenarios, como por ejemplo la simulación de comportamientos sociales.

En estas dos últimas líneas de investigación el concepto fundamental es del de **agente software**. Un agente se puede entender como una entidad

virtual o física basada en un comportamiento autónomo, entendiendo el entorno en el que se encuentra y actuando en consecuencia a los cambios producidos en él.

2.1.2. Tecnología de agentes

En la literatura existe una gran variedad de definiciones sobre el concepto de **agente inteligente**, al igual que ocurre con el término *inteligencia*, hecho que ha contribuido a crear cierta confusión. En este documento se introducirá el concepto de agente mediante las características que tradicionalmente se han asociado al mismo [WJ95]:

- **Autonomía**: los agentes operan sin la intervención directa de terceros, manteniendo cierto control sobre sus acciones y su estado interno.
- **Habilidad social**: los agentes interactúan entre sí (y posiblemente con personas) a través de algún tipo de lenguaje de comunicación entre agentes.
- **Reactividad**: los agentes perciben su entorno, el cual puede ser el mundo físico, un usuario a través de una interfaz, otros agentes, Internet o una combinación de los anteriores, y responden en función de los cambios percibidos.
- **Proactividad**: los agentes no son simples entes reactivos, sino que también son capaces de tomar la iniciativa mediante un comportamiento dirigido por objetivos.

Una de las definiciones más citadas es la de Wooldridge [Woo97]:

«*Un agente es un sistema informático situado en un entorno y que es capaz de realizar acciones de forma autónoma para conseguir sus objetivos de diseño*».

En realidad, esta definición engloba las cuatro características introducidas anteriormente, ya que menciona de manera directa la autonomía, asocia el agente a un entorno, lo cual implica poseer habilidad social, especifica que el agente realiza acciones, es decir, lo vincula al concepto de reactividad dentro de un entorno y, finalmente, termina con la idea del cumplimiento de objetivos o proactividad.

Otros autores, como por ejemplo Russel y Norvig, definen a un agente desde un punto de vista más abstracto y general (ver figura 2.1) [RN04]:

«*Un agente es cualquier cosa capaz de percibir su medioambiente con la ayuda de sensores y actuar en ese medio utilizando actuadores*».

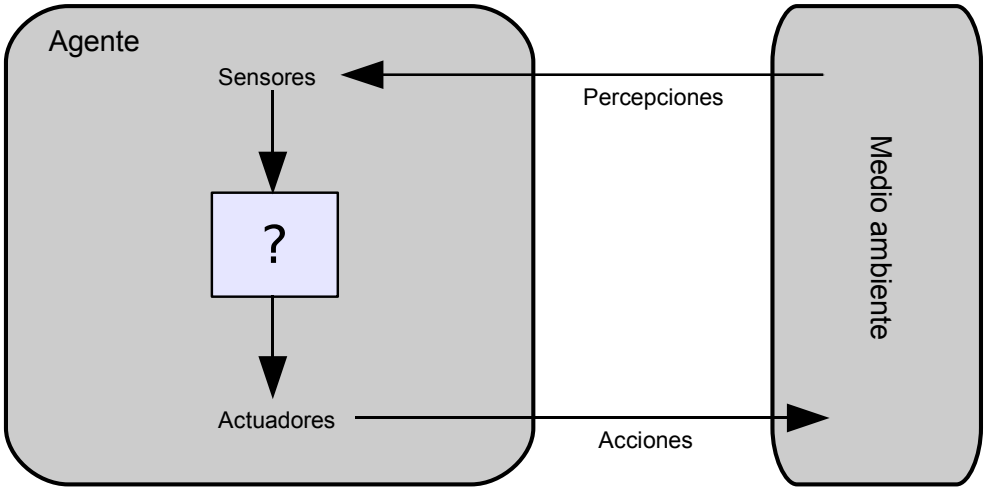


Figura 2.1: Interacción de un agente en un entorno.

Desde las primeras propuestas teóricas, Los agentes se han comparado con otros conceptos. Una de estas comparaciones ha sido con el **concepto de objeto**, vinculado a la programación orientada a objetos [BME+07]. Los objetos se definen como entidades computacionales que almacenan un estado, realizando operaciones que pueden alterar dicho estado y comunicándose con otros objetos mediante paso de mensajes. Aunque existen ciertas similaridades con el concepto de agente, la primera diferencia relevante es el grado de autonomía, ya que los objetos no pueden mostrar un control sobre su comportamiento (un ejemplo concreto es el uso de métodos públicos que pueden ser invocados por cualquier otro objeto). La diferencia entre objetos y agentes se puede resumir en el siguiente eslogan [Woo01]:

«*Los objetos lo hacen gratis; los agentes lo hacen porque quieren hacerlo*».

Otra comparación interesante es la de agentes y **sistemas expertos**. Estos sistemas jugaron un papel relevante dentro del campo de la Inteligencia Artificial en la década de los 80. Un sistema experto se define como aquél capaz de solucionar problemas o dar algún consejo de valor en un determinado dominio de conocimiento [Jac90]. Uno de los ejemplos clásicos de sistema experto es MYCIN [Sho76], cuyo objetivo era asistir a los médicos en el tratamiento de infecciones de sangre en humanos.

La principal diferencia entre este tipo de sistemas y los agentes es que los primeros no interactúan directamente con el entorno, es decir, no obtienen la información de sensores sino a través de un usuario intermedio. Del mismo modo, estos sistemas no ejercen efectos directamente sobre el entorno, sino que proporcionan una realimentación a una tercera persona. Además, los sistemas expertos no son capaces generalmente de cooperar con otros agentes.

El **entorno** juega un papel muy importante a la hora de plantear el diseño de un agente. Russel y Norvig sugieren la siguiente clasificación [RN04] en función de las propiedades del entorno:

- **Accesibles e inaccesibles**: un entorno accesible es aquél en el que el agente puede obtener información completa, precisa y actualizada sobre el estado del mismo.
- **Deterministas y no deterministas**: un entorno determinista es aquél en el que cualquier acción tiene asociada un efecto, por mínimo que sea, de manera que no existe una incertidumbre sobre el estado que resultará después de aplicar la acción.
- **Estáticos y dinámicos**: un entorno estático permanece sin cambios excepto cuando el agente desarrolla alguna acción. Por el contrario, uno dinámico puede sufrir cambios como consecuencia de las acciones desarrolladas por otros elementos del entorno.
- **Discretos y continuos**: un entorno es discreto si existe un número fijo y finito de acciones y percepciones en el mismo.

Es importante recalcar que la calidad de la decisión tomada por un agente en un momento determinado está directamente relacionada con la calidad de la información utilizada. En otras palabras, cuanto más completa y precisa sea dicha información obtenida del entorno, mayor será la probabilidad de que el agente tome la decisión más adecuada. Sin embargo, el tratamiento de más cantidad de información incrementa la complejidad asociada a tomar una decisión y, en ocasiones, puede suponer un problema en lugar de ser de más utilidad.

En el ámbito de la **vigilancia inteligente**, los entornos a monitorizar son, por regla general, inaccesibles, no deterministas, dinámicos y continuos. La información que se obtiene cuando se analiza un entorno está normalmente rodeada de incertidumbre y vaguedad. Esto se debe a la propia naturaleza física del entorno o bien al tratamiento de la información obtenida de los dispositivos de vigilancia mediante técnicas de visión por computador. Por otra parte, un entorno de vigilancia es no determinista debido a que no se puede conocer con certeza el estado en el que quedará el mismo después de aplicar una o varias acciones. Este hecho está directamente relacionado con el dinamismo del entorno, ya que existen un gran número de elementos que alteran el estado del mismo continuamente.

Además de todas estas características clásicas de los agentes, existen otros atributos que son importantes y que no todos los agentes tienen [Mas05]:

- **Razonamiento** y **aprendizaje**. Ambas características son necesarias para que el agente se comporte de manera inteligente.

- **Movilidad**. Atributo asociado a los agentes móviles, es decir, a aquellos agentes que son capaces de desplazarse entre los nodos de una red y ejecutarse en distintas plataformas.

Tanto la propia estructura interna de un agente como la organización de un conjunto de agentes están determinadas por algún tipo de arquitectura. A continuación, se estudiarán los **modelos y las arquitecturas** más significativas para un agente individual. Evidentemente, el diseño de la arquitectura de un agente difiere del diseño de la misma para un sistema compuesto de un conjunto de agentes que interactúan entre sí y con el entorno en el que habitan.

La arquitectura abstracta de un agente se puede formalizar teniendo en cuenta las propiedades de un entorno comentadas anteriormente y la interacción de un agente con el mismo [Woo01]. De este modo, un entorno se puede definir como un conjunto $S = \{s_1, s_2, \ldots, d_n\}$ de estados del entorno. En un momento concreto, se asume que el entorno se encuentra en un determinado estado. Por otra parte, la capacidad de interacción de un agente con el entorno se puede representar como un conjunto $A = \{a_1, a_2, \ldots, a_n\}$ de acciones. Por lo tanto, un agente se puede ver como una función

$$\text{acción} : S* \rightarrow A$$

que traduce secuencias de estados del entorno en acciones. Esta aproximación de *agente estándar* se complementa teniendo en cuenta que un agente decide qué acción realizar en base a un histórico, es decir, dependiendo de sus experiencias previas:

$$h : s_0 \rightarrow^{a_0} s_1 \rightarrow^{a_1} s_2 \rightarrow^{a_2} s_3 \rightarrow^{a_3} \cdots \rightarrow^{a_{u-1}} s_u \rightarrow^{a_u} \ldots$$

donde s_0 representa el estado inicial del entorno, a_u la *u-ésima* acción que el agente realizó y s_u el *u-ésimo* estado del entorno.

Existen ciertos tipos de agentes que deciden qué hacer sin tener en cuenta su histórico, es decir, basan su decisión únicamente en los eventos presentes omitiendo el pasado. Estos agentes se denominan agentes **puramente reactivos** [Woo01] debido a que responden directamente al entorno. Formalmente, el comportamiento de un agente puramente reactivo se puede representar por la siguiente función:

$$\text{acción} : S \rightarrow A.$$

Por ejemplo, se puede pensar en un agente puramente reactivo que controle el termostato de una caldera:

$$\text{acción}(s) = \begin{cases} \text{calor off} & \text{si s >= temperatura OK} \\ \text{calor on} & \text{en otro caso} \end{cases}$$

El siguiente paso en el diseño de un agente consiste en cómo diseñar la función *acción*. Para ello, hay que tener en cuenta que la arquitectura de un agente consiste en un mapa del estado interno del mismo: estructuras de datos, operaciones a realizar sobre dichas estructuras y el flujo de control entre las mismas. A continuación se resumirán algunos aspectos de alto nivel. El primero de ellos es la separación de la función de decisión del agente en dos subsistemas: **percepción** y **acción** (ver figura 2.1).

De este modo, se distinguen dos funciones principales. Por una parte *ver*, la cual genera una *percepción* y que puede estar implementada en hardware, como por ejemplo en el caso de la videovigilancia. La función *ver*

$$\text{ver} : S \rightarrow P$$

traduce estados del entorno en percepciones mientras que, por otra parte, la función *acción*

$$\text{acción} : P* \rightarrow A$$

traduce secuencias de percepciones en acciones.

Hasta ahora la función de decisión de un agente se ha modelado a partir de una secuencia de estados del entorno o percepciones generando unas determinadas acciones. Esto permite representar agentes cuya toma de decisiones esté influenciada por el histórico, es decir, la idea consiste en manejar **agentes con estado** (ver figura 2.2).

Estos agentes mantiene una estructura de datos interna, típicamente usada para registrar información sobre el entorno y el histórico. Sea I el conjunto de estados internos del agente. El proceso de toma de decisiones del agente estará basado, al menos en parte, en esta información. La función de percepción *ver* para un agente basado en estado permanece sin cambios, traduciendo estados del entorno en percepciones:

$$\text{ver} : S \rightarrow P$$

La función *acción* se define ahora como

$$\text{acción} : I \rightarrow A$$

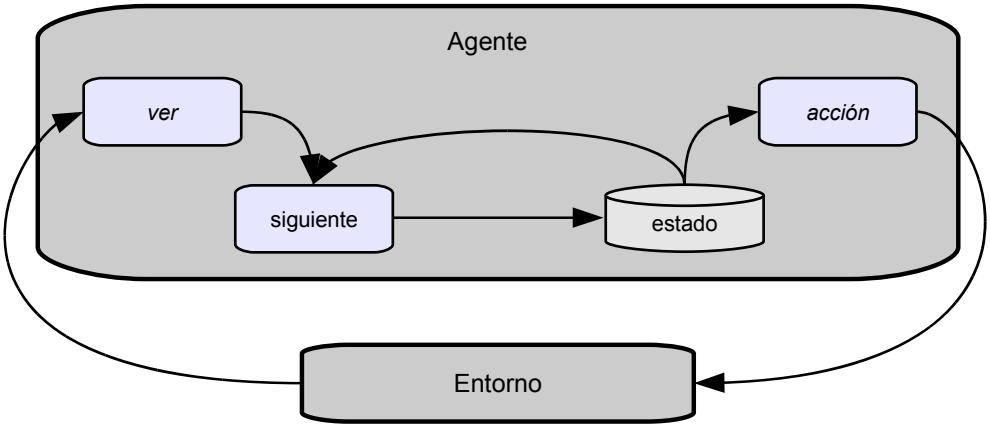


Figura 2.2: Agente que mantiene un estado.

traduciendo estados internos en acciones. Así mismo, se incluye una nueva función *siguiente*, que traduce estado interno y percepción en estado interno:

$$\text{siguiente} : IxP \rightarrow I$$

En resumen, el comportamiento de un agente basado en estado parte de un estado interno inicial i_0, para observar el estado del entorno s y generar una percepción *ver*(s). Entonces, el estado interno del agente se actualiza mediante la función *siguiente*, es decir, *siguiente*$(i_0,$ *ver*$(s))$. Finalmente, la acción seleccionada por el agente es *acción*$($*siguiente*$(i_0,$ *ver*$(s)))$. Cuando se ejecuta dicha acción, el agente comienza un nuevo ciclo.

Arquitecturas concretas para agentes inteligentes

Arquitecturas deliberativas

Las arquitecturas deliberativas son aquellas que utilizan modelos de representación simbólica del conocimiento [Mas05]. Normalmente, estas arquitecturas están basadas en la teoría clásica de la planificación, de manera que los agentes parten de un estado inicial y son capaces de generar planes para alcanzar sus objetivos.

Por lo tanto, la aproximación más común consiste en dotar a los agentes de los mecanismos de planificación necesarios para definir los pasos que han de seguir para alcanzar sus objetivos. De este modo, un agente deliberativo se entiende como aquél que maneja un modelo simbólico del mundo, el cual está representado de manera explícita, en el que las decisiones se toman atendiendo a los mecanismos de razonamiento lógico basados en la correspondencia de patrones y la manipulación simbólica.

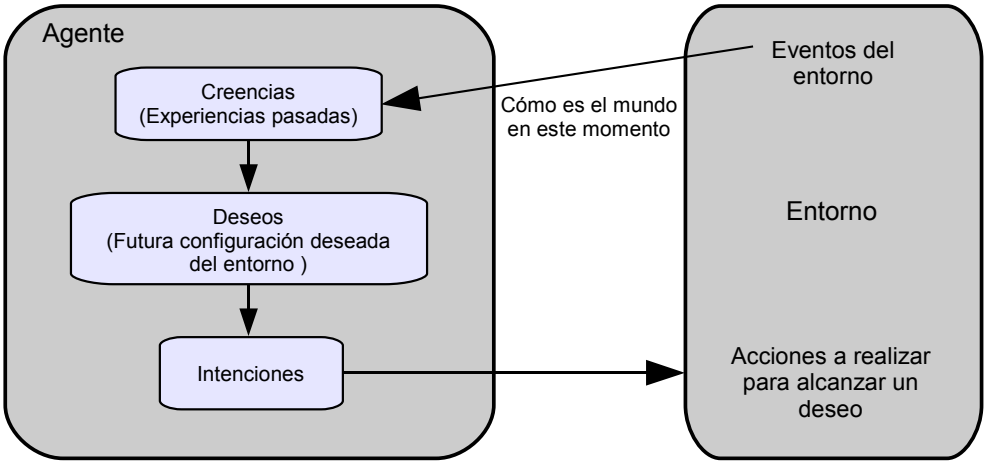


Figura 2.3: Arquitectura BDI.

La arquitectura deliberativa más estudiada y, posiblemente, más extendida es la **arquitectura BDI** o *Belief, Desire, Intention* [RG95] (ver figura 2.3). Dicha arquitectura está caracterizada por el hecho de que los agentes que la implementan se definen a través de estados mentales de creencias, deseos e intenciones. El éxito de esta arquitectura se debe a que combina un modelo filosófico del razonamiento humano fácil de comprender, un número considerable de implementaciones y una semántica lógica abstracta y elegante.

El modelo BDI es adecuado para proporcionar soluciones en entornos dinámicos o inciertos, en los que el agente no tiene una vista completa del problema, es decir, en los que el acceso a la información es limitado, y se maneja un número limitado de recursos. Las creencias, los deseos, las intenciones y los planes son clave en este tipo de sistemas.

Arquitecturas reactivas

La principal característica de una arquitectura reactiva es que no tiene asociado como elemento central de razonamiento un modelo simbólico y no utiliza un razonamiento simbólico complejo. Posiblemente, uno de los ejemplos más extendidos de este tipo de arquitecturas es la propuesta de R. Brooks [Bro92], conocida como **arquitectura de *subsunción***. Dicha arquitectura no está basada en ningún modelo simbólico y se basa en la idea de que la inteligencia es una propiedad emergente que permite generar comportamientos inteligentes. Las arquitecturas de subsunción manejan jerarquías de tareas que definen un comportamiento y suelen estar organizadas en capas de menor a mayor nivel de abstracción.

La mayor fuente de aplicaciones de las arquitecturas reactivas es la robótica [NSG+06] [ZS06] [Bui08]. Los robots se pueden considerar como agentes reales que actúan en un entorno cambiante. De hecho, la necesidad de interactuar con un entorno altamente impredecible y dinámico dificulta el uso de una arquitectura deliberativa que esté basada en la planificación de tareas.

Existen distintos tipos de arquitecturas de control reactivas, pero las más relevantes son dos. La primera de ellas se basa en alcanzar objetivos complejos a partir de un conjunto de acciones simples y, opcionalmente, de un modelo interno del mundo sobre el que razonar. La segunda de ellas se basa en utilizar únicamente resultados intermedios de razonamiento sobre la representación del entorno para resolver problemas complejos.

Arquitecturas híbridas

Debido a que tanto las arquitecturas deliberativas como las reactivas padecen ciertas limitaciones, se han propuesto una serie de arquitecturas híbridas que pretende aunar las ventajas de los dos enfoques. Una primera aproximación podría consistir en construir un agente compuesto de dos subsistemas: uno deliberativo, utilizado para generar planes a partir de un modelo simbólico, y otro reactivo, basado en un mecanismo de razonamiento que no sea complejo y que posibilite reaccionar ante eventos que provengan del entorno exterior.

De manera natural, este tipo de arquitecturas son adecuadas para una división en capas, ya sea vertical u horizontal. En este sentido, una estructuración horizontal permitiría el acceso a los sensores y a los actuadores por parte de todas las capas de la arquitectura. La mayoría de las arquitecturas de este tipo implementan un sistema de tres niveles [Fer92] [MP93]:

- **Reactivo**, o de toma de decisiones a partir de eventos.
- **Conocimiento**, o de representación simbólica del entorno.
- **Social**, o de modelado de la interacción con otros agentes, los deseos, las intenciones, etc.

Aplicación de la tecnología de agentes

El principal tipo de problemas que se pueden abordar con agentes son, de manera general, los sistemas reactivos, es decir, aquellos sistemas que mantienen una interacción continua con algún tipo de entorno. Dichos sistemas son más difíciles de diseñar e implementar que los sistemas funcionales, los cuales reciben una entrada de datos, la procesan y generan unos resultados como salida. Los sistemas reactivos se pueden dividir en tres grandes ramas [JW98]:

- **Sistemas abiertos**. Sistemas capaces de cambiar de manera dinámica, compuestos de componentes no conocidos a priori que pueden cambiar con el tiempo, y altamente heterogéneos. Un ejemplo típico podría ser Internet, que puede entenderse como una enorme fuente de información distribuida compuesta por nodos diseñados e implementados por

distintas organizaciones y con una constante interacción por parte de los usuarios. Dicha funcionalidad está directamente ligada a técnicas basadas en la negociación y la cooperación, características fuertemente arraigadas al dominio de los sistemas multi-agente.

- **Sistemas complejos**. Dos de las herramientas más potentes para tratar con la complejidad en el desarrollo de software son la modularidad y la abstracción. En este contexto, los agentes representan una herramienta muy potente para diseñar sistemas modulares. Por ejemplo, ante un sistema complejo parece lógico diseñar componentes modulares que estén especializados en las distintas funcionalidades ofrecidas por dicho sistema. En estos dominios, un enfoque basado en agentes permite abordar el problema completo como una serie de problemas de menor tamaño y complejidad, más fáciles de desarrollar y mantener. Además, la noción de agente autónomo proporciona una abstracción muy útil para conceptualizar un sistema software complejo como una sociedad de elementos que solucionan problemas de manera cooperativa.
- **Sistemas de computación ubicua**. El concepto de ubicuidad está directamente relacionado con el tratamiento de información de manera distribuida, tarea que se adapta perfectamente a las características clásicas de un agente: autonomía, proactividad, reactividad y adaptación. El componente inteligente cobra especial importancia a la hora de diseñar soluciones que mejoren la interacción entre las personas y las máquinas.

Limitaciones

Aunque la tecnología de agentes juega un rol importante en el desarrollo de aplicaciones, una solución basada en agentes no implica necesariamente el planteamiento más adecuado. La propia naturaleza de un paradigma basado en agentes conlleva una serie de problemas comunes a las aplicaciones basadas en agentes [JW98]:

- Control del sistema no global. Una solución basada en agentes puede no ser apropiada para dominios con restricciones globales, de tiempo real o en los que se deban evitar a toda costa los interbloqueos.
- Perspectiva no global. Debido a que las acciones de los agentes están determinadas por su propio estado, se ha de llevar a cabo un estudio de la toma de decisiones óptima de manera global.
- Confianza y delegación. Es necesario implementar mecanismos de confianza y delegación en relación a los individuos o entidades que intervienen en el tratamiento de información.

Ámbitos de aplicación

A continuación se comentarán las principales áreas de aplicación de la tecnología de agentes. En primer lugar destacan las aplicaciones industriales, siendo el control de procesos uno de los principales campos de desarrollo. De hecho, el control de procesos es una aplicación natural de los agentes inteligentes y de los sistemas multi-agente, ya que los controladores de procesos se pueden considerar como sistemas autónomos reactivos.

Uno de los más clásicos es ARCHON [JCL95], una plataforma software para construir sistemas multi-agente asociada a una metodología para la construcción de aplicaciones con dicha plataforma. Dentro de las aplicaciones industriales, también destacan los trabajos en fabricación y en el control del tráfico aéreo [Mas05].

Otro ámbito de aplicación relevante son las aplicaciones comerciales, en las que destacan las de gestión de información y de comercio electrónico. Dentro de esta última área de trabajo, los agentes inteligentes han cobrado especial importancia [LWJ03] [CSMGMG04]. Otros dos campos generales en los que la tecnología de agentes también se ha utilizado son el médico, en el que destacan las aplicaciones de monitorización de pacientes [CBdPT08], y en el entretenimiento, como por ejemplo los videojuegos [Mae95].

2.1.3. Sistemas Multi-Agente

El hecho de desarrollar aplicaciones complejas compuestas de distintos sub-sistemas que colaboran entre sí obliga a distribuir la inteligencia entre diversos agentes, y a construir **Sistemas Multi-Agente** [Wei99] [Woo01] que permitan la gestión inteligente de un sistema complejo. Dicha gestión se logra mediante la coordinación de los distintos subsistemas que lo componen e integrando los objetivos particulares de cada subsistema en un objetivo común.

Si cada subsistema tiene una capacidad de decisión local, el problema de la gestión se puede abordar definiendo políticas de cooperación, coordinación y negociación entre agentes. En general, un sistema multi-agente cooperante presentará las siguientes características [WHRB+88]:

- Está formado por un conjunto de agentes, cada uno de los cuales mantiene sus propias habilidades.
- El sistema multi-agente tiene una misión común, la cual puede descomponerse en diferentes tareas independientes, de forma que se puedan ejecutar en paralelo.
- Cada agente del sistema tiene un conocimiento limitado.

- Cada agente del sistema tiene cierta especialización para realizar determinadas tareas, en función de lo que conoce, la capacidad del proceso y la habilidad requerida.

Por otra parte, la distribución de las decisiones entre los distintos nodos del sistema permite fundamentalmente:

- Mantener la autonomía de cada agente.
- Eliminar la necesidad de que toda la información del sistema se encuentre en un único agente.
- Descentralizar la toma de decisiones.
- Llevar a cabo acciones coordinadas.
- Organizar dinámicamente la arquitectura de agentes, lo que permite su adaptación a los cambios que se produzcan en el entorno en tiempo real.

Los agentes operan y existen en un entorno, el cual típicamente tiene una parte computacional y una física. Como se mencionó en la sección anterior, los entornos pueden ser abiertos o cerrados y pueden tener o no más de un agente. Sin embargo, y debido en parte a la proliferación de la infraestructura de comunicación, la situación más normal para un agente consiste en interactuar con otros agentes. Por este motivo, el número de agentes puede ser en ocasiones lo suficientemente elevado como para tratar con ellos individualmente, surgiendo el concepto de sociedad de agentes para gestionarlos de manera colectiva.

Comunicación entre agentes

Los agentes se comunican para alcanzar sus objetivos individuales y el objetivo global de la sociedad a la que pertenecen de la mejor manera posible. De este modo, la comunicación juega un papel muy importante en el comportamiento de los agentes a la hora de coordinar sus acciones, obteniendo sistemas más coherentes. Un ejemplo de interacción entre agentes podría consistir en solicitar la realización de una acción a un determinado agente especializado por parte de otro. Por supuesto, el primero de ellos tendría la autonomía suficiente para aceptar la petición, en caso de que dicha acción conlleve un beneficio para el propio agente o para la sociedad, o rechazarla.

La **coordinación** es una propiedad de los SMA que posibilita el desarrollo de una acción en un entorno compartido. El grado de coordinación determinará hasta que punto el sistema es capaz de evitar acciones innecesarias o ajenas para minimizar el gasto de recursos, evitar situaciones de bloqueo y mantener unas determinadas condiciones de seguridad.

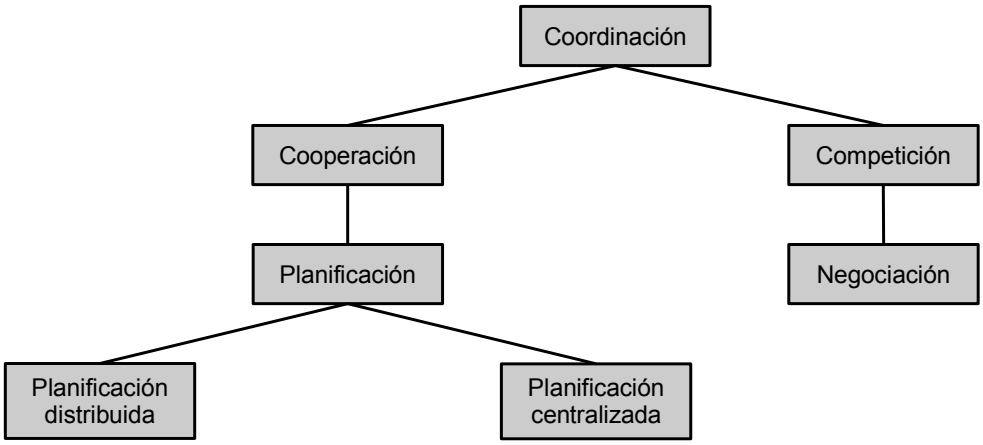


Figura 2.4: Clasificación de las principales vías de coordinación entre agentes.

En este contexto, la **cooperación** surge como la interacción entre agentes que no se oponen entre sí, es decir, que de alguna manera comparten un mismo objetivo, mientras que la negociación se refiere a la coordinación entre agentes competitivos o simplemente interesados en su propio beneficio. Típicamente, para que la cooperación entre agentes tenga éxito, éstos han de mantener un modelo interno del resto de agentes y un modelo para llevar a cabo futuras interacciones.

Existen tres aspectos principales para estudiar de una manera formal la comunicación entre agentes [Wei99]: i) la **sintaxis**, que especifica cómo se estructuran los símbolos de la comunicación, ii) la **semántica**, que se refiere a qué significan los símbolos, y iii) la **pragmática**, que alude a cómo se interpretan los símbolos. El significado se puede definir como la combinación de la semántica y la pragmática. A la hora de comunicarse, los agentes han de ser capaces de entender los mensajes que reciben, por lo que es importante tener en cuenta estos tres aspectos de la comunicación.

Debido a que normalmente los agentes que forman parte de un mismo sistema multi-agente tienen distintas capacidades a la hora de comunicarse, es importante definir un esquema de comunicación a varios niveles. Normalmente, se identifican distintos roles en una conversación, destacando los de *activo*, *pasivo* o ambos. Así mismo, existen dos tipos principales de mensajes: *afirmaciones* y *preguntas*.

Protocolos de interacción entre agentes

Mientras que la comunicación entre agentes describe los mecanismos utilizados para enviar y recibir mensajes simples, los protocolos de interacción especifican cómo se lleva a cabo el intercambio de dichos mensajes, es decir, definen las reglas de una conversación entre agentes. En la literatura exis-

Acto de comunicación	Intención	Resultado esperado
Afirmación	Informar	Aceptación
Solicitud	Preguntar	Respuesta
Respuesta	Informar	Aceptación
Solicitud	Pedir	
Explicación	Informar	Acuerdo
Orden	Pedir	
Permiso	Informar	Aceptación
Rechazo	Informar	Aceptación
Oferta	Informar	Aceptación
Aceptación		
Acuerdo		
Propuesta	Informar	Oferta
Confirmación		
Retirada		
Denegación		

Tabla 2.1: Principales tipos de mensajes entre agentes.

ten diversos protocolos de interacción entre agentes. En el caso de que los agentes tengan objetivos que entren en conflicto o simplemente sean agentes interesados en su propio beneficio, el objetivo de los protocolos de interacción es maximizar la utilidad de los agentes [RZ94]. Por el contrario, en el caso de que los agentes compartan unos objetivos comunes, el objetivo de los protocolos de interacción es mantener un rendimiento coherente desde una perspectiva global pero sin perjudicar la autonomía de los agentes, es decir, sin un control global explícito [Dur88].

En un entorno con recursos limitados, los agentes deben coordinar entre sí sus actividades para alcanzar sus propios objetivos o satisfacer metas comunes. Las acciones de varios agentes han de coordinarse porque normalmente existe una dependencia entre ellas, debido a la necesidad de satisfacer unas restricciones globales y a que ningún agente tiene la capacidad, los recursos o la información necesaria para alcanzar dichas metas globales. Ejemplos clásicos de coordinación son proporcionar la información oportuna a otros agentes, asegurar que las acciones de los agentes están sincronizadas y evitar la solución redundante de problemas.

Con el objetivo de obtener sistemas coordinados, gran parte de la investigación en el ámbito de la Inteligencia Artificial Distribuida ha sido el desarrollo de técnicas para dividir el control y los datos. Esta acción implica que los agentes tengan la suficiente autonomía para tomar sus propias decisiones y generar las acciones oportunas para alcanzar unos determinados objetivos. Sin embargo, la principal desventaja de este enfoque es que el conocimiento global del sistema está disperso, de manera que cada agente mantiene una visión parcial e imprecisa del problema global.

Por otra parte, dentro del ámbito de la cooperación, la estrategia básica de muchos de los protocolos de interacción existentes consiste en descom-

poner y distribuir tareas, es decir, en seguir un método basado en el enfoque *divide y vencerás*. Esta alternativa puede reducir la complejidad de una tarea, dividiéndola en sub-tareas de menor complejidad que requieran agentes menos capacitados y menos recursos. Sin embargo, el sistema ha de decidir cómo llevar a cabo la descomposición teniendo en cuenta ambos factores (capacidades y recursos) y las dependencias existentes entre sub-tareas.

La descomposición de tareas la puede realizar el propio diseñador del sistema, durante la fase de implementación, o se puede delegar en agentes que hagan uso de una planificación jerárquica. Dicha descomposición puede estar basada en la distribución espacial de la información o en la funcionalidad, de acuerdo a las capacidades de los agentes que componen el sistema multi-agente. Una vez que se ha realizado este proceso, las tareas se pueden distribuir de acuerdo a los siguientes criterios generales [DLC87]:

- Evitar la sobrecarga de los recursos críticos.
- Asignar tareas a agentes que tengan la capacidad para realizarlas.
- Permitir que un agente con perspectiva global asigne tareas a otros agentes.
- Asignar responsabilidades que se solapen a distintos agentes para alcanzar cierto grado de coherencia.
- Asignar tareas que sean dependientes entre sí a agentes próximos espacial o semánticamente con el objetivo de minimizar los costes de comunicación y sincronización.
- Reasignar tareas que sean urgentes para completarlas si así es necesario.

Los siguientes mecanismos se utilizan comúnmente para distribuir tareas:

- Mecanismos de mercado, en los que las tareas se asignan a ciertos agentes en función de acuerdos generalizados.
- Protocolo *contract net*, que consiste en una serie de ciclos de anunciamiento, puja y recompensa (ver figura 2.5 izquierda).
- Planificación multi-agente, realizada por agentes responsables de asignar tareas.
- Estructura organizacional, en las que los agentes tienen responsabilidades fijas para tareas particulares.

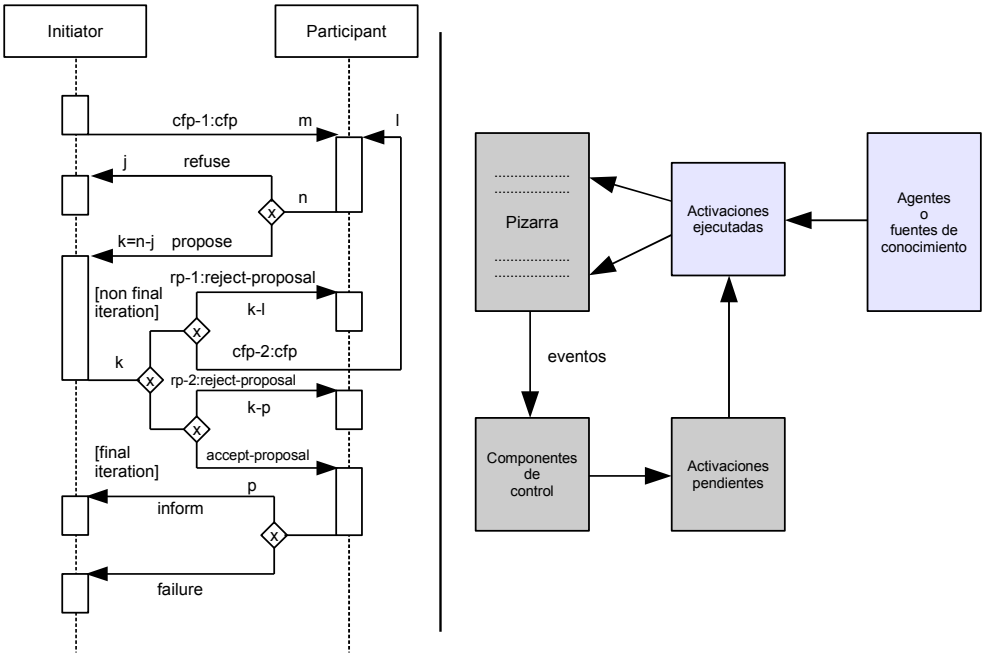


Figura 2.5: Izquierda: protocolo de interacción *FIPA Iterative Contract Net*.
Derecha: Arquitectura de un sistema de pizarra básico.

Sistemas de pizarra

La solución de problemas mediante un sistema de pizarra se basa en la cooperación de un grupo de agentes expertos en distintas tareas que hacen uso de un repositorio común de conocimiento, el cual viene determinado por una pizarra (ver figura 2.5 derecha). Básicamente, la pizarra representa el punto de encuentro entre los agentes y sirve para ir anotando las soluciones parciales que conforman la solución global a un problema determinado. La resolución del problema comienza cuando el propio problema y los datos iniciales se escriben en la pizarra.

En este punto, los distintos especialistas comienzan a observar la pizarra con el objetivo de utilizar el conocimiento anotado en la misma para aplicar su especialidad y contribuir de este modo a la solución final. Cuando un especialista recopila la suficiente información para hacer una contribución, la desarrolla y la anota en la pizarra. Esta nueva información o conocimiento posibilita que a su vez otro experto la utilice. Este proceso continúa hasta que se alcanza la solución final.

Este modo de trabajo refleja las siguientes características deseables de un sistema de pizarra [Wei99]:

- **Independencia de habilidades**. Cada agente o fuente de conocimiento no se entrena para trabajar exclusivamente con otro conjunto determi-

nado de agentes. De este modo, cada agente es un experto en una o varias tareas y puede contribuir a la solución final de manera independiente al resto de agentes.

- **Diversidad en las técnicas de solución**. La representación interna y el mecanismo de inferencia usado por cada fuente de conocimiento están ocultos.
- **Representación flexible de la información de la pizarra**. El modelo de pizarra no establece restricciones en el tipo de información a anotar en la misma.
- **Lenguaje de interacción común**. Las fuentes de conocimiento de los sistemas de pizarra deben ser capaces de interpretar correctamente la información registrada en la pizarra por otras fuentes.
- **Activación basada en eventos**. Las fuentes de conocimiento actúan en respuesta a eventos asociados a la pizarra. En lugar de escanearla periódicamente, se establece un mecanismo por el cual la pizarra notifica a las fuentes de información los eventos en los cuales están interesadas.
- **Necesidad de control**. Ha de existir una fuente de conocimiento encargada de gobernar el curso del problema, considerando las anotaciones que pueden ser más útiles para alcanzar la solución global al problema.
- **Generación incremental de soluciones**. Las fuentes de conocimiento contribuyen a la solución final desde distintas perspectivas: subsolución apropiada, incorrecta, contradictoria o incluso una que inicie una nueva línea de razonamiento.

2.1.4. Estándares y herramientas para el desarrollo de Sistemas Multi-Agente

El conjunto de estándares de FIPA

FIPA[1] (***The Foundation for Intelligent Physical Agents***) es un comité del IEEE para la promoción de la tecnología basada en agentes y en la interoperabilidad de sus estándares con otras tecnologías. Las especificaciones FIPA representan un conjunto de estándares cuyo objetivo es promover la interacción entre agentes heterogéneos y los servicios que representan los mismos.

FIPA mantiene un ciclo de vida para sus distintas especificaciones, compuesto por las etapas *Preliminary*, *Experimental*, *Standard*, *Deprecated*, y *Obsolote*. A cada especificación se le asigna un identificador único, en función del estado en el que se encuentran dentro del ciclo de vida. A continuación

[1] http://www.fipa.org

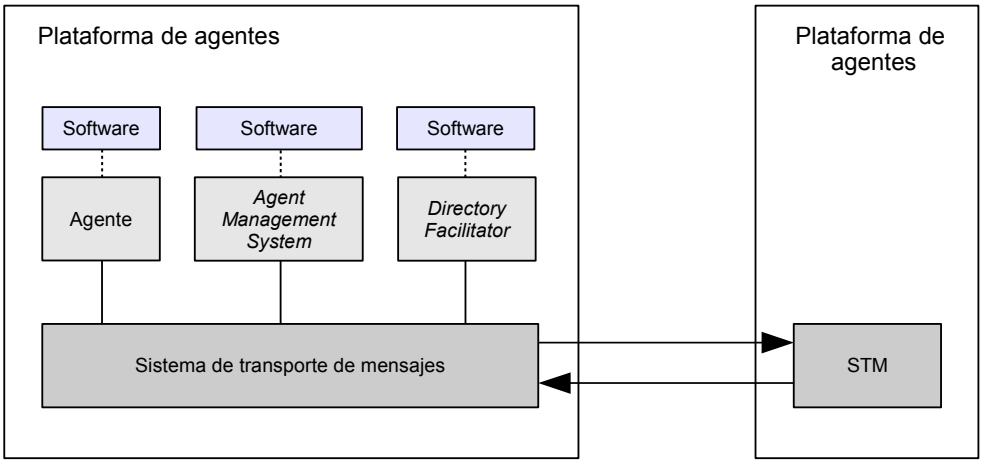


Figura 2.6: Modelo de referencia de FIPA para el desarrollo de sistemas multi-agente.

se irán comentando alguno de los documentos más importantes, que definen las partes principales de la arquitectura propuesta por FIPA.

Uno de los principales documentos es el **FIPA *Abstract Architecture Specification*** [Fou02a]. Dicho documento, y las especificaciones derivadas del mismo, definen la arquitectura abstracta propuesta por FIPA (ver figura 2.6). Las partes de dicha arquitectura incluyen:

- Una especificación que define elementos arquitectónicos y sus relaciones.
- Guías para la especificación de sistemas de agentes en lo que se refiere a software en concreto y a tecnologías de comunicación (guías para la instanciación).
- Especificaciones que gobiernan la interoperabilidad y conformidad de los agentes y los sistemas multi-agente (guías para la interoperabilidad).

El propósito principal de este documento es garantizar la interoperabilidad y la reusabilidad. Para lograr este propósito es necesario identificar los elementos de la arquitectura que deben codificarse. En concreto, si dos o más sistemas usan distintas tecnologías para alcanzar algún propósito funcional es necesario identificar las características comunes de los distintos enfoques. Este hecho lleva a la identificación de abstracciones arquitectónicas, es decir, a diseños abstractos que puedan ser formalmente trasladados a cualquier implementación válida.

Si se describen los sistemas de forma abstracta, se pueden explorar las relaciones entre los elementos principales de los sistemas multi-agente. Además, describiendo las relaciones entre estos elementos se llega a obtener de

forma clara cómo los sistemas multi-agente pueden crearse y cómo son capaces de interactuar. De este conjunto de elementos arquitectónicos y sus relaciones se puede obtener un amplio conjunto de posibles arquitecturas concretas, que pueden interoperar porque comparten un diseño abstracto.

De una forma más específica, el objetivo de este documento es definir el intercambio de mensajes entre agentes, los cuales pueden utilizar distintos protocolos de transporte de mensajes, distintos lenguajes de comunicación entre agentes y diferentes lenguajes de contenido. El propósito de esta arquitectura incluye:

- Un modelo de servicios y de descubrimiento de servicios disponibles a los agentes y a otros servicios.
- Interoperabilidad en el transporte de mensajes.
- Soporte para varias formas de representación de lenguajes de comunicación entre agentes.
- Soporte para varias formas de lenguajes de contenido.
- Soporte para múltiples representaciones de servicios de directorio.

Otro de los documentos que constituye el pilar principal de FIPA es el **FIPA *Agent Management Specification*** [Fou04]. Dicho documento contiene especificaciones para la gestión de los agentes, incluyendo servicios de gestión, ontologías y transporte de mensajes dentro de la plataforma de agentes. Además, este documento está especialmente vinculado a la definición de interfaces estándares abiertas para el acceso a los servicios de gestión de agentes. El diseño interno, la implementación y la infraestructura de la gestión de los agentes quedan al margen de este documento.

El modelo de referencia para la gestión de agentes contiene entidades que representan conjuntos de capacidades lógicas (es decir, servicios), y que no implican configuraciones físicas. Además, los detalles de implementación de los agentes y de las plataformas de agentes son decisiones particulares del desarrollador. Dicho modelo de referencia consiste en distintos componentes lógicos, cada uno de los cuales representa un conjunto de capacidades.

Un **agente**, según FIPA, es un proceso computacional que implementa la funcionalidad de comunicación y la autonomía de una aplicación. Los agentes se comunican empleando un lenguaje de comunicación de agentes. Además, un agente se puede concebir como el actor fundamental en la plataforma de agentes, y que combina una o más capacidades (servicios) dentro de un modelo de ejecución integrado. La noción de identidad para un agente es el identificador de agente o *AID*), que identifica a un agente de manera unívoca dentro de la plataforma de agentes.

El modelo de referencia de nombrado de agentes identifica un agente a través de una colección extensible de parejas parámetro-valor, denominada

Agent Identifier (AID). La extensible naturaleza del AID permite aumentar de una forma cómoda el número de parámetros de dicho elemento. Por defecto, el AID contiene los siguientes parámetros:

- Un parámetro *name*, que es el identificador único y global que se emplea como expresión para referirse al agente.
- Un parámetro *addresses*, consistente en una lista de direcciones de transporte donde se pueden entregar los mensajes a dicho agente.
- Un parámetro *resolvers*, que constituye una lista de direcciones de servicios de resolución de nombres.

El ***Directory Facilitator*** es un componente opcional de la plataforma de agentes, pero que si está presente debe implementarse como un tipo específico de servicio. Dicho componente es el encargado de proporcionar un servicio de páginas amarillas a los agentes, pudiendo existir más de uno dentro de una plataforma de agentes. Las principales funciones de este servicio son las de añadir y eliminar un registro (que representa la descripción de la funcionalidad de un agente), modificación y búsqueda.

El ***Agent Management System*** es un componente obligatorio de la plataforma de agentes, que actúa como controlador de dicha plataforma. Además, sólo se permite un AMS por plataforma de agentes. Este servicio mantiene un directorio con los AIDs de los agentes registrados en la plataforma. El AMS proporciona un servicio de páginas blancas a los agentes, y es necesario que cada agente se registre en el AMS para obtener un AID válido.

Dicho componente representa la autoridad de gestión en la plataforma de agentes, y es responsable de todas las acciones de gestión relacionadas con los agentes. El AMS proporciona las funciones de gestión asociadas al registro de agentes en la plataforma, eliminación del registro, modificación, búsqueda y obtención de descripciones. Además, el AMS tiene la capacidad para llevar a cabo las acciones de suspensión de agentes, terminación, creación, resumen, invocación, ejecución y gestión de recursos.

El ***Message Transport Service*** [Fou00a] representa el método de comunicación por defecto entre los agentes de distintas plataformas de agentes. El modelo de referencia para el transporte de mensajes entre agentes está asociado con tres niveles:

- El *Message Transport Protocol* (MTP) permite llevar a cabo el transporte físico de mensajes entre dos *Agent Communication Channels* (ACCs), entidades que proporcionan el servicio directo de transporte a los agentes.
- El *Message Transport Service* (MTS) es un servicio de la plataforma de agentes a la que un agente está vinculado. El MTS soporta el transporte de mensajes FIPA ACL entre agentes de una misma o distintas plataformas de agentes.

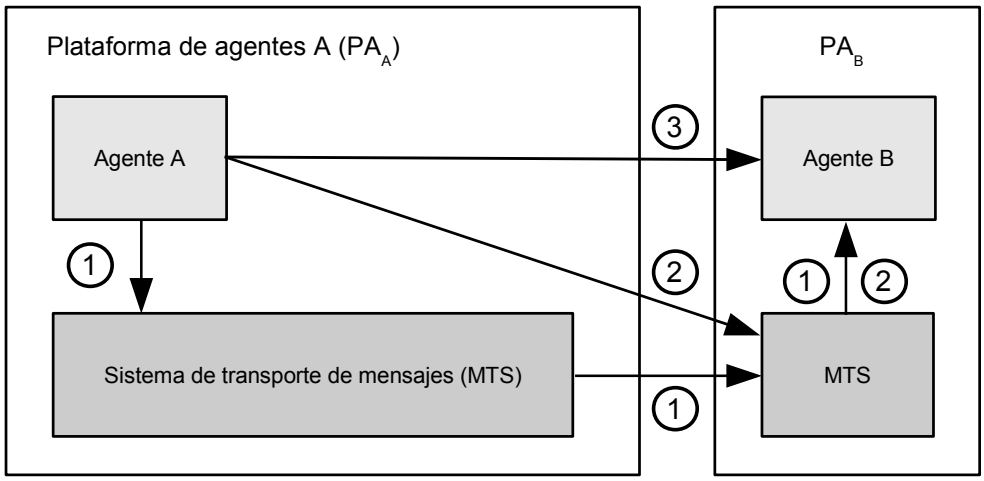


Figura 2.7: Opciones de envío de mensajes entre agentes FIPA: i) mediante MTSs en ambas plataformas, ii) mediante el MTS de la plataforma en la que habita el agente receptor, iii) comunicación directa entre agentes.

- El *Agent Communication Language* (ACL) representa la carga útil de los mensajes intercambiados entre el MTS y el MTP.

Un agente tiene diversas opciones a la hora de enviar mensajes a agentes de otras plataformas (ver figura 2.7):

1. El agente A envía un mensaje a su ACC local usando una interfaz estándar propietaria. Éste se encarga de enviar el mensaje al ACC remoto a través del MTP. Por último, el ACC remoto entregará el mensaje a su destinatario.
2. El agente A envía un mensaje directamente al ACC remoto, el cual entregará el mensaje al agente B.
3. El agente A envía directamente el mensaje al agente B.

En cuanto al ciclo de vida de un agente, se debe tener en cuenta que un agente existe físicamente en la plataforma de agentes y utiliza la funcionalidad que ofrecen los distintos servicios de la plataforma. Dentro de este contexto, un agente, como proceso software, mantiene un ciclo de vida físico gestionado por la plataforma de agentes.

Frameworks para el desarrollo de sistemas multi-agente

Debido a la proliferación de los sistemas multi-agente como herramienta para solucionar problemas complejos en entornos heterogéneos y distribuidos, existen ciertas plataformas software creadas para facilitar el desarrollo

de este tipo de sistemas. Bordini et al. presentaron una revisión tanto de lenguajes de programación como plataformas para el despliegue de sistemas multi-agente [BBD+06].

En este contexto, **JADE** (Java Agent DEvelopment Framework) [BCPR08], representa el *framework* más maduro existente en la actualidad y, después de 10 años de desarrollo, constituye la referencia para el desarrollo de sistemas multi-agente de acuerdo al conjunto de estándares definidos por FIPA. Sin embargo, también se han desarrollado otros plataformas de agentes que han contribuido a la expansión de este tipo de herramientas. Las más relevantes se resumen en la tabla 2.2.

Desde un punto de vista práctico JADE está organizado en dos módulos:

1. Un sistema multi-agente de acuerdo a las especificaciones de FIPA.
2. Un paquete para el desarrollo de agentes con el lenguaje Java.

En JADE la comunicación se realiza mediante Java RMI [Dow98], imponiendo la restricción de usar Java como lenguaje de programación. Por lo tanto, los desarrolladores que hagan uso de JADE deben implementar sus agentes con Java siguiendo la guía de desarrollo asociada a JADE [BCG07]. Los agentes en JADE están implementados como hilos y se ejecutan en los denominados *agent containers*, los cuales proporcionan el entorno de ejecución para que los agentes puedan realizar sus tareas. Por otra parte, JADE ofrece una interfaz gráfica de administración para gestionar la plataforma multi-agente. Además, proporciona soporte para la movilidad, tanto de código como del estado de los agentes, ofreciendo el despliegue automático de los servicios de gestión de FIPA y la posibilidad de hacer uso de los protocolos de interacción definidos por dicho comité.

Existen otras plataformas inspiradas por JADE, como por ejemplo **Jadex** [PBL05], un *framework* de código abierto desarrollado para diseñar agentes guiados por metas basados en el modelo BDI. El principal objetivo de este proyecto es el desarrollo de sistemas basados en agentes de una manera fácil e intuitiva, proporcionando para ello un *middleware* diseñado en base a principios bien conocidos de la Ingeniería del Software. Jadex permite que los desarrolladores construyan agentes software utilizando XML y Java que, posteriormente, se pueden desplegar sobre determinados *middlewares* como JADE. Esta herramienta se ha usado en varios proyectos, como por ejemplo en la planificación de tratamientos para pacientes de un hospital [PZR+06] o en juegos [RS05].

Una alternativa a las herramientas descritas previamente es **Cougaar** [HW05], una arquitectura de código abierto programada en Java para desarrollar aplicaciones distribuidas basadas en agentes. Uno de los principales objetivos de este proyecto consiste en proporcionar una herramienta para el desarrollo de sistemas multi-agente flexibles y con una gran escalabilidad.

Nombre	Lenguaje	Estándar	Libre	Artículo relevante
JADE	Java	FIPA	Yes	[BCPR08]
Jadex	Java	FIPA	Yes	[PBL05]
Cougaar	Java	No	Yes	[HW05]
Agent Factory	Java	FIPA	Yes	[RCO04]
JACK	Java	No	No	[Win06]

Tabla 2.2: Plataformas de desarrollo de sistemas multi-agente relevantes en la actualidad.

Sin embargo, Cougaar no hace uso de ningún estándar conocido, por lo que la interoperabilidad del sistema se ve reducida notablemente.

Agent Factory [RCO04] representa otra alternativa relevante en este ámbito, y se puede entender como un *framework* extensible para la creación y el despliegue de aplicaciones basadas en agentes. Su principal objetivo es ofrecer soporte al desarrollo de sistemas distribuidos complejos en múltiples dominios de aplicación. Las principales características de este *framework* son la adopción de los estándares de FIPA, el uso de una filosofía *plug & play* para desplegar y configurar agentes de distintos tipos, el uso de Java como lenguaje de programación y la inclusión de los denominados intérpretes de agentes.

Finalmente, otro trabajo relevante es **JACK** [Win06], un entorno comercial para construir, ejecutar y desplegar sistemas multi-agente utilizando un modelo basado en componentes. Esta herramienta también proporciona un lenguaje basado en Java para soportar entidades del tipo agente, capacidad, evento o plan; además de un editor para asignar planes a los agentes.

2.1.5. Consideraciones finales

En esta sección no se ha presentado otra de las propuestas relacionadas con los sistemas multi-agente, las recomendaciones del grupo *Agent Platform Special Interest Group* (*Agent PSIG*) del OMG. La misión de este grupo consiste en interactuar con la plataforma OMG para estimular la creación de especificaciones de agentes [Obj08a] directamente vinculadas a la tecnología utilizada por el OMG.

Los objetivos más importantes de este grupo son la recomendación de extensiones para el uso de la tecnología de agentes, la promoción de lenguajes de modelado estándar para incrementar la consistencia y el soporte a desarrolladores para comprender cómo crear sistemas multi-agente.

2.2. Sistemas de Vigilancia Inteligente

2.2.1. Introducción

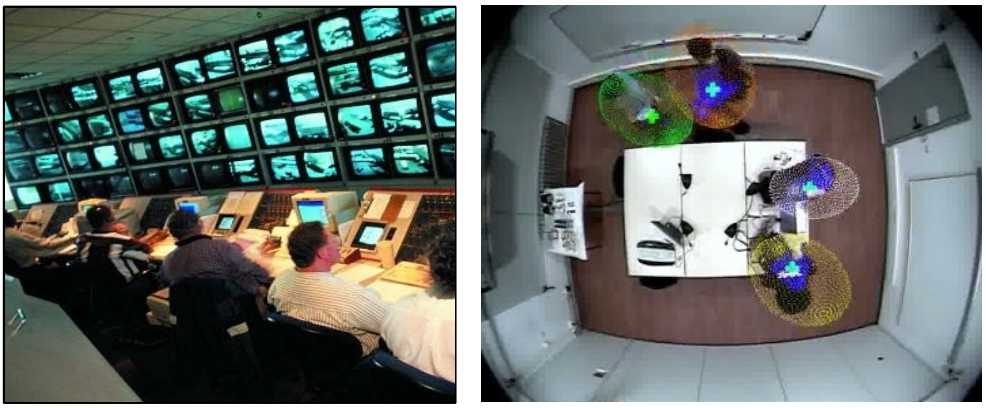
Lejos han quedado los días en los que los sistemas de vigilancia sólo eran capaces de procesar un único flujo de vídeo obtenido desde una cámara de vigilancia. Estos sistemas hacían uso de simples algoritmos que se evaluaban en entornos controlados con un reducido número de personas, las cuales se movían de una determinada manera para que el sistema obtuviera resultados aceptables. Sin embargo, los sistemas de vigilancia actuales plantean necesidades mucho más ambiciosas (en parte debido a las demandas de la sociedad por vivir en entornos más seguros) como la monitorización de escenas complejas mediante redes de sensores de vigilancia heterogéneos, la vigilancia en tiempo real o la identificación de los comportamientos llevados a cabo por los objetos móviles que intervienen en el entorno monitorizado (ver figura 2.8).

Actualmente, la **monitorización** de espacios, tanto públicos como privados, es una necesidad importante debido a dos factores principales: i) reducir el número de actividades delictivas y ii) desarrollar sistemas de vigilancia inteligentes para mejorar la calidad de vida de las personas en el día a día [RVFT07]. Una forma de apreciar esta expansión es conocer el número de cámaras CCTV (*closed-circuit television* o televisión de circuito cerrado) desplegadas internacionalmente [NMW04]. En este contexto, el Reino Unido es el gran líder. En 2009, el número aproximado de habitantes del Reino Unido se aproxima a los 62 millones, mientras que en el año 2005 el número de cámaras desplegadas en dicho país se estimó en torno a 4 millones [2]. Sin tener en cuenta el incremento del número de cámaras en estos últimos cinco años, el *ratio* es aproximadamente *1:15*, es decir, una cámara de vigilancia por cada 15 personas.

En la mayoría de grandes instalaciones con cámaras CCTV, con cientos de ellas, sólo una pequeña fracción de las mismas se vigilan. Este hecho se debe fundamentalmente a dos factores. Por una parte, el número de operarios es reducido con respecto al número de cámaras y, por otra, los flujos de vídeo almacenados por determinadas cámaras sólo se revisan en caso de que haya ocurrido algún tipo de incidente. Este último fenómeno se suele denominar **análisis forense**. Por lo tanto, la conclusión es que aunque en teoría todas las cámaras se monitoricen de alguna manera, aunque sea como parte de un procedimiento de prueba, sólo un pequeño porcentaje se hace en tiempo real.

Desde una perspectiva práctica, se conoce que un operador puede monitorizar realmente entre 1 y 4 cámaras simultáneamente [WD88], de manera que en una instalación con 100 cámaras y 3 operarios, sólo el 3 % de las cámaras se monitorizarán activamente en un momento determinado por una

[2] http://www.liberty-human-rights.org.uk/privacy/cctv.shtml



(a) (b)

Figura 2.8: (a) Centro de vigilancia tradicional. **(b)** Sistema de detección de tumultos (Proyecto HESPERIA [GHF+07]).

persona, sin tener en cuenta otros efectos como la fatiga y el cansancio que se comentarán posteriormente.

La **complejidad** de los espacios de vigilancia públicos y privados, determinada por las características de entornos tanto interiores como exteriores, es uno de los principales factores a la hora de monitorizar un entorno, ya que las características físicas del mismo pueden variar y la red de sensores no tiene por qué cubrir el entorno completo. En el caso de la videovigilancia, se podrían desplegar una gran cantidad de cámaras con una topología compleja que permitiera monitorizar todo el entorno, pero esta solución no es práctica debida a varias razones:

- Tanto el despliegue como la configuración hardware serían impracticables.
- Los costes derivados del personal de vigilancia se dispararían si se pretende realizar una vigilancia eficaz.

Así mismo, esta dependencia respecto a los operadores humanos lleva asociada la cuestión de la **fatiga** y el **cansancio**, derivados de la observación continua de este tipo de dispositivos de vigilancia [Smi04], así como la dificultad de tratar varias amenazas simultáneas cuando se presenta una situación caótica.

Por estos motivos, resulta esencial el desarrollo de algoritmos inteligentes que sean robustos y capaces de adaptarse a las situaciones derivadas del entorno a monitorizar con la mínima intervención humana. Esta nueva generación de sistemas de vigilancia inteligente ha de girar en torno a la autonomía, es decir, a la capacidad de actuar por sí mismos teniendo en cuenta el entorno a vigilar. Sin embargo, un sistema de vigilancia de este

Figura 2.9: Vista desde dos cámaras de vigilancia públicas (Creighton University - California. View of St. John's Church y Skinner Mall from Swanson Residence Hall).

tipo ha de reunir otras características importantes como las que se exponen a continuación:

- **Escalabilidad**, con el objetivo de que el sistema de vigilancia pueda expandirse en función de los requisitos del entorno a monitorizar.
- **Portabilidad**, para permitir que el sistema pueda adaptarse a distintos entornos y condiciones de funcionamiento.
- **Modularidad**, que permite diferenciar claramente qué partes del sistema proporcionan cada uno de los servicios del sistema de vigilancia, al mismo tiempo que se garantiza la interoperabilidad de los mismos.
- **Robustez**, para que el sistema sea tolerante a fallos y siga funcionando cuando se produzca cualquier tipo de error en la infraestructura que le da soporte.

Desde un punto de vista general, el análisis inteligente en escenas dinámicas incluye las siguientes **etapas funcionales** [HTWM04] (ver figura 2.10): modelado del entorno, detección de movimiento, clasificación de objetos móviles, *tracking*, comprensión y descripción de comportamientos y fusión de la información de múltiples sensores. Este conjunto de etapas están directamente relacionadas con la vigilancia visual o videovigilancia, debido a que es el tipo de análisis más común dentro de este área de investigación. Sin embargo, es posible extrapolarlas a sistemas más generales que incluyan cualquier otro tipo de sensores de vigilancia.

Dentro de las aplicaciones de videovigilancia, una de las vertientes más trabajadas es la que lleva a cabo el análisis de personas y/o vehículos, debido a que ambos tipos de actores son los más frecuentes a la hora de vigilar un entorno. Según Hu et al. [HTWM04], las aplicaciones de vigilancia en las que se realiza la monitorización de personas y/o vehículos incluyen las que se listan a continuación:

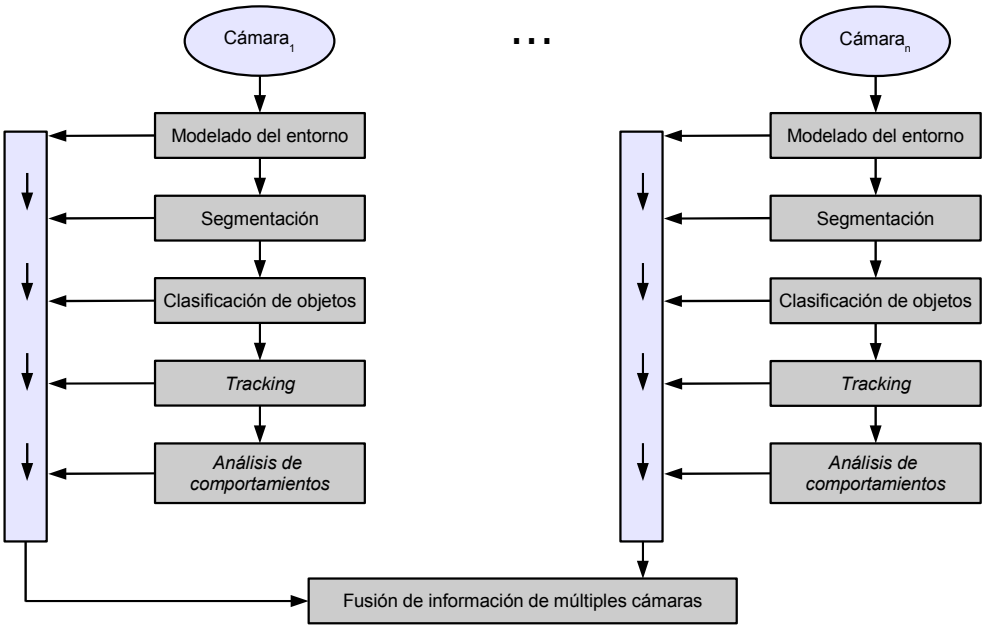


Figura 2.10: Esquema general de las etapas de un sistema de videovigilancia.

- **Control de acceso en áreas especiales**, lo que implica una identificación del actor.
- **Identificación de personas específicas en escenas concretas**, lo que permite el reconocimiento automático de ciertos individuos.
- **Análisis de aglomeraciones de personas**, lo que posibilita una gestión correcta del flujo de individuos en un determinado entorno (ver figura 2.8 (b)).
- **Detección de anomalías y alarmas**, gracias al análisis del comportamiento de objetos en un determinado entorno.
- **Vigilancia interactiva mediante múltiples cámaras**, para proporcionar una monitorización más avanzada a través de la fusión de información obtenida de múltiples sensores.

La importancia de las aplicaciones de vigilancia inteligente se ve reflejada en varios aspectos. Uno de ellos es el diseño y desarrollo de hardware adaptado para la vigilancia inteligente. En este contexto existe una gran cantidad de dispositivos concebidos para facilitar este proceso. Por ejemplo, grandes compañías como Sony e Intel han diseñado equipos adaptados para la vigilancia, como por ejemplo cámaras activas, cámaras *inteligentes* [KPP+97] o cámaras omnidireccionales [BME+98].

Otro aspecto clave de dicha relevancia es la organización de conferencias internacionales específicamente orientadas a temas relacionados con la vi-

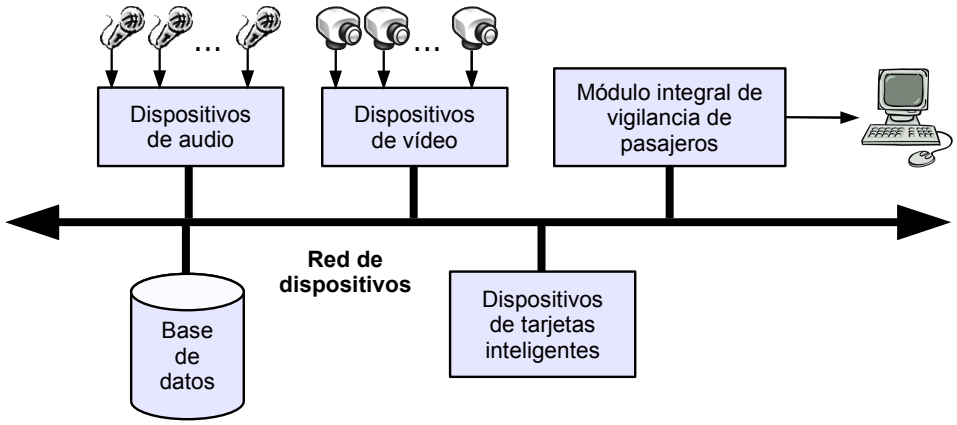


Figura 2.11: Vista abstracta de los componentes que forman el sistema PRISMATICA [VBL+05].

gilancia inteligente. Por ejemplo, dos de los eventos más importantes son la *IEEE International Conference on Advanced Video and Signal based Surveillance* (AVSS), organizada por IEEE, y la *ACM/IEEE International Conference on Distributed Smart Cameras* (ICDSC), organizada por IEEE y ACM.

En tercer lugar, es importante resaltar ciertos **proyectos de investigación** que se han desarrollado a lo largo del mundo dentro del ámbito de la vigilancia. Uno de los más relevantes fue VSAM [CLK+00] (*Visual Surveillance and Monitoring*), financiado en 1997 por el Ministerio de Defensa de los Estados Unidos, y cuyo propósito fue el desarrollo de tecnologías que facilitan la compresión automática de vídeo en escenas complejas.

A nivel europeo destacan principalmente los proyectos CROMATICA [3] (*Crowd Monitoring with Telematic and Communication Assistance*) y PRISMATICA [VBL+05]. (*Pro-active Integrated Systems for Security Management by Technological Institutional and Communication Assistance*). El objetivo del primero consistió en mejorar la vigilancia de los pasajeros en el transporte público, mientras que en el segundo se prestaba atención a los aspectos sociales, éticos, organizacionales y técnicos de la vigilancia en el transporte público.

Finalmente, resulta interesante reflexionar sobre la evolución de los sistemas de vigilancia inteligente y sobre cómo podemos contribuir al desarrollo de sistemas más robustos. Según [DV08], unas de las cuestiones pendientes es si debemos medir la bondad de una aplicación de vigilancia inteligente comparando su eficiencia con respecto a operadores humanos, como se plantea en [Mee04]. Sin embargo, se ha de tener en cuenta que la vigilancia inteligente abarca tareas a distintos niveles, desde la detección de eventos simple hasta el análisis de comportamientos complejos. Además, estas ta-

[3] http://dilnxsrv.king.ac.uk/cromatica/D2_WP3.pdf

reas suelen estar conectadas de manera que la entrada de datos de una suele ser la salida de datos de otra, por lo que el rendimiento final del sistema de vigilancia inteligente dependerá del rendimiento de cada una de las etapas que lo forman.

2.2.2. Evolución

En la tabla 2.3 se muestra un resumen de la evolución técnica de los sistemas de vigilancia. Algunos autores [VV05] clasifican este tipo de sistemas en distintas generaciones, en función de las características técnicas y de las ventajas que aportan con respecto a la generación anterior.

La evolución técnica de los sistemas de vigilancia comenzó con los sistemas analógicos CCTV (*Close-Circuit Television*), los cuales consisten en un determinado número de cámaras conectados a un conjunto de monitores normalmente emplazados en una única sala de control. Aunque esta tecnología es madura y ha sido ampliamente utilizada en todo el mundo, sufre de los problemas que se introdujeron en la sección 2.2.1 y que se derivan de la observación continua de monitores por parte del personal encargado de la

1ª generación	
Técnicas	Sistemas analógicos CCTV
Ventajas	Tecnología madura
Problemas	Uso de técnicas analógicas
Investigación	Técnicas digitales Compresión
2ª generación	
Técnicas	Vigilancia automatizada (CCTV+visión por computador)
Ventajas	Incremento de la eficiencia de CCTV
Problemas	Requiere análisis de comportamiento
Investigación	Algoritmos de visión robustos Aprendizaje automático de escenas *Gap* entre análisis estadístico e interpretación natural
3ª generación	
Técnicas	Sistemas de vigilancia automatizados
Ventajas	Información más eficiente Planteamiento distribuido
Problemas	Distribución de información Metodología de diseño Plataformas móviles
Investigación	Inteligencia distribuida versus centralizada Fusión de información Razonamientos probabilistas Vigilancia con múltiples cámaras

Tabla 2.3: Resumen de la evolución de los sistemas de vigilancia inteligentes.

seguridad. Básicamente, estos problemas se resumen en la imposibilidad de visualizar todos los monitores y el cansacio generado por dicha observación.

El siguiente paso en la evolución de los sistemas de vigilancia es la automatización de ciertas tareas que permitan obtener una mayor eficiencia a la hora de vigilar un entorno. En este contexto, la investigación está centrada en el desarrollo de algoritmos para la detección en tiempo real de eventos o de patrones de comportamiento.

Finalmente, los sistemas de vigilancia inteligente pretenden monitorizar entornos más amplios en los que la información obtenida está distribuida y, por lo tanto, es necesario el uso de técnicas de fusión de la información obtenida de múltiples sensores. Además, estos sensores pueden ser de distinta naturaleza, extendiendo el uso de las cámaras de vídeo tradicionales. La distribución de información, sin embargo, complica la vigilancia del entorno debido a que es necesario el uso de algún modelo que permita agregarla y componerla para proporcionar una vigilancia más completa y sofisticada.

2.2.3. Arquitecturas

Arquitecturas específicas para vigilancia

Normalmente, las arquitecturas de los sistemas de vigilancia suelen estar estructuradas en capas, de manera que cada capa contiene una serie de módulos encargados de proporcionar los servicios necesarios para llevar a cabo la monitorización de un entorno. Por ejemplo, Haritaoglu et al. [HHD00] diseñaron un sistema para integrar técnicas de vigilancia en un conjunto de PCs de bajo coste. Desde un punto de vista arquitectónico, existe una primera capa responsable de la detección de objetos y otra segunda de la clasificación de los mismos. El sistema está orientado a la detección de siluetas para monitorizar comportamientos en entornos exteriores. Sin embargo, la arquitectura del sistema no está concebida para expandir el sistema con el objetivo de analizar nuevas amenazas.

Actualmente, uno de los sistemas de vigilancia más sofisticados es PRISMATICA [VBL+05], desarrollado ante la necesidad de detectar situaciones de interés en entornos complejos. Este sistema se ha probado y evaluado en estaciones de tren como primer paso hacia sistemas de inteligencia ambiental más complejos. Desde el punto de vista arquitectónico, PRISMATICA está formado por un conjunto de dispositivos que contribuyen a la monitorización desde una perspectiva local. En otras palabras, cada dispositivo supervisa un área física reducida de manera que la información obtenida de manera local se envía a un controlador central cuando sea necesario. La comunicación entre los componentes de la arquitectura [VLS04] también juega un papel fundamental dentro del sistema de vigilancia distribuido. Así mismo, los autores concluyen el artículo reflexionando sobre la necesidad de incor-

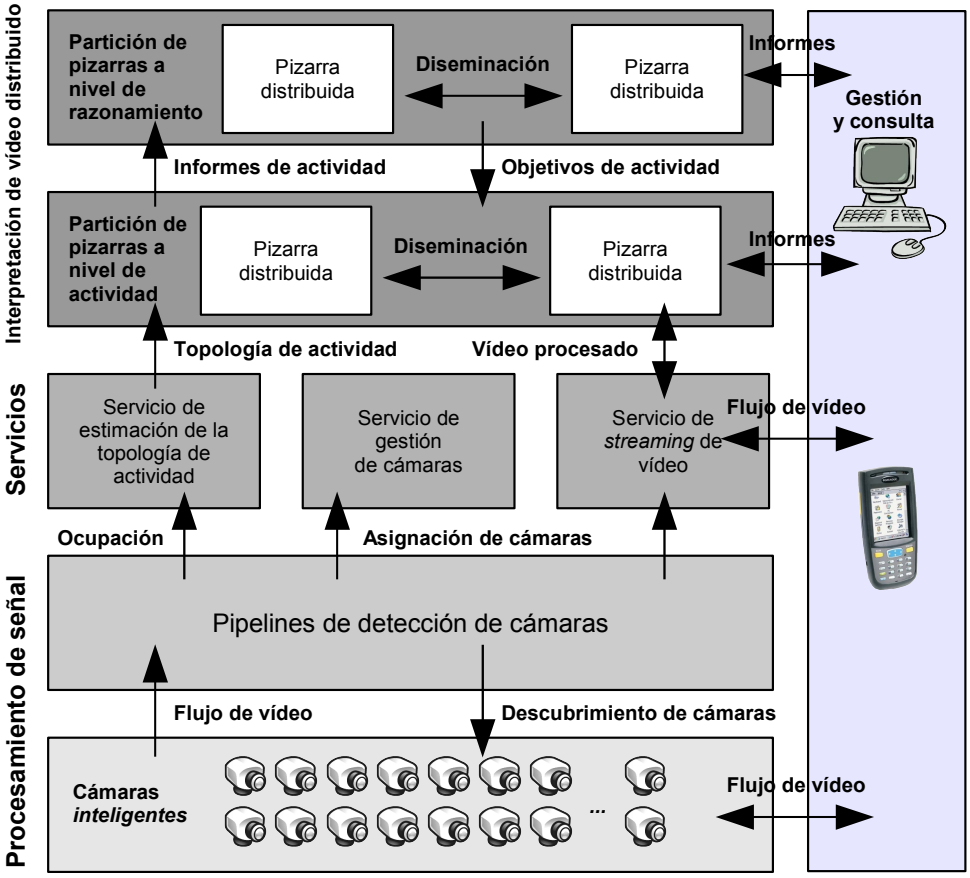


Figura 2.12: Arquitectura multi-capa de un sistema de videovigilancia distribuido. Esquema adaptado de [DvdHD+08].

porar conocimiento experto y aprendizaje automático para proporcionar una vigilancia más avanzada.

Dentro del ámbito de las arquitecturas para sistemas de vigilancia destaca otra rama orientada a garantizar la escalabilidad del sistema cuando es necesario incluir nuevos dispositivos de vigilancia o nuevos módulos de análisis del entorno. Esta serie de aproximaciones surge ante la necesidad de integrar la variedad de dispositivos que se vienen utilizando para monitorizar entornos, tanto en tipo (vídeo, audio, lectores de tarjetas, etc) como en número. Dentro de este ámbito, en [ERLA06] se presenta un trabajo interesante basado en la adopción de una arquitectura orientada a servicios [HS05] como base de un *framework* para el despliegue de aplicaciones de videovigilancia. Los autores hacen uso del sistema para desplegar algoritmos de detección y conteo de personas. Sin embargo, no existe una metodología clara de cómo escalar la arquitectura ante nuevas necesidades de vigilancia.

Un trabajo más reciente es el propuesto por Detmold et al. [DvdHD+08] (ver figura 2.12), en el que se plantea un *middleware* como mecanismo de despliegue de sistemas de videovigilancia inteligente basado en una topología que describe la actividad de los objetos monitorizados. Esta propuesta mantiene una arquitectura más clara y estructurada y plantea una arquitectura de pizarra distribuida con el objetivo de proporcionar un modelo de comunicación flexible y escalable. IBM también ofrece una alternativa en esta misma línea [TBH+08], basada en la gestión inteligente de los datos obtenidos de los dispositivos de vigilancia y en estándares abiertos.

No obstante, estos trabajos están muy orientados a la escalabilidad del sistema desde el punto de vista sensorial por una parte, es decir, teniendo en cuenta los dispositivos de vigilancia desplegados en el entorno, y por otra parte a la inclusión de módulos de análisis muy concretos para ciertos problemas o entornos. En otras palabras, no están diseñados teniendo en cuenta las ventajas que el uso de conocimiento experto puede aportar para proporcionar una vigilancia más avanzada a través de modelos y esquemas de conocimiento más generales. En el siguiente apartado se discute cómo el uso de la tecnología de agentes puede contribuir al desarrollo de este tipo de sistemas.

Sistemas multi-agente

Como se introdujo en la sección 2.1.2, un agente se puede entender como una entidad software que presenta como principales características la autonomía, habilidad social, reactividad y proactividad. Este conjunto de características básicas, que se puede complementar con aspectos como el aprendizaje automático, se adaptan muy bien a las necesidades de los sistemas de vigilancia inteligentes. Estas necesidades se refieren al tratamiento de la información obtenida del entorno, normalmente distribuida a lo largo del mismo, y al uso de técnicas de Inteligencia Artificial que permiten llevar a cabo una vigilancia más avanzada y sofisticada.

Así mismo, si tenemos en cuenta la naturaleza dinámica, heterogénea y compleja de los entornos de monitorización, en los que es necesario acometer la realización de distintas tareas y la composición de resultados (fusión de información), los Sistemas Multi-Agente se plantean como una solución natural en la que los agentes inteligentes se comunican y cooperan para realizar la vigilancia global de un entorno.

En esta sección se llevará a cabo un estudio de la contribución de la tecnología de agentes en la vigilancia inteligente, haciendo especial hincapié en los Sistemas Multi-Agente y en las arquitecturas propuestas para tratar con este problema.

Segmentación

El proceso de segmentación permite distinguir los objetos de interés de un entorno monitorizado visualmente. Al igual que ocurre con el *tracking*, la segmentación es una tarea clave para que las etapas de más alto nivel, como por ejemplo el análisis de comportamientos, funcionen correctamente.

En la literatura existen diversas propuestas que hacen uso de la tecnología de agentes para segmentar imágenes. Una de las primeras propuestas se discute en [LT99], en la que los autores plantean un enfoque en el que una imagen se interpreta como un entorno bidimensional donde habitan los agentes encargados de etiquetar segmentos homogéneos. Para ello, los agentes están basados en comportamientos reactivos con la capacidad de instanciar nuevos agentes en regiones vecinas si la detección de ciertos píxeles fue exitosa.

R. Lee también propuso un sistema de vigilancia basado en agentes que aplicó para la segmentación de imágenes y para la detección facial [Lee03]. La arquitectura propuesta en este artículo es de propósito general y se divide en varias capas: capa de aplicación, capa *consciente* o *inteligente*, capa tecnológica y capa de infraestructura. En este contexto, los agentes propuestos para segmentar e identificar caras habitan en la capa de más alto nivel, es decir, en la de aplicación. Sin embargo, no existe una cooperación clara entre ambos agentes, sino que se conciben desde un punto de vista más individualista, en el que cada uno realiza su tarea sin tener en cuenta el estado del otro.

E. Bovenkamp et al. [BDBR04] dieron un paso más en el uso de la tecnología de agentes para la segmentación proponiendo un sistema multi-agente en el que los agentes se adaptan dinámicamente a los algoritmos de segmentación. Dicha adaptación se realiza en base al conocimiento de restricciones globales, conocimiento contextual, información local de las imágenes y creencias personales. De este modo, cada agente del sistema es responsable de identificar un objeto de alto nivel, cooperando con otros agentes para mantener una interpretación completa y consistente de la imagen.

En [RBG+04] se plantea un sistema de procesamiento de imágenes paralelo basado en agentes reactivos y que hace uso de un lenguaje que permite la descripción del comportamiento de los agentes para el tratamiento de imágenes.

Tracking

La tecnología de agentes también se ha utilizado en la etapa de *tracking*, con el objetivo de proporcionar un sistema de seguimiento más completo y eficaz que, normalmente, hace uso de varias cámaras de manera simultánea.

Una de las aproximaciones más utilizadas en la literatura para abordar el *tracking* mediante la tecnología de agentes consiste en instanciar un agente

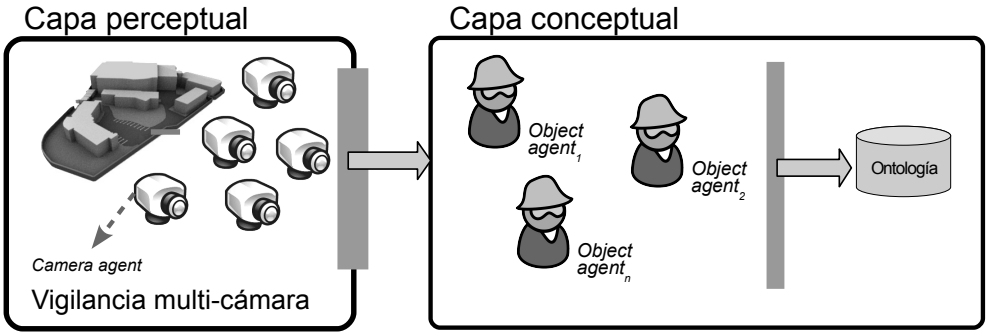


Figura 2.13: Arquitectura multi-agente abstracta para el *tracking* multi-cámara. Esquema adaptado de [RSJ04].

por cámara, con el objetivo de que cada uno de ellos realice su propio *tracking* local para, posteriormente, comunicarse con el resto de agentes para manejar unas referencias globales de los objetos monitorizados. Un ejemplo lo podemos encontrar en el trabajo de Remagnino et al. [RSJ04], en el que se hace uso del *camera agent* para efectuar las tareas de calibración de la cámara, *tracking* y aprendizaje. Así mismo, los autores apuntan a trasladar el entorno de ejecución de este tipo de agentes hacia la propia cámara, en lugar de instanciarlos en el computador que utiliza la cámara como recurso.

En [GCM05] se plantea, desde un punto de vista general, la vigilancia de varios objetivos móviles mediante el uso de cámaras en un entorno cerrado. Al igual que ocurría con el trabajo anterior, cada agente controla a una cámara de manera que todos los agentes se coordinan para proporcionar una vigilancia global del entorno. El modelo interno de los agentes consiste en emular a los operadores reales, intercambiando mensajes bien definidos para interpretar las distintas situaciones del entorno.

En esta misma línea, Ukita y Matsuyama [UM05] presentaron un sistema de *tracking* multi-objetivo y multi-cámara en tiempo real formado por los denominados *Agentes de Visión Activos* (*Active Vision Agents*) o AVAs. En este trabajo, la principal novedad es el enfoque de la arquitectura para el análisis en tiempo real de la escena. Dicha arquitectura está compuesta de tres capas en las que los procesos se ejecutan de manera concurrente y se lleva a cabo una gestión dinámica de la memoria del sistema. Cada AVA puede jugar uno o varios roles, como por ejemplo los de búsqueda o *tracking*.

M.A. Patricio et al. [PCP+07] presentan un trabajo más reciente y detallado en lo relativo a la tercera generación de sistemas de vigilancia. En este caso, los autores hacen uso del modelo BDI para implementar la deliberación acerca de las imágenes de una cámara, utilizando JADEX para implementar los agentes software.

Como en algunos de los trabajos citados anteriormente, los autores plantean agentes del tipo *camera agent*, los cuales tomarán decisiones de acuerdo

a un modelo simbólico interno que representa las situaciones encontradas y los estados mentales en la forma de deseos, creencias e intenciones. Las creencias representan información sobre el mundo exterior, los deseos están vinculados a la vigilancia permanente y al seguimiento temporal y, por último, las intenciones están relacionadas con las acciones externas (comunicación con otros agentes) e internas (órdenes al sistema de seguimiento o a la propia cámara). Los principales objetivos del sistema se pueden resumir en el diálogo para compartir información de un objeto, el razonamiento sobre esa información y la gestión de dependencias entre cámaras vecinas. En otras palabras, el eje central del trabajo reside en la coordinación entre agentes y en la comunicación asociada a los mismos.

Una de las líneas de evolución del *tracking* basado en agentes es la inclusión de estos en las propias cámaras, es decir, que tanto el software (agentes) como el hardware (cámara) sean un todo, dibujando un esquema homogéneo de elementos de cómputo. En este contexto, un trabajo interesante es el desarrollado por M. Bramberger et al. [BDM+06], en el que se usa la combinación de entidades de este tipo para conformar un sistema de vigilancia que soporta la comunicación y la cooperación entre cámaras mediante los denominados *agentes móviles*.

Análisis de comportamientos e identificación de eventos

Una de las principales funciones de alto nivel más extendidas en los sistemas de vigilancia inteligente es el análisis de comportamientos de objetos móviles. Esta línea de investigación está vinculada a detectar de manera automática situaciones anómalas dentro de un entorno, como por ejemplo la identificación de objetos abandonados o la realización de circulaciones sospechosas.

Algunos de los trabajos basados en agentes previamente comentados, además de llevar a cabo procesos de más bajo nivel como la segmentación y el *tracking*, complementan el sistema de vigilancia con actividades de más alto nivel, como las mencionadas recientemente. Por ejemplo, en [RSJ04] se plantea un agente específico denominado *object agent* cuyo objetivo es el identificar el comportamiento de un objeto a partir del conocimiento asociado a una serie de actividades concretas. Este agente interactúa con el *camera agent*, el cual le proporciona la información de posicionamiento del objeto monitorizado, y con otros *object agents* para detectar si dos o más agentes están monitorizando a un mismo objeto móvil.

Dentro del análisis de comportamientos, una de las principales áreas de aplicación es la monitorización de vehículos. En este contexto, el principal objetivo es el de detectar qué vehículos no se comportan normalmente de acuerdo a las normas establecidas en un determinado entorno. No obstante, algunos de estos sistemas tratan de gestionar el flujo de tráfico mediante los propios semáforos [Moh06] o incluso mostrar información dinámica a los conductores en función de las condiciones meteorológicas [TG05].

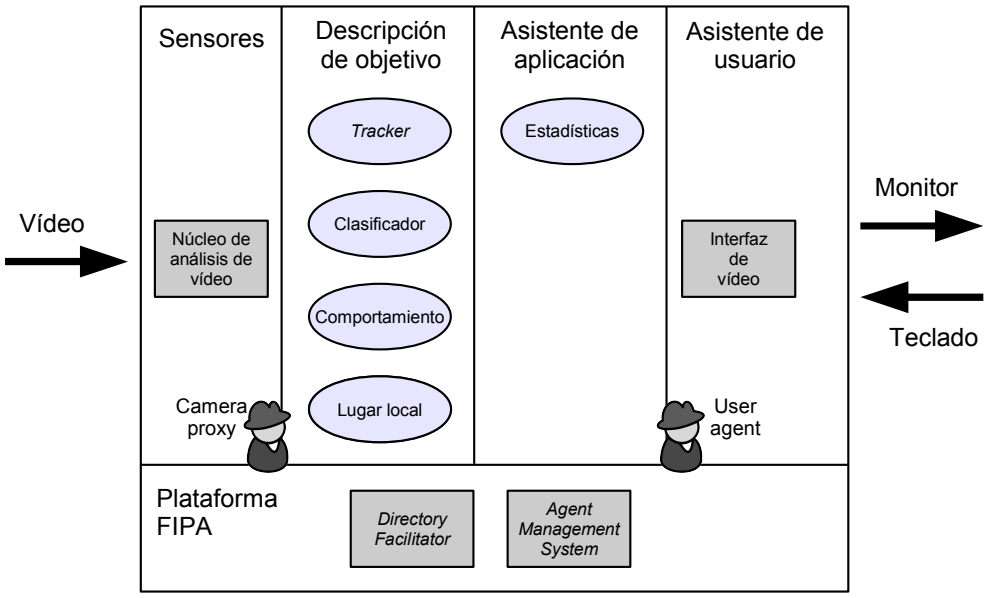


Figura 2.14: Arquitectura multi-agente del sistema *Monitorix*.
Esquema adaptado de [ABC+00].

Otro trabajo importante es el desarrollado por B. Abreu et al. [ABC+00], en el que los autores proponen una arquitectura bien estructurada en capas en las que se sitúan distintos tipos de agentes (ver figura 2.14). La capa de más bajo nivel se corresponde con la capa de sensores y actuadores, en la que existe un agente por cámara (*camera proxy*). A continuación, y sobre la capa anterior, se establece la capa de descripción de objetos en la que se maneja una semántica de alto nivel. En esta capa se sitúan distintos tipos de agentes que proporcionan servicios de repositorio, de *tracking* o de clasificación. En un nivel superior se emplaza la capa de asistencia a aplicaciones, utilizada para la toma de decisiones y compuesta de agentes del tipo *estadísticas*. Por último, en la capa de asistencia al usuario se establecen los perfiles que condicionan las interacciones en capas de nivel inferior a través de agentes del tipo *user agent*.

Algunos trabajos más recientes tienen en cuenta cuestiones como la interoperabilidad con otros sistemas multi-agente y la flexibilidad de la arquitectura, como el trabajo propuesto en [CCP09], en el que se plantea el diseño de un agente FIPA-compliant [Fou04] para la gestión y la detección de tráfico en tiempo real.

Fusión de información

Dentro del ámbito de la vigilancia, la fusión de información es el proceso mediante el cual se agregan datos o, desde un punto de vista más abstracto, conocimiento para proporcionar una vigilancia más completa y robusta. La agregación o fusión de datos consiste en combinar la salida de múltiples

sensores para obtener una información de mayor calidad, es decir, una información que recoja de manera global los datos obtenidos por un conjunto de sensores [Mit07].

En la figura 2.15 se plantea la arquitectura de un sistema multi-agente en el que el proceso de fusión se realiza directamente a partir de los datos recogidos por los distintos sensores que forman el sistema de vigilancia [LLACSJ09]. En este caso, existe una capa semántica encargada de fusionar los datos de dichos sensores para posteriormente enviar esta información en forma de eventos a una capa de más alto nivel responsable de analizar distintas amenazas en el entorno monitorizado.

Karlsson et al. plantearon un sistema multi-agente para monitorizar objetivos de interés mediante agentes software [KBKA05], los cuales son responsables de la fusión de información y de usar el sensor más cercano al objeto monitorizado, a través de una política de migración. Al igual que ocurría con el trabajo mencionado anteriormente, esta propuesta se enmarca dentro de la fusión de información obtenida directamente desde los sensores. En este contexto, la fusión de información también se puede utilizar para evitar inconsistencias y obtener unos resultados más eficaces en sistemas de videovigilancia con vistas solapadas [CGPM08, RSJ04].

Algunos trabajos en los que se plantea una red de sensores plantean arquitecturas que faciliten tanto la obtención y la fusión de información a los agentes software encargados de obtener información de más alto nivel. En [BQX08] se plantea una arquitectura basada en agentes móviles que van migrando de sensor a sensor en función de un itinerario preestablecido o calculado en tiempo de ejecución. A diferencia del trabajo de [KBKA05], esta propuesta está más orientada a la eficiencia relativa al consumo de los dispositivos que forman parte del sistema de vigilancia.

Idealmente, el modelo teórico utilizado para el proceso de fusión de información debería ser lo suficientemente flexible para agregar tanto los datos obtenidos de los sensores de vigilancia como la información de más alto nivel generada por los propios agentes de vigilancia, e incluso debería posibilitar la fusión de estos dos tipos de información. De este modo, la capa de fusión de información estaría situada por encima de la capa cognitiva de vigilancia, permitiendo una monitorización desde un punto de vista más global que solventara la posible aparición de inconsistencias entre procesos de vigilancia locales, e incluso proporcionando una vigilancia más sofisticada.

Arquitecturas de propósito general

Además de la existencia de propuestas de arquitecturas específicas para la vigilancia inteligente, existe la posibilidad de utilizar una arquitectura de propósito general. De este modo, habrá elementos de la arquitectura que se puedan usar directamente y otros que tengan que adaptarse al problema de la vigilancia o incluso ser diseñados desde cero.

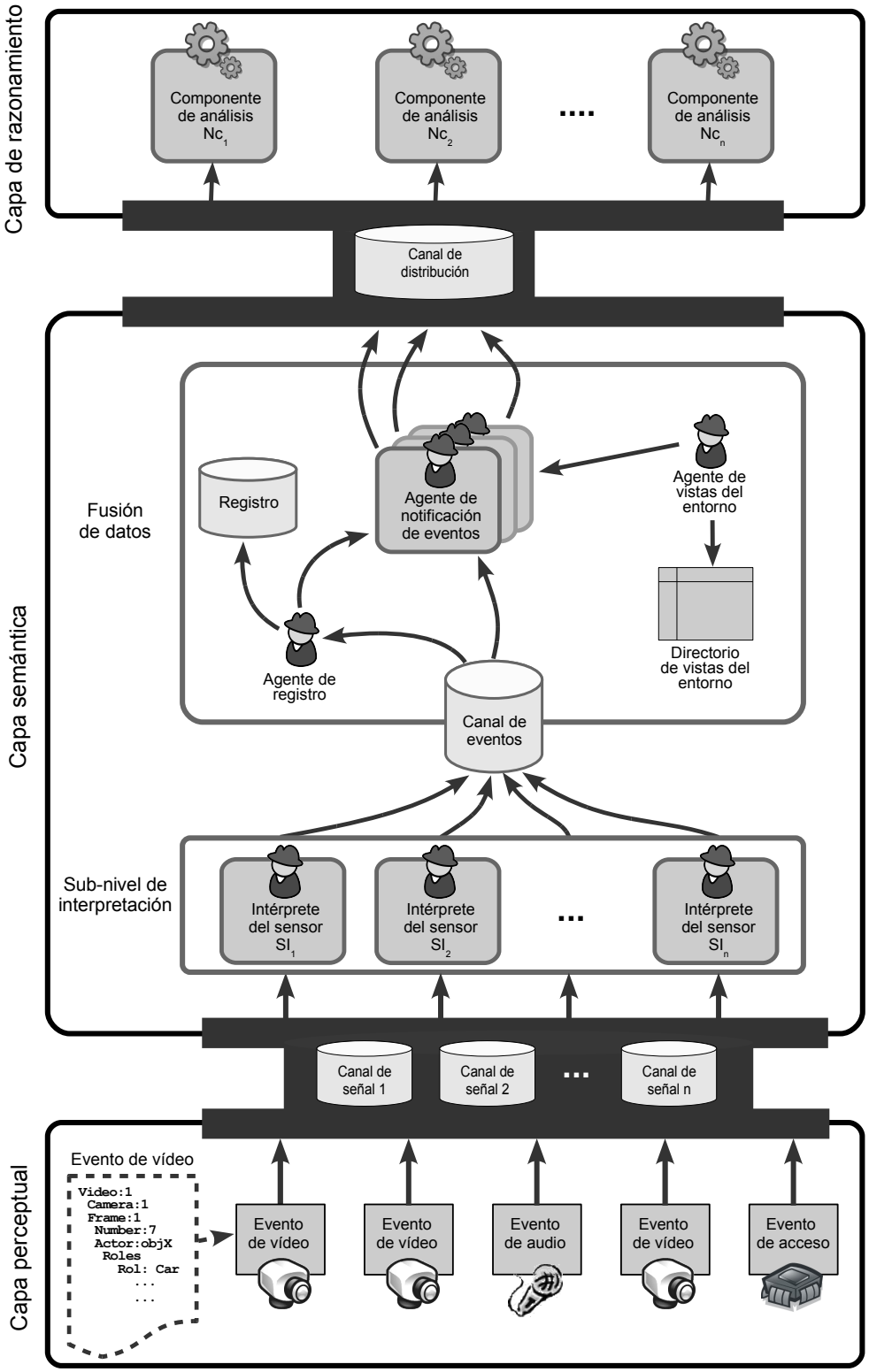


Figura 2.15: Arquitectura multi-agente para la fusión multi-sensor. Esquema adaptado de [LLACSJ09].

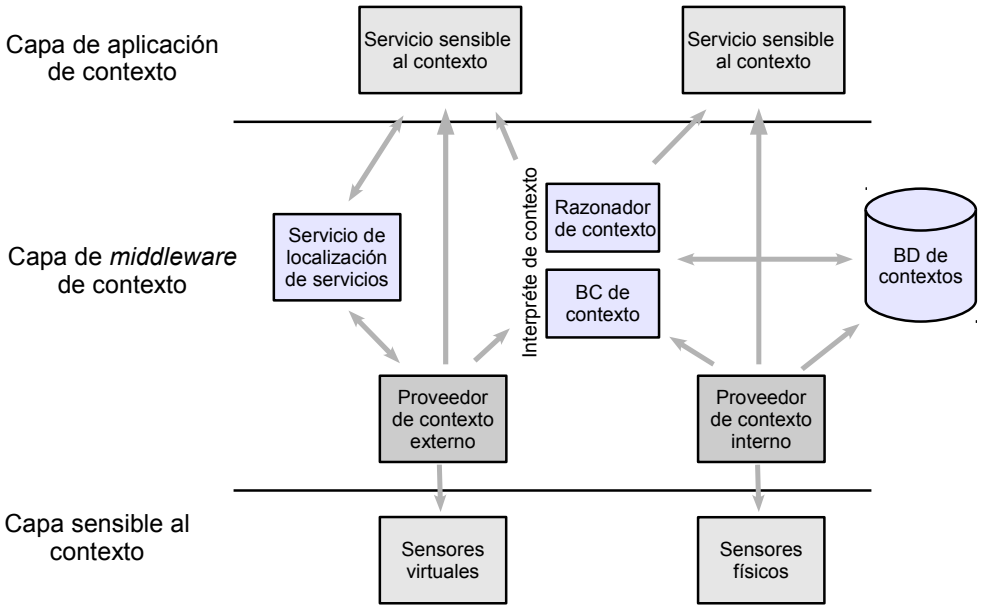


Figura 2.16: Ejemplo de arquitectura orientada a servicios y sensible al contexto. Esquema adaptado de [Set05].

Uno de estos enfoques gira en torno al uso de un entorno que facilite el despliegue de **sistemas orientados a servicios**. En otras palabras, la idea principal consiste en proporcionar una herramienta para crear sistemas orientados a servicios [HS05] desde una perspectiva general. Dentro de este planteamiento, T. Gu et al. [Set05] proponen la arquitectura SOCAM (*Service-Oriented Context-Aware Middleware*) para construir una arquitectura basada en servicios sensibles al contexto (ver figura 2.16). Los autores afirman que una infraestructura de este tipo requiere los siguientes elementos:

1. Un modelo de contexto común para que pueda compartirse entre todos los dispositivos y servicios. En este ámbito, compartir el contexto es importante para que el resto de entidades puedan entenderlo dentro de un dominio o de varios. Para ello, los autores hacen uso de un modelo de contexto basado en ontologías y lo representan con OWL.

2. Un conjunto de servicios para llevar a cabo la adquisición de contextos, el descubrimiento, la interpretación y la difusión. Para ello, es necesario diseñar una serie de servicios que den soporte a estas funciones, lo cual implica clasificar los servicios en dependientes del dominio e independientes del dominio.

En este trabajo, la noción de contexto se utiliza principalmente para dar soporte a la adquisición, el descubrimiento, la interpretación y el acceso a los servicios de la plataforma. En lo relativo al nivel semántico, proponen

un modelo formal de contexto basado en ontologías y utilizan OWL para llevar a cabo la representación semántica, el razonamiento y la clasificación de contextos y dependencias. A nivel de evaluación, dichos autores hacen uso exclusivo de la tecnología Java para el desarrollo de un prototipo de hogar inteligente. En realidad, esta propuesta encaja con el diseño de la arquitectura de un sistema de vigilancia cognitivo, ya que permite el desarrollo de servicios sensibles al contexto y propone una serie de servicios independiente del dominio de conocimiento para dar soporte a todo lo relacionado con la gestión de servicios.

En cuanto a la arquitectura de SOCAM (ver figura 2.16), no queda claro cómo opera el servicio de localización de servicios, es decir, cómo se consulta y qué información obtiene. Por otra parte, tampoco se mencionan ciertos servicios que son muy interesantes en estos sistemas generales, y en los de vigilancia en particular, como los de despliegue y gestión o los servicios de persistencia. Por lo tanto, la principal conclusión que se obtiene a partir del estudio de este trabajo es que puede resultar adecuado para instanciar un sistema de vigilancia pero que necesita mejoras importantes, como los servicios básicos comentados y la posibilidad de mejorar la eficiencia a través del uso de otro tipo de tecnologías.

Dentro de esta categoría de sistemas de propósito general, el uso de agentes inteligentes supone un paso más en la creación de sistemas orientados a servicios. En este contexto, L. Chunlin et al. [CL02] han desarrollado JASE (*Java-based Agent-oriented and Service-oriented Environment*) con el objetivo de crear un entorno para el despliegue de sistemas distribuidos dinámicos. Los agentes móviles se utilizan como soporte de las aplicaciones, mientras que los agentes de interfaz de servicio se emplean como envolturas de esos servicios. La idea principal reside en la concepción de una aplicación independiente de recursos, los cuales se modelan a través de servicios proporcionados por las aplicaciones que están basadas en agentes.

Aunque la arquitectura incluye un mecanismo de localización de servicios, el número de pasos necesarios para contactar con un servicio determinado es muy elevado y requiere el análisis y la comparación de documentos XML, los cuales describen a cada uno de los servicios. Este factor influiría negativamente en un sistema de vigilancia a la hora de localizar servicios. Por otra parte, el trabajo menciona la posibilidad de manejar agentes móviles, pero no comenta ningún tipo de mecanismo que facilite la gestión y el despliegue del sistema distribuido.

El otro gran enfoque dentro de esta sección de sistemas de propósito general tiene como núcleo un concepto que actualmente está siendo empleado para referirse a la siguiente generación de *grids* [CT03] (también referenciado como ***grids* cognitivos** o semánticos). Dicha generación tiene como base un sistema de alto nivel para crear *grids* [FKNT02] basados en servicios de descubrimiento de conocimiento. La arquitectura de este sistema contiene dos capas: i) servicios y middleware básicos y genéricos y ii) servicios para el diseño y la ejecución de aplicaciones de descubrimiento de servicios.

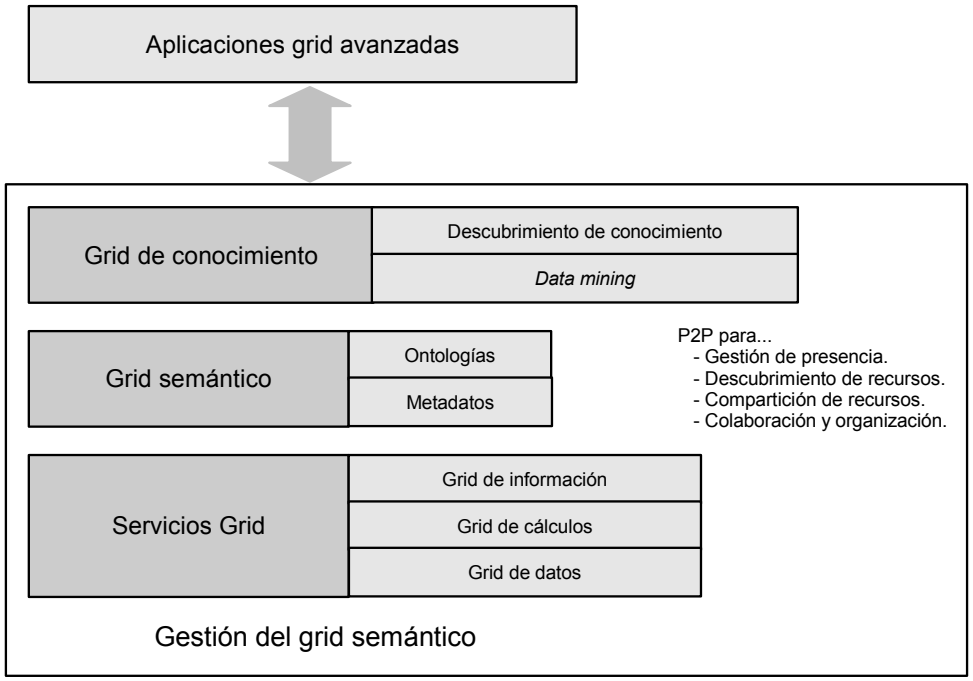


Figura 2.17: Capas de la siguiente generación de grids.
Esquema adaptado de [CT04].

Como ocurre con el anterior enfoque, existen diversos trabajos que proporcionan un entorno sobre el cual enmarcar dichos servicios. M. Cannataro y D. Talia [CT04] proponen el uso de una arquitectura P2P para asegurar la escalabilidad del *grid* y descentralizar sus funcionalidades, con el objetivo principal de evitar posibles cuellos de botella (ver figura 2.17). No obstante, habría que evaluar más en detalle el uso de un sistema P2P como base para un sistema de vigilancia, ya que en principio, los autores orientan la arquitectura a tareas más pesadas computacionalmente, como por ejemplo el análisis de grandes cantidades de datos. Así mismo, habría que incluir otros servicios en el grid de cara a desplegarlo en un entorno de vigilancia.

Un punto de vista distinto está basado en la combinación de los *grids* y los agentes, como plantea Y. Gil [Gil06]. En este trabajo, la autora expone los beneficios obtenidos al combinar dichos elementos, justificando por qué se puede utilizar la tecnología *grid* para complementar a la tecnología de agentes y viceversa. No obstante, el concepto fundamental sigue girando en torno a la nueva generación de *grids*, es decir, a los *grids* cognitivos. Los principales argumentos de este trabajo residen en que la tecnología *grid* aporta robustez e incrementa el rendimiento de los agentes, además de manejar conceptos como el de estado, persistencia y gestión de los ciclos de vida. Por otra parte, se justifica que la teoría de agentes proporciona aprendizaje, planificación, interacción y coordinación.

Otra posible solución de propósito general es el uso de un sistema distribuido basado en el conocimiento para desplegar instancias concretas del mismo en problemas concretos, como por ejemplo la vigilancia inteligente. Éste es el enfoque que se sigue en [DGRMPP05]. En este artículo se presenta una arquitectura abierta y flexible basada en la tecnología de agentes y en el uso de ontologías para la representación del conocimiento. Para evaluar el sistema, los autores despliegan un sistema para la consulta de desórdenes psicológicos.

2.3. Arquitecturas Orientadas a Servicios

2.3.1. Conceptos básicos

La **arquitectura orientada a servicios (SOA)** se puede entender como un patrón de diseño arquitectural relativo a la definición de relaciones débilmente acopladas entre productores y consumidores. Desde un punto de vista abstracto, esta arquitectura no está directamente vinculada con el software, la programación ni la tecnología. SOA se entiende como una arquitectura que hace hincapié en la orientación a servicios como principio fundamental de diseño [SH05]. La idea principal es que en una arquitectura de este tipo puedan coexistir distintos servicios independientes y accesibles sin necesidad de conocer la tecnología subyacente utilizada para su implementación [HS05].

La **computación orientada a servicios (SOC)** es el paradigma de computación que utiliza los servicios como elementos centrales en el desarrollo de aplicaciones [PG03]. Los servicios básicos, sus descripciones y las operaciones básicas (publicación, descubrimiento, selección, etc.) que producen o usan dichas descripciones constituyen los pilares de una arquitectura orientada a servicios.

En este contexto, los **servicios** son componentes autosuficientes y abiertos que definen a la aplicación distribuida. Las descripciones de los servicios sirven para definir sus capacidades, sus interfaces, sus comportamientos y sus calidades de servicio. La publicación de esta información sobre los servicios disponibles proporciona el mecanismo necesario para descubrir, seleccionar o componer servicios. Así mismo, la descripción de la interfaz de un servicio especifica la firma del servicio, es decir, sus parámetros de entrada y salida, las posibles excepciones y los tipos de los mensajes. El comportamiento esperado de un servicio durante su ejecución puede especificarse a través de su descripción de comportamiento (por ejemplo usando un diagrama de flujo). También es posible asociar descripciones y atributos de calidad de servicio, como por ejemplo el coste o la disponibilidad. La interfaz general de un servicio se puede ver en el siguiente listado.

Descripción de un servicio

```
1  interface Blackboard
2  {
3
4    idempotent T read (TID id);
5    idempotent TSeq readAll ();
6    bool write (T data);
7
8    void subscribe (Object* prx) throws AlreadySubscribed;
9    idempotent void unsubscribe (Object* prx);
10
11 };
```

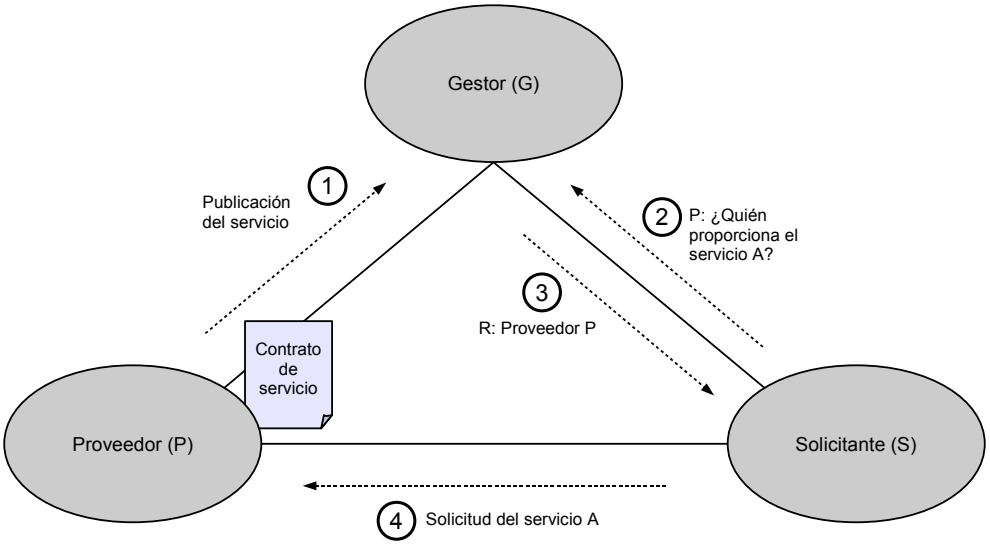


Figura 2.18: Arquitectura orientada a servicios.

Aunque puede dar la sensación de que la orientación a servicios no supone ninguna novedad y podría asociarse directamente a ciertos principios básicos de la Ingeniería del Software, sí que se desmarca en cierto sentido a la hora de plantear las aplicaciones de negocio como un conjunto de servicios distribuidos que se pueden utilizar cuando sea necesario. Para ello, quizás la principal práctica del desarrollo orientado a servicios consista en establecer y adherirse a los contratos de los servicios [Mey92], separando la definición de la interfaz de su implementación. Por otra parte, una arquitectura orientada a servicios no sustituye a los principios de la orientación a objetos. De hecho, se puede implementar una arquitectura orientada a servicios haciendo uso de las ventajas que proporciona la orientación a objetos.

Para garantizar la eficiencia en una arquitectura orientada a servicios se deben cumplir los siguientes requisitos:

1. **Interoperabilidad** entre los distintos sistemas, lenguajes de programación y plataformas empleados.
2. Uso de un **lenguaje de descripción** claro, no ambiguo e independiente de la plataforma de ejecución.
3. Adopción de un mecanismo de **recuperación** de servicios.

A nivel de componente, cada elemento de la arquitectura puede asumir uno o más de los tres siguientes roles (ver figura 2.18) [HS05]:

- **Proveedor** de servicios; la idea reside en la publicación de su interfaz para que otros servicios puedan acceder a ella.

- **Gestor** de servicios; este rol alude a la responsabilidad de manejar un registro de servicios de forma que éstos sean accesibles por parte de los posibles solicitantes.
- **Solicitante** de servicios; un elemento de la arquitectura que tenga este rol buscaría entradas en el registro del gestor de servicios empleando distintos operadores y, posteriormente, solicitaría un servicio a un hipotético proveedor.

Esta arquitectura orientada a servicios se adapta bien a un sistema de vigilancia inteligente. Por un lado, es necesario garantizar la interoperabilidad entre los distintos componentes que forman el sistema global, ya que dicho sistema estará compuesto por una gran variedad de elementos que se ejecutarán en distintas plataformas, bajo diferentes sistemas operativos y será necesario acceder a ellos a través de APIs que estarán codificados utilizando distintos lenguajes de programación.

Por otra parte, en un sistema de vigilancia resulta muy importante el uso de un lenguaje de descripción de interfaces que sea capaz de reflejar los distintos servicios ofrecidos por cada uno de los componentes. De este modo, se establece un contrato entre los distintos elementos de la plataforma, permitiendo conocer de manera clara y no ambigua los servicios que proporcionan cada uno de los mismos. Por último, en un sistema de vigilancia inteligente es necesario un elemento que conozca los distintos servicios accesibles no sólo de manera interna al propio sistema, sino también de manera externa por otros usuarios o sistemas.

2.3.2. Arquitectura general

Existen distintos principios arquitectónicos que debería seguir una arquitectura orientada a servicios [HS05]:

- **Mínimo acoplamiento**. Los servicios mantienen una relación que minimiza las dependencias y que sólo requiere tener noción de la existencia del resto de servicios (mecanismo de localización). En otras palabras, se debe considerar que las relaciones contractuales de alto nivel especifican la interacción entre servicios y proporcionan la consistencia a nivel de sistema.
- **Neutralidad en la implementación**. De manera general, una aplicación desarrollada bajo un enfoque orientado a servicios no debería ser específica de un conjunto de lenguajes de programación, es decir, se ha de separar la interfaz de un servicio y su implementación. La interfaz de un servicio define el contrato de dicho servicio.
- **Configuración flexible**. La configuración de un sistema orientado a servicios es dinámica, es decir, puede cambiar en tiempo de ejecución y

no debería alterar la consistencia del sistema global. Por ejemplo, podría ser deseable activar servicios bajo demanda sin tener que instanciar todos los posibles servicios de un sistema cuando éste se ponga en funcionamiento.

- **Persistencia**. Los servicios no necesitan necesariamente un período de vida muy largo, pero como se está tratando con cálculos entre elementos heterogéneos y autonómos en entornos dinámicos se deberían gestionar las posibles excepciones. Los servicios deben tener la capacidad de detectar excepciones, tomar las acciones correctivas pertinentes y responder de manera adecuada ante situaciones imprevisibles.
- **Granularidad**. Los participantes de una arquitectura orientada a servicios han de modelarse mediante una granularidad gruesa. En lugar de modelar interacciones con un cierto nivel de detalle se debe reflejar la funcionalidad de más alto nivel que define a un servicio dentro de un sistema. Las interfaces de grano grueso reducen las dependencias y las comunicaciones a un número menor de mensajes, los cuales poseen mayor importancia.
- **Comunidades**. En lugar de entornos de cálculo centralizado se debería pensar en términos de cooperación entre elementos. Este principio está directamente relacionado con los sistemas multi-agente.

Servicios relevantes

La **localización** es uno de los servicios críticos dentro de un sistema distribuido. Debido a que un sistema desarrollado bajo una arquitectura orientada a servicios no deja de ser una sistema distribuido, el servicio de localización juega un rol fundamental a la hora de solicitar servicios dentro de una plataforma. A la hora de gestionar un servicio de localización surgen dos elementos principales (ver figura 2.18):

- **Cliente** o **solicitante de servicios**. Este elemento representa a la entidad que desea solicitar un servicio en un momento determinado.
- **Servidor** o **proveedor de servicios**. Este elemento, por el contrario, está vinculado a aquella entidad que es capaz de proporcionar un determinado servicio bajo ciertas condiciones o parámetros.

En una arquitectura orientada a servicios, el tercer elemento relevante y asociado a la localización de servicios es el denominado **gestor de servicios**. Sin embargo, esta entidad no es imprescindible a la hora de diseñar un mecanismo de localización de servicios. Una primera aproximación podría consistir en utilizar algún tipo de mecanismo de difusión o *broadcast* dentro de la propia plataforma (ver figura 2.19 izquierda). De este modo, la petición de un determinado cliente llegaría a todos los potenciales servidores

o proveedores de servicios. De todos ellos, sólo responderían a dicha petición aquellas entidades que implementaran el servicio solicitado. Siendo aún más restrictivos, se podría utilizar algún lenguaje que restringera aún más esas respuestas. Evidentemente, el mecanismo planteado no es escalable. Si el número de dispositivos que componen la plataforma es elevado, entonces el rendimiento del sistema se ve afectado negativamente debido al número de peticiones enviadas.

Una alternativa más viable consiste en centralizar la información relativa a los proveedores de servicios de una plataforma a través de un tercer elemento que lleve a cabo la gestión de dicha información. De este modo, la difusión de mensajes por parte de un solicitante de servicios desaparece y ahora el mecanismo de localización de servicios consiste en preguntar directamente a dicho gestor. No obstante, los proveedores de servicios deben informar al gestor sobre los servicios que proporcionan de manera previa (ver figura 2.19 derecha).

Tradicionalmente, este servicio proporcionado por el gestor se ha denominado **servicio de páginas amarillas**. Mediante este modelo, el gestor maneja un directorio en el que almacena la información de los servicios que se proporcionan en la plataforma. Cuando el solicitante de un determinado servicio pregunta al gestor, éste le indica dónde puede contactar con las entidades que proporcionan dicho servicio, es decir, sus direcciones físicas. De manera análoga a un servicio de páginas amarillas, también puede existir un **servicio de páginas blancas**. Este servicio permite contactar con la entidad que tenga un determinado identificador, es decir, permite obtener su dirección física a partir de su nombre. Evidentemente, el servicio de páginas amarillas tiene mucha más importancia que el servicio de páginas blancas en una plataforma orientada a servicios.

Aunque la escalabilidad de la propuesta de un servicio de directorio mejora con respecto a un mecanismo de difusión, surge un nuevo problema de manera directa: la fiabilidad. Debido a que el gestor reside en un determinado nodo de la red, éste supone un único punto de acceso. Si el gestor finaliza su ejecución de manera inesperada, el servicio de páginas amarillas desaparece automáticamente. La solución más obvia consiste en replicar el gestor, es decir, instanciarlo n veces dentro de la plataforma. De este modo, si una instancia del gestor cae, el resto de las instancias pueden asumir el control del servicio de directorio.

Sin embargo, este planteamiento genera una nueva cuestión a tratar y que está relacionada con la consistencia de la información almacenada en el directorio. Las distintas instancias del gestor deben implementar un mecanismo para asegurar que la información del directorio de servicios ahora distribuido sea correcta en un determinado instante de tiempo. Además, este mecanismo también hace que la eficiencia del sistema sea menor debido a que es necesario un mayor número de mensajes para garantizar que la información del directorio esté actualizada.

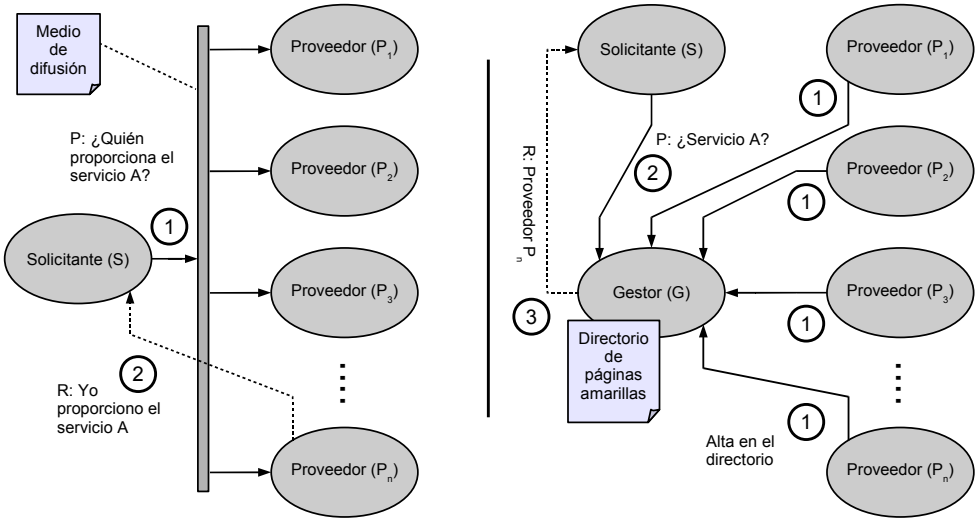


Figura 2.19: Izquierda: localización por difusión. Derecha: localización a través de un servicio de directorio.

En la figura 2.20 se muestra un esquema de replicación basado en un maestro y un número indeterminado de esclavos, de manera que cada esclavo hace uso del patrón de diseño observador [GHJV95] para comunicarse con el maestro. Básicamente, cada uno de los esclavos observa al maestro: si el maestro actualiza su estado, éste envía una notificación a los esclavos. Del mismo modo, cada esclavo mantiene un estado interno que sólo se actualiza cuando así lo notifica el maestro. Si éste no es capaz de enviar notificaciones de cambio de estado, la sesión previamente establecida se cerrará y el esclavo tendrá que establecer una nueva sesión con el maestro y sincronizar su estado. Si por algún motivo el maestro deja de realizar correctamente su funcionamiento, como por ejemplo debido a una caída de la máquina en la que se ejecuta, entonces se ha de utilizar algún tipo de mecanismo de promoción de esclavos, para que uno de ellos sea el nuevo maestro. Dos posibles soluciones a este problema son la adopción de un mecanismo de coordinación para elegir aleatoriamente un esclavo o el uso de un mecanismo basado en prioridades, donde el esclavo con mayor prioridad promociona automáticamente a maestro.

Los dos mecanismos, tanto el de difusión de mensajes como el de páginas amarillas, han sido implementados con mayor o menor éxito. Actualmente, una de las prácticas más comunes para implementar un servicio de localización consiste en usar las capacidades de uno o varios *middlewares* de comunicaciones. Por ejemplo, el *middleware* ZeroC ICE implementa un mecanismo de localización completamente transparente para el usuario tanto a nivel de nombrado como de implementación de interfaces. Para ello, se hace uso del servicio *IceGrid*, el cual actúa como gestor en una arquitectura orientada a servicios, y de su componente *Registry*, el cual almacena la información de los distintos componentes de la plataforma. De este modo, el

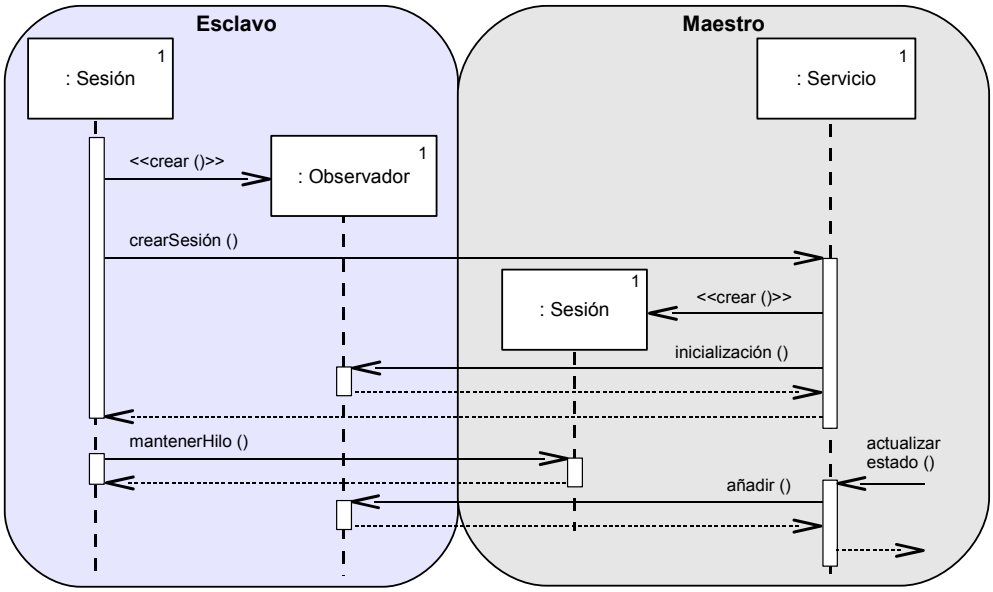


Figura 2.20: Esquema maestro-esclavos para replicar un servicio.

propio *middleware* consulta a *IceGrid* cuando sea necesario. Para solventar el problema asociado a un único punto de acceso, el propio *Registry* puede replicarse siguiendo un modelo maestro-esclavo para asegurar la disponibilidad del servicio de localización. Para ello, simplemente hay que ajustar unos parámetros de configuración y el propio *middleware* se encarga de la gestión del servicio de manera transparente tanto para los solicitantes de servicios como para los proveedores.

Otro servicio interesante en una arquitectura orientada a servicios es el servicio de **publicación** [GHJV95]. Mediante este servicio se establecen dos roles principales:

- **Publicador**. Este rol hace alusión a aquellos componentes del sistema que notifican una determinada información siguiendo unos ciertos criterios.
- **Subscritor**. Este rol está vinculado a aquellos componentes del sistema que reciben información de los publicadores.

Este servicio resulta también especialmente relevante en un sistema de vigilancia, en el que normalmente existen elementos que procesan información proveniente de múltiples fuentes. Es el caso de un ordenador central que procesa la información obtenida de diversos dispositivos de vigilancia, como por ejemplo cámaras o micrófonos. La primera aproximación para llevar a cabo la implementación de este servicio consistiría en establecer enlaces entre el publicador y los subscriptores (ver figura 2.21 izquierda), distribuyendo periódicamente la información a dichos subscriptores. Sin embargo,

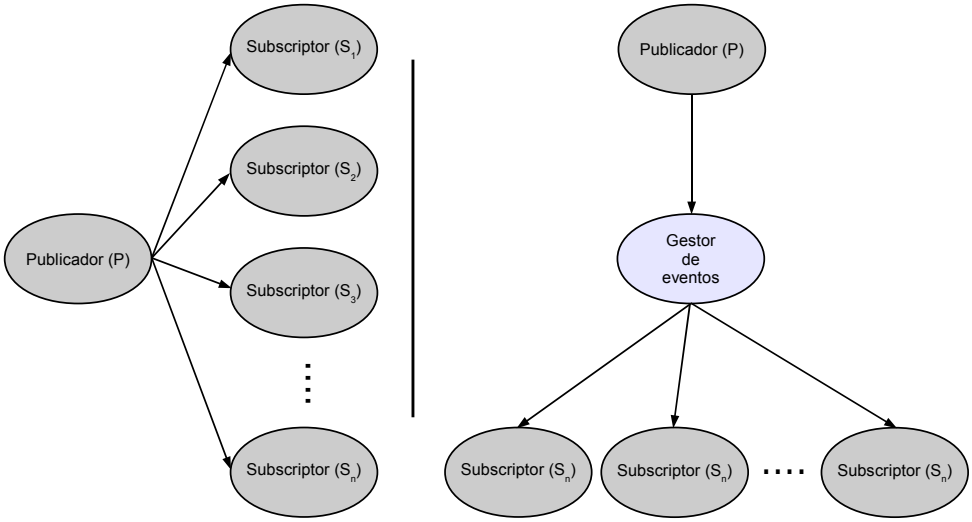


Figura 2.21: Izquierda: un publicador y múltiples subscriptores. Derecha: servicio de publicación y subscripción con un gestor de mensajes.

la principal desventaja de este enfoque reside en el fuerte acoplamiento del publicador respecto a los subscriptores, complicando la implementación del publicador ya que éste ha de gestionar los detalles del registro de los subscriptores, la entrega de los mensajes y los mecanismos de recuperación de errores. Una posible solución consistiría en incorporar un nuevo elemento a la arquitectura encargado de llevar a cabo esta funcionalidad y desacoplando al publicador de los subscriptores (ver figura 2.21 derecha).

Como ocurría con el servicio de localización, una opción para implementar un mecanismo de publicación consiste en elegir un *middleware* que ofrezca algún tipo de mecanismo de soporte para la publicación de eventos. Por ejemplo, el *middleware* ZeroC ICE proporciona el servicio *IceStorm* de publicación-subscripción, el cual facilita enormemente la gestión de este tipo de escenarios.

2.3.3. Lenguajes de descripción de interfaces

En la sección 2.3.1 se resumieron los principales requisitos que ha de cumplir una arquitectura orientada a servicios para garantizar la eficiencia y la completitud de la misma:

1. *Interoperabilidad* entre sistemas, lenguajes y plataformas empleados.
2. *Lenguaje de descripción de interfaces* claro, no ambiguo e independiente de la plataforma de ejecución.
3. Mecanismo de *recuperación de servicios*.

El primer punto se puede garantizar a través del uso de algún tipo de sistema que sea capaz de abstraer las características vinculadas a las distintas tecnologías a usar. Una opción muy común es utilizar un *middleware* de comunicaciones, estudiados en la sección 2.4 de este documento. El tercer punto se puede abordar gracias al uso de un mecanismo de localización de servicios en función de las características de los mismos. En la sección 2.3.2 se detallaron algunas de las posibles soluciones. Así, de este modo faltaría por estudiar más en detalle el lenguaje utilizado para definir las interfaces asociadas a los servicios del sistema, es decir, qué contratos ofrecen los distintos elementos que forman parte de la plataforma. Dichos lenguajes, normalmente referenciados como *Interface Description Languages* (IDLs) serán objeto de estudio de esta sección.

Desde un punto de vista general, los lenguajes de descripción de interfaces pueden tener una cierta variedad de propósitos. Sin embargo, si queremos definir un sistema en torno a los servicios que proporcionan sus componentes, el objetivo principal de un lenguaje de descripción de interfaces consiste en definir el sistema de tipos. De este modo se establecen los contratos vinculantes a los servicios, especificando cuáles son los parámetros de entrada y de salida de los mismos. Sin un sistema de tipos sería imposible que el solicitante de un servicio y su proveedor se pusieran de acuerdo sobre cómo se contratará el servicio o cuáles son los elementos que definen esa relación. Por ejemplo, se podría definir una operación que acepte una cadena de texto e invocarla con un entero, lo cual generaría un problema importante.

Además de esta parte básica del sistema de tipos, ciertos lenguajes de descripción de interfaces pueden proporcionar características más avanzadas, como por ejemplo la herencia de tipos de datos, nociones de orientación a objetos o polimorfismo. Por ejemplo, *Web Services Description Language* (WSDL) [CDK+02] no proporciona nociones de orientación a objetos mientras que *Slice* (IDL de ZeroC ICE [Hen04b]) sí. Estos conceptos estarían vinculados a los elementos básicos de un lenguaje de descripción de interfaces. Sin embargo, y desde una perspectiva más práctica, los propios lenguajes pueden servir para alcanzar una serie de objetivos.

Los lenguajes de generación de interfaces pueden servir para definir las APIs. Un compilador podría tomar como entrada una descripción del sistema con un lenguaje de descripción de interfaces y generar el código fuente para un determinado lenguaje de programación, como por ejemplo C++. Este código generado proporcionaría la API para ser usada desde el código de la aplicación con el objetivo de manipular valores, invocar operaciones, manejar excepciones, etc. Todas las reglas que definen cómo se genera una API en un determinado lenguaje de programación a partir de una descripción neutra se denominan *language mappings*.

De nuevo, existen determinados lenguajes que están relacionados con estas reglas de generación de código, mientras que otros no. Por ejemplo, WSDL

no define *language mappings*, es decir, la especificación no dice nada sobre cómo generar código fuente a partir de una descripción de un servicio web. En otras palabras, el estándar de WSDL [Wor01] define el propio lenguaje de descripción de interfaces pero no regula cómo generar código fuente. Este hecho genera un problema de interoperabilidad, ya que una determinada especificación relativa a un cierto lenguaje de programación puede que no sea interoperable con otra especificación que se base en la misma descripción con WSDL.

Otro elemento importante es la definición de protocolos con un lenguaje de descripción de interfaces. Dichos protocolos determinan aspectos relativos a la codificación de los mensajes y a las reglas asociadas a cómo se efectúa el intercambio de mensajes. El punto crítico relacionado con este aspecto consiste en decidir dónde se indican estos criterios. Hay dos opciones posibles:

1. Incluirlos en el propio lenguaje de descripción de interfaces.
2. Especificarlos de manera independiente a dicho lenguaje.

Lenguajes como WSDL incluyen las cuestiones del protocolo de transporte dentro de las propias definiciones, mientras otros como CORBA IDL lo separan. Así mismo, y relacionado con estos conceptos, también se puede tener en cuenta la elección del protocolo de transporte en sí mismo, como por ejemplo TCP o UDP. Una vez más, las opciones son incluir estos aspectos dentro de la propia definición con el lenguaje de descripción. Idealmente, las cuestiones que no están relacionadas con la descripción de los servicios y los tipos de datos deberían ser independientes del lenguaje de descripción.

Actualmente, WSDL está cobrando una especial importancia a la hora de definir servicios, dentro el marco de los servicios web [Jon05] [CDK+02] [PG03]. Los principales argumentos de esta tendencia se basan en la existencia de un estándar, el uso de la tecnología XML como mecanismo de difusión de información y, en general, la utilización de tecnologías vinculadas a Internet, como por ejemplo HTTP. A continuación, se estudiarán los elementos relacionados con los servicios web o *web services*, como el lenguaje de descripción de interfaces (WSDL) o el protocolo de transporte (SOAP), entre otros.

El marco de desarrollo de los *web services* gira en torno a tres áreas [CDK+02]: protocolos de comunicación, descripción de servicios y descubrimiento de servicios:

- *Simple Object Access Protocol* (SOAP), como protocolo de comunicación entre servicios web.
- *Web Services Description Language* (WSDL), como descripción formal y estándar de los servicios web.

```
<message name="GetFlightInfoInput">
 <part name="airlineName" type="xsd:string"/>
 <part name="flightNumber" type="xsd:int"/>
</message>

<message name="GetFlightInfoOutput">
 <part name="flightInfo" type="fixsd:FlightInfoType"/>
</message>

<message name="CheckInInput">
 <part name="body" element="eticketxsd:Ticket"/>
</message>

<portType name="AirportServicePortType">
 <operation name="GetFlightInfo">
  <input message="tns:GetFlightInfoInput"/>
  <output message="tns:GetFlightInfoOutput"/>
 </operation>
 <operation name="CheckIn">
  <input message="tns:CheckInInput"/>
</operation>
</portType>
```

```
POST /travelservice
SOAPAction: "http://www.acme-travel.com/flightinfo"
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP:Envelope xmlns:SOAP=
  "http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP:Body>
  <m:GetFlightInfo
   xmlns:m="http://www.acme-travel.com/flightinfo"
   SOAP:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance">
   <airlineName xsi:type="xsd:string">UL
   </airlineName>
    <flightNumber xsi:type="xsd:int">506
    </flightNumber>
  </m:GetFlightInfo>
 </SOAP:Body>
</SOAP:Envelope>
```

Figura 2.22: Izquierda: ejemplo de interfaz abstracta con WSDL. **Derecha**: ejemplo de llamada RPC con SOAP (comprobando la hora de un vuelo).

- *Universal Description, Discovery, and Integration directory* (UDDI), como registro de la descripción de los servicios web.

SOAP [Wor07] es un protocolo basado en XML desarrollado inicialmente por Microsoft, en colaboración con IBM, Lotus y otras entidades. En lugar de definir un protocolo de transporte nuevo, SOAP se apoyó en protocolos ya existentes y ampliamentes usados en los entorno web, como HTTP o SMTP. La estructura de un mensaje SOAP es simple y consta de dos elementos principales (ver figura 2.22 derecha): la cabecera y el cuerpo. Debido a que SOAP no impone la utilización de una implementación del tipo RPC/RMI, es necesario definir un protocolo RPC (o utilizar uno existente) que cubra los siguientes aspectos:

- Cómo se relacionan los tipos de datos definidos en XML, por una parte, y en la representación de la aplicación (por ejemplo Java), por otra.
- En qué parte del mensaje se definen cada uno de los elementos de una invocación remota (identidad del objeto, nombre de la operación y parámetros).

En los *Web Services*, SOAP ofrece la comunicación básica, pero no especifica qué mensajes se han de intercambiar para interactuar de manera adecuada con un servicio. Ese rol lo asume WSDL, un formato XML desarrollado por IBM y Microsoft para describir los servicios web como una colección de elementos de comunicación que pueden intercambiarse mensajes. En otras palabras, WSDL define la interfaz de un servicio web y proporciona

a los usuarios un punto de contacto para dicho servicio (ver figura 2.22 izquierda). Una descripción completa de un servicio con WSDL proporciona la siguiente información:

- Descripción del servicio a nivel de aplicación.
- Detalles dependientes del protocolo de comunicación empleado para acceder al servicio.

WSDL proporciona una descripción formal de las interacciones existentes entre los clientes y los servidores a través de la definición de interfaces. En la fase de desarrollo se pueden utilizar generadores de código para que, a partir de la descripción en WSDL, se obtenga el código de la aplicación que represente los requisitos de los servicios. En teoría, sólo sería necesario conocer la descripción de un servicio con WSDL, pero debido a que no existen especificaciones sobre cómo definir los *language mappings*, se pueden producir problemas de interoperabilidad entre distintas implementaciones que partan de la definición de un mismo servicio.

Finalmente, y dentro del marco de los servicios web, se pueden utilizar las especificaciones UDDI [Ope08] como mecanismo de localización de proveedores de servicios. UDDI proporciona dos especificaciones básicas para definir la estructura de un registro de servicios:

- La definición de la información a proporcionar por cada servicio.
- Una API de consulta y actualización del registro de servicios, que define cómo se accede y se actualiza dicha información.

Una vez discutida la estructura general de los servicios web y sus principales componentes, destacando el lenguaje de descripción de interfaces WSDL, resulta importante destacar el uso de otro tipo de lenguajes de descripción de interfaces, como por ejemplo Slice de ZeroC ICE [Hen04b] o CORBA IDL [HV99], debido a que han sido concebidos desde aproximaciones muy distintas a WSDL. A continuación se estudiará Slice como ejemplo de este segundo enfoque de IDLs, siguiendo CORBA IDL un enfoque similar.

Al igual que WSDL, Slice es un lenguaje de descripción de interfaces independiente de distintos elementos tecnológicos subyacentes, como por ejemplo los lenguajes de programación o la plataforma de ejecución (no tanto de los protocolos de comunicación en el caso de WSDL). Sin embargo, la diferencia principal reside en que WSDL sigue un estándar y Slice no. El principal objetivo de Slice es proporcionar la abstracción necesaria para independizar la interfaz de un servicio y su implementación, estableciendo un contrato entre el cliente y el servidor. Posteriormente, se puede utilizar un compilador a un determinado lenguaje de programación para generar las APIs requeridas por la aplicación de usuario.

```
<?xml version="1.0"?>
<definitions name="StockQuote"
 targetNamespace=
  "http://example.com/stockquote.wsdl"
 xmlns:tns="http://example.com/stockquote.wsdl"
 xmlns:xsd1="http://example.com/stockquote.xsd"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/
soap/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
  <schema targetNamespace=
   http://example.com/stockquote.xsd
   xmlns="http://www.w3.org/2000/10/XMLSchema">
   <element name="TradePriceRequest">
    <complexType>
      <all>
       <element name="tickerSymbol"
        type="string"/>
      </all>
    </complexType>
   </element>
   <element name="TradePrice">
    <complexType>
     <all>
      <element name="price" type="float"/>
     </all>
    </complexType>
   </element>
  </schema>
</types>
  <message name="GetLastTradePriceInput">
   <part name="body" element=
    "xsd1:TradePriceRequest"/>
  </message>
  ...
  <portType name="StockQuotePortType">
   <operation name="GetLastTradePrice">
    <input message="tns:GetLastTradePriceInput"/>
    <output message="tns:GetLastTradePriceOutput"
    />
   </operation>
  </portType>
  ...
  <operation name="GetLastTradePrice">
   <soap:operation
    soapAction=
     "http://example.com/GetLastTradePrice"/>
    <input>
     <soap:body use="literal"/>
    </input>
    <output>
     <soap:body use="literal"/>
    </output>
   </operation>
  </binding>
  <service name="StockQuoteService">
   <documentation>My first service
   </documentation>
   <port name="StockQuotePort"
    binding="tns:StockQuoteBinding">
    <soap:address location=
     "http://example.com/stockquote"/>
   </port>
  </service>
</definitions>
```

```
// Interfaz con Slice.
interface StockQuoteService
{
  float GetLastTracePrice(string tickerSymbol);
};
```

Figura 2.23: Comparativa de la misma interfaz con WSDL (arriba) y Slice (abajo).

Como la mayoría de IDLs, Slice proporciona tipos de datos básicos (*int, bool, float...*), tipos de datos de usuario (estructuras, secuencias, diccionarios, interfaces o clases) y modificadores que actúan sobre las operaciones (invocaciones asíncronas, operaciones idempotentes...). Una de las diferencias más importantes con respecto a WSDL es la orientación a objetos, y es que Slice permite utilizar la herencia a nivel de interfaz y a nivel de clase, posibilitando el uso del polimorfismo y, por lo tanto, generando una semántica mucho más rica. La figura 2.23 muestra una misma interfaz definida con WSDL y Slice, mientras que la tabla 2.3.3 resume las principales diferencias de estos dos lenguajes de descripción de interfaces.

Una última cuestión a tener en cuenta respecto a la implementación de una arquitectura SOA puede ser la integración de distintas tecnologías. Por ejemplo, se podría pensar en un enfoque que aprovechara las ventajas de

Característica	**WSDL**	**Slice**
Language mappings	No	Sí
Orientación a objetos	No	Sí
Protocolo como parte del IDL	Sí	No
Serialización como parte del IDL	Sí	No
Protocolo de transporte como parte del IDL	Sí	No
Simplicidad	No	Sí
Claridad	Relativa	Sí
Ortogonalidad	Relativa	Sí
Sintaxis	XML	Propia

Tabla 2.4: Comparativa entre WSDL y Slice.

WSDL respecto a estándares y las ventajas de otro *middleware* más robusto, como por ejemplo ZeroC ICE. Una posible opción sería utilizar técnicas SDO (*Service Data Objects*) [BBNP03], enviando y devolviendo documentos XML a los objetos ICE. De este modo, se pueden crear *wrappers* de servicios web sobre objetos ICE y haciéndolos accesibles utilizando algún otro estándar. Mediante enfoques de este tipo, se podrían crear arquitecturas SOA que se beneficien de las ventajas de una y otra alternativa.

2.4. *Middlewares* de comunicaciones

2.4.1. Introducción y características

Se puede entender un ***middleware*** como un software de conectividad que hace posible que aplicaciones distribuidas pueden ejecutarse sobre distintas plataformas heterogéneas, es decir, sobre plataformas con distintos sistemas operativos, que usan distintos protocolos de red y que, incluso, involucran distintos lenguajes de programación en la aplicación distribuida [CDK05].

Tradicionalmente, el *middleware* se define como la capa de comunicaciones existente entre el sistema operativo y las aplicaciones de usuario, cuya función principal consiste en proporcionar los mecanismos de comunicación necesarios entre los distintos elementos de un sistema distribuido (ver figura 2.24). De hecho, uno de los objetivos de un *middleware* es ofrecer un acuerdo en las interfaces y en los mecanismos de interoperabilidad, como contrapartida de los distintos desacuerdos en hardware, sistemas operativos, protocolos de red y lenguajes de programación.

Sin embargo, esta definición de *middleware* se ha ido extendiendo a distintos ámbitos, por lo que es posible establecer una **clasificación** de *middlewares* en función de su aplicación e implementación:

- ***Middleware* de comunicaciones**. Está asociado a la definición más tradicional de *middleware* y pretende abstraer a las aplicaciones distribuidas de las distintas redes de comunicaciones empleadas y sus protocolos asociados. A su vez, dentro de esta categoría se distinguen los sistemas de invocación a procedimientos remotos o a métodos remotos (RPC/RMI), los sistemas de colas de mensajes (MQ) y los sistemas orientados al procesado paralelo de datos (DRI, PAS).
- **Servidores de aplicaciones**. Su objetivo consiste en facilitar la ejecución simultánea y la interacción de varias aplicaciones en un mismo servidor. Actualmente, también son catalogados como *middleware*.
- ***Middlewares* a nivel de aplicación para bases de datos**. Por último, existe cierto número de *middlewares* que se ocupan fundamentalmente de facilitar la interacción entre las aplicaciones y bases de datos (ADO.NET, JDBC, ODBC, etc).

La mayoría de las clasificaciones de *middlewares* suelen ser incompletas debido a la propia generalidad que ha adquirido el concepto de *middleware*. Por ejemplo, un *middleware* puede tratar de abstraer las peculiaridades de un determinado hardware (UPnP [UPn08]), proporcionar una simplificación en el desarrollo de aplicaciones distribuidas (CORBA [Obj08b]) acercando el modelo de programación distribuido a un modelo más tradicional o local, ofrecer una interfaz de mayor abstracción (*Entreprise Java Beans*

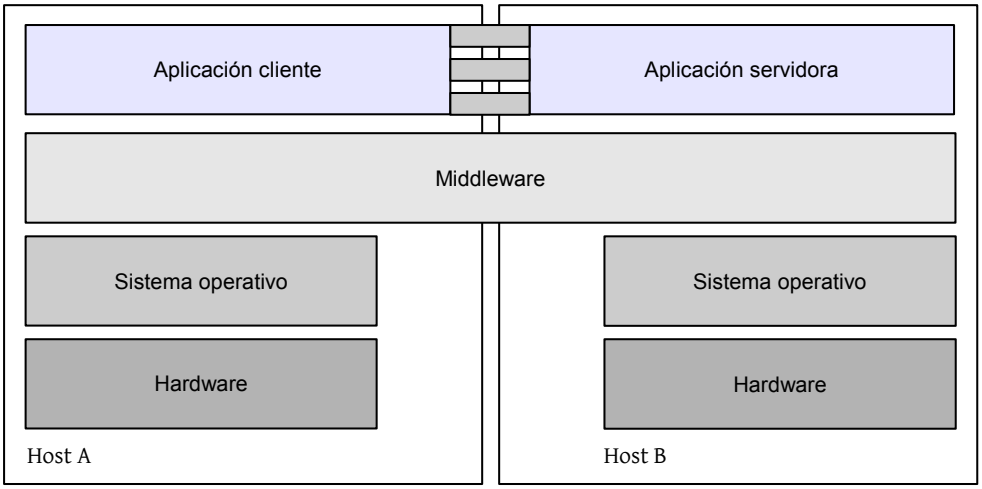


Figura 2.24: *Middleware* como mecanismo de abstracción.

[Sun06]), o incluso facilitar la gestión y administración de software (*Globus Toolkit* [Fos06]).

En la siguiente sección se estudiarán los distintos *middlewares* aplicables a este trabajo de investigación y se comentarán las principales características de cada uno de ellos. Antes de llevar a cabo este estudio, es importante destacar que la arquitectura de aplicaciones distribuidas es un aspecto importante de cara a implementar un sistema distribuido, pero no tanto en la elección del *middleware*. Esto es debido a que, de manera general e independiente del *middleware* seleccionado, se puede adaptar casi cualquier tipo de arquitectura. En este contexto, uno de las arquitecturas más relevantes en el desarrollo de aplicaciones distribuidas en Internet es REST (*Representational State Transfer*) [Fie00].

2.4.2. Alternativas actuales

En esta sección se resumen las principales características de un conjunto de *middlewares* disponibles en la actualidad y relevantes para este trabajo. El objetivo de esta sección es poner de manifiesto las principales características de cada uno de ellos y proporcionar las referencias más relevantes para un estudio más detallado de los mismos.

ZeroC ICE

ICE (*Internet Communication Engine*) [Hen04a] es un *middleware* orientado a objetos, es decir, ICE proporciona herramientas, *APIs*, y soporte de

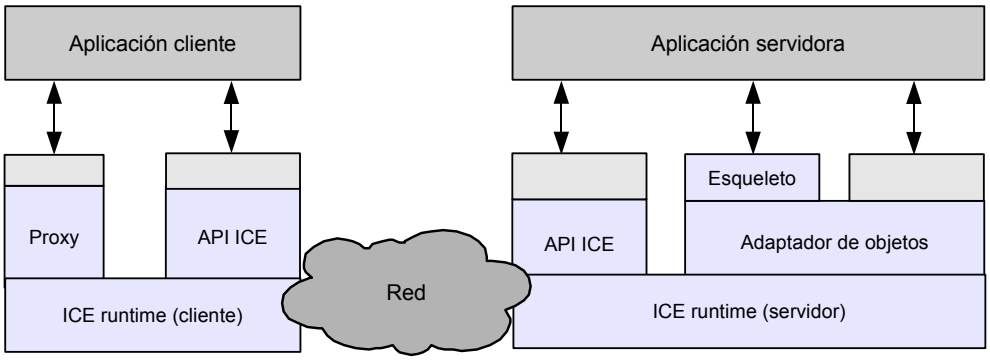


Figura 2.25: Arquitectura abstracta de ZeroC ICE.

bibliotecas para construir aplicaciones cliente-servidor orientadas a objetos (ver figura 2.25). Una aplicación ICE se puede usar en entornos heterogéneos: los clientes y los servidores pueden escribirse en diferentes lenguajes de programación, pueden ejecutarse en distintos sistemas operativos y en distintas arquitecturas, y pueden comunicarse empleando diferentes tecnologías de red (ver tabla 2.5). Además, el código fuente de estas aplicaciones puede portarse de manera independiente al entorno de desarrollo.

Los principales **objetivos de diseño** de ICE son los siguientes:

- Proporcionar un *middleware* listo para usarse en sistemas heterogéneos.
- Proveer un conjunto completo de características que soporten el desarrollo de aplicaciones distribuidas reales en un amplio rango de dominios.
- Evitar una complejidad innecesaria, haciendo que ICE sea fácil de aprender y de usar.
- Proporcionar una implementación eficiente en ancho de banda, en uso de memoria y en carga de CPU.
- Proporcionar una implementación basada en la seguridad, de forma que se pueda usar sobre redes no seguras.

Se puede decir que la **filosofía** de ICE se basa en construir una plataforma tan potente como CORBA, pero con una menor complejidad y una alta cohesión para facilitar tanto su aprendizaje como su uso [Hen06]. Fundamentalmente, las diferencias se aprecian en los siguientes aspectos:

- Mejora de prestaciones, ya que simplifica el protocolo de comunicación y las reglas de *marshalling*. Por otra parte, el diseño del *middleware* exprime las ventajas de la programación multi-hilo, al mismo tiempo

Nombre	Internet Communications Engine (ICE)
Definido por	ZeroC Inc. (`http://www.zeroc.com`)
Documentación	`http://zeroc.com/doc/latest/manual/`
Lenguajes	C++, Java, C#, Visual Basic, Python, PHP, Ruby
Plataformas	Windows, Windows CE, Unix, GNU/Linux, *BSD OSX, Symbian OS, J2RE 1.4 o superior, J2ME
Destacado	APIs claras y bien diseñadas Conjunto de servicios muy cohesionados Despliegue, persistencia, cifrado...
Descargas	`http://zeroc.com/download.html`

Tabla 2.5: ZeroC ICE. Resumen de características.

que mantiene una API extremadamente sencilla. Al contrario que ocurre en el caso de CORBA, la programación con interfaces asíncronos no involucra a los dos extremos de la comunicación, sino que el propio *middleware* ofrece dos interfaces de programación independientes: uno síncrono y otro asíncrono.

- La ausencia de un comité de estandarización lleva a un diseño mucho más limpio, guiado exclusivamente por criterios técnicos.
- La seguridad es un elemento fundamental de ICE, mientras que en CORBA se han generado posteriormente una secuencia de especificaciones que no han sido globalmente adoptadas por todos los fabricantes.
- La persistencia de objetos es prácticamente automática en ICE, y a partir de la especificación del estado de los objetos en el lenguaje de descripción de interfaces. CORBA introduce características similares a partir de la introducción de CCM en CORBA3 pero con una API mucho más compleja.
- ICE mantiene una arquitectura modular, lo que permite ampliar la arquitectura con nuevos protocolos de transporte, nuevas tecnologías de red y nuevos servicios sin afectar al resto de aplicaciones. Actualmente, ZeroC ICE incluye tres protocolos de transporte con diversas garantías de entrega (TCP, UDP y SSL). Por otra parte, la posibilidad de utilizar protocolos no confirmados como UDP es algo que en principio no está contemplado en CORBA, aunque algunas implementaciones de CORBA añaden mecanismos similares.

El núcleo de ICE proporciona una sofisticada plataforma cliente-servidor para el desarrollo de aplicaciones distribuidas. Sin embargo, las aplicaciones reales necesitan normalmente algo más que las capacidades remotas, como por ejemplo la activación de servidores bajo demanda, la distribución de proxies a los clientes, la distribución de eventos asíncronos, la configuración de aplicaciones, etc. ICE maneja una serie de **servicios** que gestionan estas características y otras (ver tabla 2.6). Dichos servicios se implementan como servidores ICE de forma que la aplicación desarrollada actúe como

IceGrid	Servidor de activación y despliegue de herramientas de computación Grid.
IceBox	Servidor de aplicaciones ICE.
IceStorm	Servicio de publicación-subscripción.
IcePatch2	Servicio de distribución de archivos
Glacier2	Servicio de cortafuegos de ICE.

Tabla 2.6: ZeroC ICE. Servicios avanzados.

un cliente. Ninguno de estos servicios utilizan características internas a ICE ocultas a la aplicación en cuestión, por lo que en teoría es posible desarrollar servicios equivalentes por parte del desarrollador.

Sin embargo, mantener estos servicios disponibles como parte de la plataforma permite al desarrollador centrarse en el desarrollo de la aplicación en lugar de construir la infraestructura necesaria en primer lugar. Además, la construcción de dichos servicios no es un esfuerzo trivial, por lo que es aconsejable conocer y usar los servicios disponibles en lugar de reinventar la rueda en cada aplicación. Los servicios avanzados de ICE se resumen en la tabla 2.6.

CORBA

CORBA [Obj08b] *Common Object Request Broker Architecture* es un estándar que establece una plataforma para el desarrollo de sistemas distribuidos con el objetivo de facilitar la invocación a procedimientos remotos y utilizar un paradigma de orientación a objetos (ver tabla 2.7). CORBA utiliza el protocolo estándar IIOP [Obj08c], de forma que un programa que utilice CORBA de cualquier vendedor, en cualquier computador, lenguaje de programación y red puede interoperar con otro programa del mismo u otro vendedor. CORBA es una de las arquitecturas de objetos distribuidos más madura. Implementa el concepto de invocación remota a métodos con interfaces síncronas y asíncronas.

La madurez de esta arquitectura implica por un lado la existencia de multitud de especificaciones estándar para casi cualquier servicio o patrón de uso, pero por otro lado acarrea el lastre de la compatibilidad hacia atrás. Por ejemplo, la compatibilidad hacia atrás del protocolo inicialmente definido en CORBA (GIOP 1.0) impide extensiones que faciliten el manejo de los mensajes sin conocer la estructura. Esto tiene implicaciones en el servicio de eventos, complicándolo innecesariamente, y terminó con extensiones del sistema de tipos muy poco elegantes[4].

Dentro del consorcio **OMG**[5] *Object Management Group*, CORBA representa la arquitectura de comunicaciones. Su objetivo es proporcionar un *bus*

[4]Un ejemplo típico es *CORBA::Any*.
[5]http://www.omg.org

Nombre	Common Object Request Broker Architecture
Definido por	Object Management Group (http://www.omg.org)

Documentación	`http://www.omg.org/docs/formal/04-03-12.pdf`
	`http://www.omg.org/docs/formal/06-04-01.pdf`
	Advanced CORBA Programming with C++ [HV99]

Lenguajes	C, C++, Java, COBOL, SmallTalk, Ada, Lisp, Python, IDLscript, PL/1, ...
Plataformas	Windows, Unix, GNU/Linux, *BSD, OSX, J2RE 1.3 o superior, Windows CE, J2ME, .NET, ...
Destacado	Multitud de fabricantes Arquitectura muy madura Muchos servicios y documentación
Descargas	`http://deuce.doc.wustl.edu/` TAO, CIAO `http://www.gnome.org/projects/ORBit2/` ORBit2, pyORBit `http://www.java.com/` JDK (>1.3) `http://www.jacorb.org/` JacORB `http://omniorb.sf.net/` OmniORB

Tabla 2.7: CORBA. Resumen de características.

software independiente de quién fabricó la aplicación, qué plataforma o plataformas la soportan, cómo se programó, y dónde se ejecuta.

Una de las características más destacables de CORBA es el gran número de implementaciones libres disponibles en la actualidad. Por desgracia, el grado de uniformidad en el conjunto de características soportadas no es muy grande. La mayoría de las implementaciones soportan un único lenguaje de programación o, como mucho, un par.

Las aplicaciones CORBA están compuestas de objetos, que se definen como unidades individuales de ejecución que combinan funcionalidad y datos, distinguiéndose los siguientes actores:

- Los **objetos**, cuya funcionalidad se describe a través de su interfaz en un IDL (*Interface Description Language*).
- Los **clientes** que invocan métodos sobre los objetos de manera transparente.
- El **ORB** (*Object Request Broker*), que es el mediador entre los clientes y los objetos encargado de garantizar la interoperabilidad.

Web services

El consorcio W3 está definiendo desde hace algún tiempo una arquitectura de servicios distribuidos alrededor de estándares muy populares en Internet, como por ejemplo XML o HTTP. El resultado, denominado ***Web Services***

Nombre	Web Services
Definido por	World Wide Web Consortium, W3C (`http://www.w3c.org`)
Documentación	`http://www.w3.org/TR/ws-arch/`
Lenguajes	C++, Java, C#, Visual Basic, Python, PHP
Plataformas	Windows, Windows CE, Unix, GNU/Linux, *BSD OSX, Symbian OS, J2RE 1.4 o superior, J2ME
Destacado	Mucho interés industrial
Descargas	`http://java.sun.com/webservices/` GlassFish, JAX-WS, JAX-WSA, JAXB, XWSS (Java) `http://celtix.objectweb.org/` Celtix ESB (Java) `http://labs.jboss.com/portal/jbossws/` JBoss Web Services `http://www.iona.com/products/artix/` Artix ESB (Java, C++, ...) `http://ws.apache.org/axis2/` Apache Axis2 (Java)

Tabla 2.8: Web Services. Resumen de características.

Architecture [Wor09] es un conjunto de recomendaciones que pueden utilizarse con independencia unas de otras. Al contrario que la mayoría de las plataformas de sistemas distribuidos, no se intenta estandarizar la API para ningún lenguaje de programación y tampoco se define un conjunto obligatorio de recomendaciones para una determinada implementación. Técnicamente, este hecho se justifica con el objetivo de obtener una mayor libertad. Esto ha llevado a un amplio rango de implementaciones parciales con infinidad de interfaces de programación (ver tabla 2.8). De hecho, cada fabricante tiene una media de dos APIs diferentes para implementar *Web Services*.

La tecnología *Web Service*, además de soportar el concepto de servicio, tiene asociados conceptos de caracterización y descubrimiento de servicios a través de la Web Semántica que, junto a la capacidades de gestión de servicios, definen un entorno de desarrollo de aplicaciones dinámico y flexible. Además, abren un amplio abanico de posibilidades para que, dentro de un navegador web, se desarrollen distintas herramientas que permitan el descubrimiento, gestión e invocación de servicios web de manera dinámica.

Las principales desventajas de esta propuesta son el bajo rendimiento (aproximadamente dos órdenes de magnitud inferior que arquitecturas con un formato de mensajes binario), la inmadurez, la falta de soporte para orientación a objetos y la falta de estandarización de las APIs.

.NET *Remoting*

.NET Remoting [MWN02] es un entorno orientado a objetos para el desarrollo de sistemas distribuidos. Permite la implementación de RMI/RPC sobre cualquier canal que cumpla con las especificaciones de la API de .NET

Nombre	.NET Remoting
Definido por	Microsoft (www.microsoft.com)
Documentación	http://msdn2.microsoft.com/en-us/library/72x4h507.aspx http://msdn2.microsoft.com/en-us/library/aa185916.aspx .NET Remoting [MWN02]
Lenguajes	C++, Java, C#, Visual Basic, Python, Eiffel, etc
Plataformas	Windows, Windows CE, Unix, GNU/Linux, *BSD
Destacado	Arquitectura elegante API sencillo Integración con WS y DCOM Fuertemente ligado a la plataforma .NET
Descargas	http://go-mono.com/ Versión libre de .NET http://www.microsoft.com/ .NET 2.0 Framework

Tabla 2.9: .NET Remoting. Resumen de características.

Remoting. .NET dispone de tres implementaciones de dicha API: canales TCP/IP con codificación binaria, canales HTTP con codificación según SOAP y canales internos que utilizan los mecanismos de comunicación entre procesos (IPC) de Windows (ver tabla 2.9).

La característica más relevante de esta arquitectura es que el propio usuario puede definir sus propios canales utilizando cualquier protocolo de transporte y cualquier codificación de mensajes que cumpla con los requisitos de la arquitectura. Por otra parte, .NET Remoting proporciona mecanismos de activación implícita, de persistencia, de seguridad y de gestión del ciclo de vida. El número de servicios disponibles es relativamente limitado pero está muy bien integrado dentro de la arquitectura .NET.

Para llevar a cabo el envío de mensajes .NET Remoting necesita un objeto canal, entre los que destacan HTTP/SOAP, TCP/binario e IPC/binario. Cada canal es capaz de establecer las conexiones, de traducir la invocación a un mensaje y de traducir el resultado a un mensaje. Como especificación de la interfaz, .NET Remoting permite utilizar cualquier lenguaje soportado por .NET. Además, no es necesario compilar la interfaz y no se necesitan convenios especiales de nombrado.

Enterprise Java Beans

Enterprise Java Beans [Sun06] proporciona un modelo de componentes a la plataforma Java. Hace especial énfasis en la persistencia, el despliegue, las transacciones y la gestión de dependencias (ver tabla 2.10). Utiliza JNDI [6] (*Java Naming and Directory Interface*) para nombrado y descubrimiento de servicios y JMS (*Java Message Service*) para la propagación de eventos. La persistencia de objetos se realiza mediante una API bien conocida, que permite al usuario personalizar los mecanismos. La comunicación remota se

[6] http://java.sun.com/products/jndi/

Nombre	Enterprise Java Beans
Definido por	Sun Microsystems (`http://java.sun.com`)
Documentación	`http://java.sun.com/javaee/reference/` `http://java.sun.com/products/ejb/docs.html`
Lenguajes	Java
Plataformas	Windows, Windows CE, Unix, GNU/Linux, *BSD
Destacado	Persistencia automática en cualquier BD con interfaz JDBC API muy sencillo Integración con JSP, Java Faces, ...
Descargas	`http://java.sun.com/javaee/downloads/` Java EE 5 SDK `http://jboss.org` RedHat Application Server (EJB3)

Tabla 2.10: Enterprise Java Beans. Resumen de características.

realiza mediante el mismo protocolo de CORBA (IIOP), lo que permite cierto nivel de interoperabilidad. La definición de Web Services utilizando EJB también es relativamente sencilla.

Globus Toolkit

Debido a que el concepto de grid está vinculado con un sistema distribuido heterogéneo, **Globus Toolkit** (ver tabla 2.11) [Fos06] también se puede estudiar como un *middleware* que se encarga de solucionar problemas parecidos a los que ya trataban arquitecturas como CORBA. La diferencia fundamental está en la función principal de este tipo de *middlewares*, que se ocupan especialmente de la gestión de la ejecución remota sobre grandes volúmenes de datos o de cálculo. Los *middlewares* para sistemas grid todavía no están maduros y carecen de muchos servicios elementales.

Globus Toolkit es una implementación de la arquitectura OGSA (Open Grid Services Architecture) [FKNT02] definida por el *Global Grid Forum*[7]. La mayoría de los servicios utilizan Web Services para el acceso a los mismos, y siguen los estándares del *Web Services Resource Framework*[8] en desarrollo dentro de OASIS[9].

Jini

El sistema **Jini** (*Java Intelligent Network Infrastructure*) es una arquitectura para sistemas distribuidos basados en Java, que explota la idea de grupos

[7] http://www.gridforum.org/
[8] http://www.globus.org/wsrf/
[9] http://www.oasis-open.org/home/index.php

Nombre	Globus Toolkit (GT)
Definido por	Globus Alliance (http://www.globus.org)
Documentación	http://www.globus.org/toolkit/docs/
Lenguajes	Java, Python, C, C++
Plataformas	Windows, Unix, GNU/Linux, *BSD
Destacado	Control de la ejecución remota Monitorización y seguridad Replicación de datos
Descargas	http://www.globus.org/toolkit/ Globus Toolkit http://www-itg.lbl.gov/gtg/projects/pyGlobus/ Python Interface http://www.gridway.org/ GridWay Meta-Scheduler http://www.gridlab.org/Software/ GridLab Software

Tabla 2.11: Globus Toolkit. Resumen de características.

de usuarios y recursos federados. Jini considera la incorporación a la red de dispositivos empotrados móviles y, por tanto, permite la adición y desactivación de servicios de manera muy flexible (ver tabla 2.12). Dos de los objetivos más importantes de Jini son los siguientes:

- Simplificar la tarea de construir, mantener y alterar una red de dispositivos, software y usuarios.
- Proporcionar a los usuarios la capacidad de acceder a los recursos desde cualquier punto de la red.

Jini extiende el entorno de aplicaciones de Java para sustituir el modelo de una única máquina virtual a una red de máquinas virtuales que interaccionan. El empleo de Java permite incorporar un buen número de ventajas. Tanto código como datos puedan migrarse de un dispositivo de cómputo a otro. Además, los mecanismos de seguridad permiten mantener cierto grado de confianza al ejecutar código que proviene de otros dispositivos de cómputo. También es posible aprovechar las características de introspección. El sistema de tipos de Java permite identificar la clase de un objeto aunque no sea originario del mismo dispositivo de cómputo.

Nombre	Jini (Java Intelligent Network Infrastructure)
Definido por	Sun Microsystems (http://java.sun.com)
Documentación	http://java.sun.com/products/jini/2_1index.html
Lenguajes	Java
Plataformas	Windows, Unix, GNU/Linux, *BSD
Destacado	Mecanismos híbridos de búsqueda Uso inteligente del mecanismo de federación
Descargas	http://java.sun.com/products/jini/ Jini, JavaSpaces

Tabla 2.12: Jini. Resumen de características.

El objetivo de Jini es federar grupos de dispositivos y componentes software en un único sistema distribuido dinámico. El sistema resultante es sencillo de acceder y fácil de administrar. La arquitectura de un sistema Jini está orientada a un grupo de trabajo. Se asume que los miembros de una federación han acordado las nociones de confianza, administración, identificación y políticas de uso.

2.4.3. Servicios relevantes para un sistema de vigilancia

Además de llevar a cabo la abstracción de las redes de comunicaciones subyacentes a un sistemas distribuido, un *middleware* debería proporcional idealmente una serie de servicios que facilitaran el desarrollo de aplicaciones distribuidas. En esta sección se describirán aquellos servicios más directamente relacionados con un sistema de vigilancia y, de manera indirecta, con los conceptos principales asociados a una arquitectura orientada a servicios (ver sección 2.3).

Localización

Uno de los servicios más interesantes e importantes en un sistema distribuido es la localización, tanto de servicios como de elementos o dispositivos de cómputo. La principal característica asociada al servicio de localización es la transparencia, es decir, la capacidad del *middleware* de abstraerse de la dirección física de un determinado dispositivo o servicio. Por ejemplo, resulta mucho más deseable acceder a un dispositivo por su nombre que por su dirección IP.

De todos los *middlewares* descritos en la sección anterior, el único que realmente proporciona un servicio de localización completamente transparente para el usuario es ICE. El servicio de localización está integrado dentro del servicio *IceGrid* a través del componente *Registry* y el *middleware* lo consulta automáticamente cuando lo necesita.

El despliegue, incluyendo el nombrado de objetos y la asignación de protocolos y puertos, es completamente ortogonal al desarrollo de la aplicación distribuida. Sólo los denominados *proxies indirectos* necesitan consultar el servicio *IceGrid.Registry*. En determinadas situaciones, resulta interesante buscar un servicio por la interfaz que implementa, es decir, por su funcionalidad. En esos casos, proporciona un interfaz de búsqueda muy sencillo en las tablas del servicio *IceGrid.Registry* (mediante el interfaz *IceGrid.Query*).

En CORBA la localización por nombre se realiza mediante la consulta directa del servicio de nombres. El servicio de nombres de CORBA es innecesariamente complejo puesto que en los sistemas reales muy pocos objetos necesitan realmente tener un nombre. De hecho CORBA dispone de otro ser-

vicio de nombrado mucho más simple que se utiliza (también explícitamente por las aplicaciones) para consultar determinados objetos especiales, pero que no puede utilizarse para los objetos de aplicación.

En EJB o Java RMI puede utilizarse el *RMI Registry*, un servicio similar a *IceGrid.Registry* pero cuya consulta debe ser explícita. En Web Services el mecanismo de localización más simple es proporcionado por *WS-Addressing*, en cuyo caso es preciso conocer el URI del servicio Web.

Despliegue y gestión remota

El hecho de ser capaz de desplegar y gestionar todos los dispositivos de un sistema de vigilancia de manera remota supone una importante restricción a la hora de seleccionar un *middleware*. De todas las plataformas consideradas, la única que tiene una infraestructura integral para todas las tareas de despliegue, incluido el transporte de los programas, verificación, seguridad, configuración, activación implícita, gestión remota de objetos en ejecución, etc. es ZeroC ICE.

En ICE el despliegue de una aplicación distribuida se describe en un archivo XML denominado descriptor de aplicación (ver listado de la siguiente página), incluyendo todos los nodos participantes, ejecutables, servidores, servicios y objetos bien conocidos. Este archivo es interpretado por las herramientas de administración del servicio *IceGrid* y almacenado de forma persistente por el propio *IceGrid*.

EJB, WS y en general todos los servidores de aplicaciones disponen de mecanismos para facilitar este despliegue (encapsulado en archivos comprimidos y firmados WAR, JAR, etc.) pero no disponen del control dinámico de la ejecución, dependencias, etc.

Activación implícita

En una arquitectura de servicios de vigilancia a gran escala se debe tener en cuenta la posibilidad de manejar miles de objetos. Ninguna arquitectura podría soportar un número ilimitado de objetos en cada elemento de cómputo pero, teniendo en cuenta que no todos los objetos se utilizan a la vez, es posible mejorar la escalabilidad enormemente con un mecanismo de activación implícita y almacenamiento persistente del estado de los objetos.

CORBA, EJB, ICE y .NET Remoting permiten un control preciso del modelo de ejecución de los objetos remotos. Los objetos pueden activarse bajo demanda en cada invocación, activarse bajo demanda solo la primera vez, o activarse de forma anticipada. Cualquiera de estos modelos son soportados por las plataformas mencionadas, pero sólo en ZeroC ICE los modos de activación son ortogonales a la codificación de la aplicación.

En CORBA y en ICE se utiliza el concepto de sirviente (*servant*) de manera independiente del concepto de objeto distribuido, de manera que un mismo sirviente puede implementar o respaldar distintos objetos o, de manera inversa, un mismo objeto puede ser respaldado en múltiples sirvientes. Esto proporciona transparencia de replicación en el lado del cliente y una escalabilidad muy superior.

Ejemplo de descriptor de IceGrid

```
<icegrid>
  <application name="Conversor">

    <replica-group id="ConversorAdapters">
      <load-balancing type="round-robin"/>
      <object identity="FabricaConversores"
              type="::Ripper::MP3FabricaConversores"/>
    </replica-group>

    <server-template id="PlantillaServidorConversor">
      <parameter name="index"/>
      <parameter name="exepath" default="./Servidor.py"/>
      <server id="Conversor${index}"
        exe="${exepath}"
        activation="on-demand">
        <adapter name="ConversorAdapter"
          replica-group="ConversorAdapters"
          register-process="true"
          endpoints="tcp"/>
      </server>
    </server-template>

    <node name="Nodo1">
      <server-instance template="PlantillaServidorConversor"
                       index="1"/>
    </node>

    <node name="Nodo2">
      <server-instance template="PlantillaServidorConversor"
                       index="2"/>
    </node>

  </application>
</icegrid>
```

Persistencia

La activación implícita no podría desarrollar todo su potencial sin un mecanismo que permitiera almacenar el estado de los objetos no usados, es decir, un mecanismo de persistencia de objetos. CORBA permite implementar objetos persistentes, pero el usuario debe añadir un buen número de líneas de código no triviales para ello. Sin embargo ICE y EJB implementan la persistencia de objetos con una mayor facilidad. En ICE la persistencia se implementa mediante el servicio *Freeze*. El estado de los objetos se define en el propio lenguaje de descripción de interfaces (*Slice*) y el código para almacenar y recuperar el estado de los objetos se genera automáticamente.

Balanceado de carga

Una arquitectura orientada a servicios que pretenda ser escalable debe soportar mecanismos de replicación, aunque sea de forma restrictiva, y balanceo de carga entre las diferentes réplicas. Teóricamente, CORBA soporta replicación y existe una especificación de tolerancia a fallos que lo incluye. Realmente existen muy pocas implementaciones que cuenten con esta característica. En cambio, ICE implementa un sistema de replicación en *IceGrid*, con diferentes políticas de balanceo de carga entre las réplicas. Además, en determinados casos (servicios sin estado), el desarrollo del servicio es prácticamente ortogonal a la configuración replicada.

2.4.4. Consideraciones finales

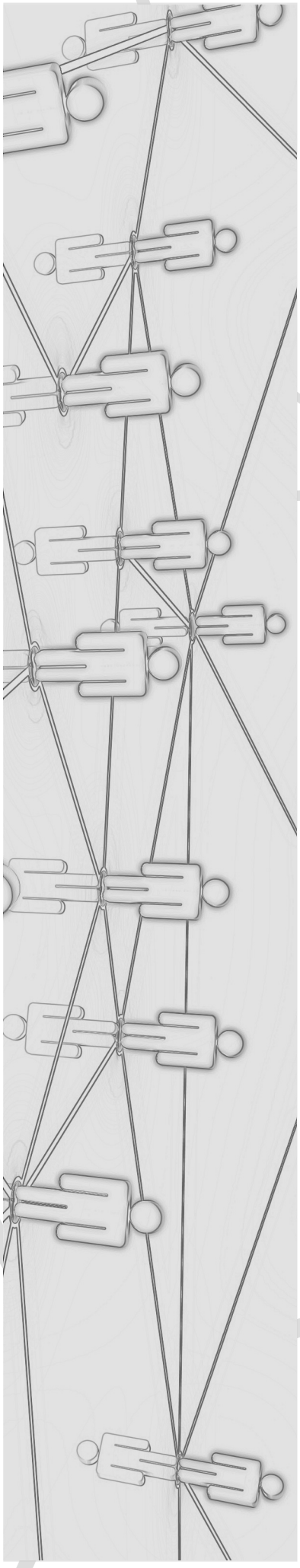
Los *middlewares* de comunicaciones representan una opción importante a la hora de adoptar herramientas transversales que den soporte a la arquitectura de un sistema de vigilancia. La principal ventaja de este enfoque reside en el uso de servicios proporcionados de manera nativa por el *middleware*, como se ha discutido en la sección anterior, y en las facilidades que proporciona un *framework* de comunicaciones de manera independiente al hardware (computadores, dispositivos de vigilancia, etc) y al software (sistemas operativos, protocolos de red, lenguajes de programación, etc) utilizados.

Sin embargo, y como se comenta en la sección 2.2.3, existen otras opciones para abordar la naturaleza distribuida que conlleva de manera inherente la evolución de los sistemas de vigilancia. Una de estas opciones alternativas es **MPI** (*Message Passing Interface*) [GLS99], que define la especificación para una API que permite la comunicación entre dispositivos de cómputo, normalmente utilizada en el ámbito de los clusters y los super-computadores. En este contexto, existen algunos trabajos [BHK+98, DMS+07] que hacen uso alguna implementación de esta especificación para llevar a cabo la comunicación entre los componentes del sistema de vigilancia.

En caso de utilizar un *middleware* de comunicaciones para soportar la arquitectura de un sistema de vigilancia, otra cuestión relevante es el rendimiento del mismo en dicho dominio. Existen algunos estudios e informes técnicos que profundizan en este aspecto, como el informe técnico [Hen09] publicado por ZeroC Inc. para comparar el rendimiento y la escalabilidad de ICE, *Windows Communication Foundation* y Java RMI. La misma compañía ha realizado estudios empíricos para evaluar ICE respecto a CORBA y SOAP [Hen09].

Capítulo 3

Arquitectura Propuesta



Capítulo 3
Arquitectura Propuesta

En este capítulo se estudia en profundidad la propuesta de arquitectura planteada en este trabajo para dar soporte al desarrollo y despliegue de sistemas escalables de vigilancia inteligente. Así mismo, se establece la formalización de la arquitectura multi-agente, se definen los agentes que la forman y los mecanismos de comunicación empleados para especificar las interacciones entre dichos agentes. También se profundiza en los procesos de vigilancia y de los pasos necesarios para utilizar la arquitectura en el proceso de despliegue de un sistema de vigilancia inteligente.

"Nada desaparece sin un rastro."
Albert Hosteen (Anasazi, The X-Files)

3

Arquitectura del Sistema de Vigilancia

En el capítulo 1 se introdujo la necesidad de diseñar y desarrollar sistemas de vigilancia inteligentes para tratar con la complejidad, cada vez mayor, de los actuales entornos de monitorización. En este contexto, dos de los principales retos de este tipo de sistemas son la gestión de un alto número de dispositivos o sensores de vigilancia y la integración de métodos de Inteligencia Artificial, los cuales permiten ofrecer un análisis más completo y sofisticado del entorno.

Desde un punto de vista general, la mayoría de las arquitecturas que dan soporte a estos sistemas de vigilancia están orientadas al problema particular que ha de resolverse, teniendo en cuenta las características del entorno concreto que se va a monitorizar. Este hecho limita la reutilización de estos sistemas a la hora de tratar problemas relacionados o en diferentes entornos que no tienen por qué ser los mismos para los que fueron diseñados inicialmente. Una posible solución a esta limitación es la adopción de arquitecturas flexibles y escalables basados en el uso de **conocimiento experto** [VBL+05], los cuales pueden contribuir a la construcción de sistemas de vigilancia aplicados a múltiples dominios y problemas.

La aplicación inteligente y posterior refinamiento de este conocimiento permitiría el despliegue de sistemas de vigilancia personalizados en función de las necesidades de monitorización y análisis de entornos. Para ello, la arquitectura que diera soporte a este enfoque debería posibilitar la instanciación de componentes de vigilancia a partir del conocimiento experto previamente adquirido o aprendido. Así mismo, dicha arquitectura proporcionaría los mecanismos de comunicación necesarios para que dichos componentes se comunicaran o cooperasen, ofreciendo un esquema lo suficientemente general y escalable, adaptable a distintos problemas y entornos. En este contexto, los **Sistemas Multi-Agente** [Wei99] se adaptan perfectamente a esta

problemática gracias al uso de entidades autónomas que permiten la gestión y distribución del conocimiento necesario para solucionar un problema global, en el que dichos agentes cooperan para proporcionar una solución conjunta.

La arquitectura para el desarrollo y despliegue de sistemas de vigilancia inteligentes propuesta en esta tesis está inspirada en los Sistemas Multi-Agente (SMA). Como se presentó en la sección 2.2.3, los SMA se han utilizado para solucionar diferentes problemas que se presentan en las distintas etapas del proceso de vigilancia, desde procesos de más bajo nivel como la segmentación hasta tareas de más alto nivel como la toma de decisiones y la gestión de crisis ante posibles amenazas.

Por otra parte, esta propuesta tiene como base teórica un modelo formal de conocimiento basado en el análisis de la normalidad en entornos de monitorización [AVJL+09], el cual se discute brevemente en la sección 3.3. Además de definir componentes de vigilancia, este modelo ha servido para extraer ciertos servicios de vigilancia que serán gestionados por el conjunto de agentes diseñados en este trabajo de investigación.

No obstante, y aunque la propuesta de arquitectura multi-agente de vigilancia esta inspirada en dicho modelo, el despliegue particular de la misma se puede realizar de manera indendiente al mismo. Esto se debe a que el conjunto de agentes diseñados están concebidos para realizar tareas muy generales, como por ejemplo el procesamiento de la información obtenida por los sensores, el análisis de normalidad y anormalidad o la fusión de información. Por lo tanto, la adopción de otros modelos de vigilancia inteligente es posible gracias al diseño de soluciones y mecanismos de comunicación generales.

3.1. Primera aproximación: arquitectura de un sistema de vigilancia cognitivo para un paso de peatones

En esta sección se describe una arquitectura multi-agente diseñada para llevar a cabo la vigilancia en un escenario de tráfico urbano [VAJ+09]. Este sistema cognitivo ha sido diseñado con la finalidad de detectar comportamientos anómalos de vehículos en un paso de peatones, el cual puede estar regulado o no por semáforo. El sistema propuesto consta de cuatro agentes de propósito específico responsables, entre otras tareas, de formalizar el conocimiento de dominio, de analizar cada una de las situaciones monitorizadas o de determinar la sanción para el conductor que no se comporte de acuerdo a la normalidad en dicho paso de peatones. Dichos agentes se comunican mediante un protocolo de interacción definido por FIPA [Fou00b] para solicitar la realización de acciones o tareas a los agentes.

Dentro del trabajo de investigación reflejado en esta tesis, esta primera aproximación ha servido principalmente para afrontar dos retos: i) el diseño y desarrollo de una arquitectura de vigilancia en un entorno de tráfico delimitado y concreto, ii) la obtención de experiencia a la hora de abordar el diseño de una arquitectura general y escalable que permita el despliegue de sistemas de vigilancia más sofisticados. Esta primera toma de contacto ha permitido además identificar las dificultades a la hora de desarrollar y desplegar un sistema multi-agente real de vigilancia, experiencia que se ha utilizado para definir el modelo de arquitectura que se discutirá más adelante en la sección 3.4.

3.1.1. Contexto y motivación

Los accidentes de tráfico causan anualmente una gran cantidad de heridos y fallecidos. En muchos de ellos están envueltos los peatones, los cuales representan una de las mayores fuentes de víctimas mortales. La tabla 3.1 muestra las cifras de accidentes en entornos urbanos en España a lo largo del año 2005, según la información obtenida directamente desde la Dirección General de Tráfico (DGT) [dT07]. Aunque los tipos de accidentes más comunes son las colisiones frontales y laterales, la tercera cifra negativa está representada por los **atropellos**, que suponen el 41 % de las víctimas mortales.

En este contexto, cobra especial importancia el escenario del **paso de peatones** como parte de la calzada que comparten tanto los vehículos como los peatones y que puede estar regulado o no por semáforo. Según la DGT, en el año 2005 se registraron 1.504 sanciones que tuvieron como elemento común un paso de peatones, y 2.647 en relación a semáforos. No obstante, estas cifras sólo representan las infracciones recogidas y sancionadas y no el número total de conductas irregulares que realmente se produjeron en todas las ciudad españolas.

En 2006, y con el objetivo de reducir estas trágicas cifras, las autoridades españolas aprobaron la puesta en funcionamiento del **permiso por puntos**, estableciendo una serie de infracciones que restarían puntos a aquellos

Tipo de accidente	Heridos		Víctimas mortales	
	Número	%	Número	%
Salida de la vía	2703	6 %	123	16 %
Colisión frontal	1456	3 %	45	6 %
Colisión lateral	20490	42 %	183	23 %
Colisión trasera	9078	19 %	42	5 %
Atropello	9070	19 %	326	41 %
Vuelco	2055	4 %	20	3 %
Otros	3711	8 %	51	6 %
Total	48563	100 %	790	100 %

Tabla 3.1: Accidentes en entornos urbanos en España durante el año 2005.

conductores que las cometieran[1]. De este modo, cada conductor parte inicialmente con un número fijo de puntos (8 para los conductores noveles y 12 para el resto) que se decrementa en caso de infringir alguna de las normas así especificadas en dicho sistema de puntos. El principal objetivo de dicho sistema es incrementar el grado de responsabilidad de los conductores, siempre con el objetivo final de reducir el número de accidentes y, por lo tanto, el número de heridos y víctimas.

La principal limitación de este tipo de alternativas es la necesidad de un policía o guardia de tráfico para detectar la mayoría de los comportamientos irregulares, haciendo inviable la dedicación de los recursos humanos necesarios para sancionar la mayor parte de estas conductas que suponen un riesgo para la seguridad vial. En el caso de utilizar monitores de televisión para vigilar determinadas escenas, también hay que tener en cuenta los problemas derivados del cansancio y la fatiga, como ya se introdujo y justificó en la sección 2.2.1. En el caso de los pasos de peatones, sobre todo los que no están regulados por semáforo, la detección de comportamientos incorrectos de vehículos y peatones requiere el continuo control de un agente de tráfico.

Desde una perspectiva general, existen distintas propuestas en el contexto de la gestión y la vigilancia del tráfico, como los trabajos desarrollados por Mohammadian [Moh06], Tomás y García [TG05] o Uddin y Shioyama [US05]. Estos sistemas están basados en el análisis de vídeo y, por lo tanto, la mayoría necesita una persona para detectar los comportamientos anómalos que se produzcan en la escena monitorizada. De este modo, este tipo de sistemas se pueden entender como sistemas de videovigilancia para la monitorización pasiva del tráfico.

La principal **motivación** del sistema de vigilancia cognitivo planteado en esta sección es precisamente invertir esta pasividad, de manera que el propio sistema sea capaz de detectar los comportamientos anómalos asociados a los vehículos que intervienen en un paso de peatones regulado o no por semáforo.

3.1.2. Descripción de la escena monitorizada

La escena elegida para desplegar el sistema de vigilancia cognitivo que analiza comportamientos de vehículos es un escenario común de tráfico urbano. Este escenario está formado por una calzada de doble sentido en la que hay un paso de peatones regulado por semáforo. La figura 3.1 ilustra los principales elementos de dicha escena. La elección de entorno se justifica a través de dos motivos: i) la importancia del uso de sistemas artificiales que posibiliten la reducción del número de accidentes de tráfico y ii) el hecho de afrontar un problema acotado cuya complejidad se puede incrementar de manera gradual.

[1] http://www.permisoporpuntos.com

Figura 3.1: Escenario de tráfico urbano a monitorizar.

El semáforo mantiene dos modos de funcionamiento: activo e inactivo. En el primero de ellos, el semáforo se puede encontrar en rojo, obligando a los vehículos a detenerse y permitiendo el paso de los peatones; ámbar, informando a los vehículos sobre el cambio inminente a rojo; y verde, habilitando la circulación de vehículos por la calzada y obligando a los peatones a esperar antes de poder cruzar. En el modo inactivo, el semáforo puede estar apagado o con la luz ámbar parpadeando y, como regla general, los vehículos tienen el deber de ceder el paso a los peatones a la hora de cruzar la calzada. El siguiente conjunto de reglas sirve para resumir la **normalidad del escenario** de tráfico y explicitar el comportamiento correcto de los conductores:

- Si el semáforo está en rojo, entonces los vehículos han de detenerse, independientemente de que haya peatones esperando para cruzar o no.
- Si el semáforo está en ámbar, entonces los vehículos pueden rebasar el semáforo o no en función de cuándo percibieron el cambio de verde a ámbar.
- Si el semáforo está en verde, entonces cualquier vehículo puede pasar.
- Si el semáforo está inactivo y hay peatones esperando para cruzar la calzada o cruzándola, entonces los vehículos han de parar para permitir el paso de peatones.
- Si un peatón está cruzando la calzada por el paso, entonces cualquier vehículo deberá parar para evitar un posible atropello.

3.1.3. Arquitectura del sistema de vigilancia

Desde un punto de vista funcional, el sistema de vigilancia desarrollado para vigilar el escenario de tráfico urbano descrito en la sección anterior está compuesto de los siguientes **agentes**: i) *Simulation Agent*, ii) *Formalizer Agent*, iii) *Environment Analyzer Agent*, y iv) *Decision Manager Agent* (ver figura 3.2). En realidad, cada uno de estos agentes es una especialización del agente básico definido de acuerdo a las especificaciones del comité FIPA [Fou04]. A continuación se resumen las principales responsabilidades de cada uno de estos agentes.

El ***Simulation Agent*** es el responsable de simular las distintas situaciones que se pueden dar en el escenario a monitorizar cuando así sea necesario, como por ejemplo a la hora de generar situaciones que no suelen ser comunes en la realidad (p.e. cuando un coche se salta el semáforo en rojo de manera descarada). De este modo, cuando el usuario del sistema solicita la simulación de un nuevo escenario, este agente recrea toda la información necesaria para su posterior análisis. Por lo tanto, el modelo de funcionamiento de este agente se basa en un esquema reactivo. La información del entorno comprende los distintos vehículos que aparecen en la misma, como por ejemplo coches o camiones, así como los distintos grupos de personas existentes. Por otra parte, este agente también se encarga de generar el estado del semáforo.

El ***Formalizer Agent*** es el encargado de formalizar la información generada por el *Simulation Agent*, utilizando para ello el lenguaje PROLOG y permitiendo la representación formal de dicha información. La finalidad de este enfoque es el de habilitar el posterior proceso de razonamiento y de análisis de la escena mediante un lenguaje formal. Desde el punto de vista del diseño, el propósito de la definición de este agente consiste en desacoplar los procesos de simulación de información y de formalización de la misma.

El ***Environment Analyzer Agent*** tiene como misión distinguir entre la normalidad y la anormalidad de una situación a partir de la información proporcionada por el *Analyzer Agent*. Este agente mantiene un modelo deliberativo basado en la ontología definida para representar el conocimiento asociado al entorno y a la detección de anomalías. Para llevar a cabo esta última tarea, el agente hace uso del motor de inferencia de PROLOG. En la tabla 3.2 se muestran algunas de las definiciones de la ontología utilizada por este agente.

Finalmente, el ***Decision Manager Agent*** es el responsable de tomar la decisión apropiada en caso de que el *Environment Analyzer Agent* detectara un comportamiento anómalo por parte de algún conductor. Este agente podría estar basado en algún tipo de algoritmo de toma de decisiones que sugiriese la multa correspondiente en función de la infracción cometida. Actualmente, este agente mantiene un sencillo sistema de reglas para llevar a cabo dicha tarea. En la tabla 3.3 se muestran algunas de las multas que se podrían asignar dependiendo de la infracción detectada.

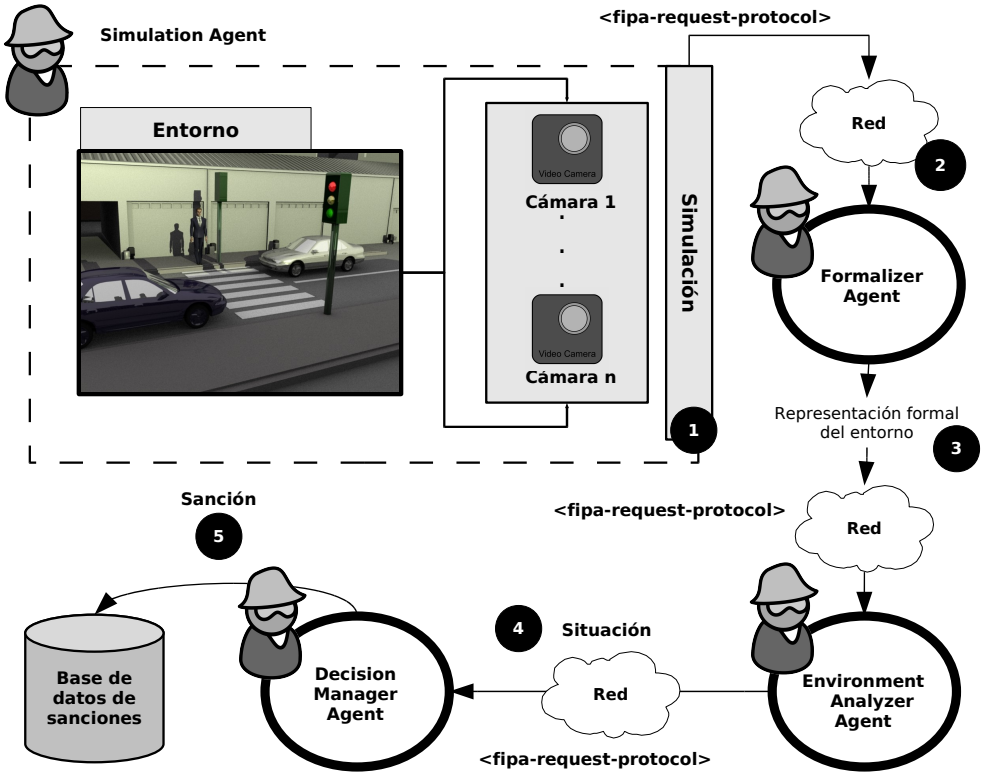


Figura 3.2: Arquitectura abstracta del sistema de vigilancia inteligente para el escenario del paso de peatones.

Concepto	Definiciones
Vehicle	`vehicle(V):-car(V); motorbike(V); truck(V).` `car(V):-height(V, medium), wheels(V, medium).`

Normality without pedestrians	`normal(V, red):-vehicle(V), light(red), \+(is_on(V, crossing)).`
	`normal(V, green):-vehicle(V), (light(green); light(amber)).`
	`normal(V, amber):-normal(V, green).`
	`normal(V, off):-vehicle(V), light(off).`

Normality with pedestrians	`normal(V, G, green):-group(G), normal(V,green), \+(is_on(G, pc)).` `normal(V, G, amber):-normal(V, G, green).`

Tabla 3.2: Algunas de las definiciones de la ontología de tráfico definida con PROLOG.

Infracción	Puntos	Detectado cuando...
Saltarse un semáforo	4	...el semáforo está en rojo y el conductor no para
No ceder el paso a los peatones	4	...los peatones están esperando para cruzar, el semáforo está inactivo y el conductor no cede el paso
Conducción temeraria	6	...los peatones están cruzando y el conductor no para

Tabla 3.3: Ejemplos de multas como respuesta a la detección de infracciones o situaciones anómalas.

3.1.4. Sistema de Comunicaciones

La comunicación entre los distintos agentes que forman el sistema de control de tráfico urbano, tanto a nivel de contenido de mensaje como a nivel de protocolo de interacción, se ha llevado a cabo utilizando los **estándares** definidos por el comité FIPA [Fou02b, Fou00b, Fou00a].

Inicialmente, los agentes se registran tanto en el gestor de la plataforma (*Agent Management System*) como en el servicio de páginas amarillas (*Directory Facilitator*). Cuando un agente necesita interactuar con otro agente, el primero de ellos solicita una búsqueda al servicio de páginas amarillas, siendo éste el responsable de proporcionarle la dirección de contacto (*proxy*) del agente al que se le enviará la solicitud. Por ejemplo, este proceso lo realizará el *Simulator Agent* cuando desee ponerse en contacto con el *Formalizer Agent*. En sucesivas interacciones no será necesario volver a contactar con el servicio de páginas amarillas, ya que tras la primera solicitud el agente almacena de manera interna la dirección del agente con el que desea interactuar.

La interacción entre los agentes que dan soporte a las distintas tareas descritas en la sección 3.1.3 se lleva a cabo mediante el ***FIPA Request Interaction Protocol*** [Fou00b], como se puede apreciar en la figura 3.3. Mediante este protocolo, cada agente realiza una petición (*request*) para que el agente receptor proporcione el servicio al que da soporte, el cual típicamente aceptará con una primitiva especificada en el protocolo de interacción (*agree*). Si el agente receptor aceptó realizar la tarea propuesta, finalizará el protocolo con la notificación de los resultados que obtuvo (*inform*).

3.1.5. Limitaciones

En el contexto de la vigilancia inteligente, la principal limitación de esta propuesta está asociada al coste que habría que asumir en el caso de que o bien sea necesario cambiar el propio sistema de vigilancia desplegado, o bien el entorno monitorizado requiera otro tipo de análisis. El término exacto que define esta cuestión en términos arquitectónicos es el atributo de calidad *modifiability* [BCK03]. Concretamente, uno de las propiedades a las que afecta

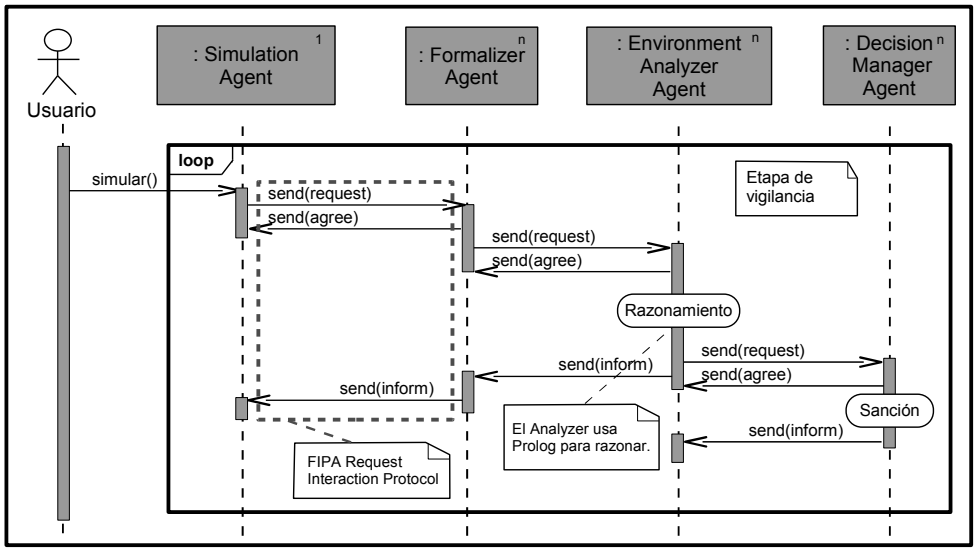


Figura 3.3: Diagrama de interacción que refleja la comunicación entre los agentes del sistema de control de tráfico.

este atributo es a la repercusión respecto a los cambios de funcionalidad del sistema. Por ejemplo, ¿qué ocurriría si el escenario a monitorizar cambia? o ¿la propuesta descrita permitiría simultáneamente el análisis de otra cuestión en el mismo entorno?

En este contexto, la arquitectura discutida anteriormente no es lo suficientemente general para abordar el despliegue de sistemas de vigilancia más sofisticados. Por ejemplo, dicha arquitectura no permite actualmente el análisis de **varias amenazas** o **eventos de interés** dentro del mismo entorno debido a varias cuestiones: i) no existe un mecanismo que permita el despliegue de varios agentes de análisis de normalidad, ii) no es posible la cooperación de dos o más agentes que monitoricen el escenario de tráfico de manera simultánea, y iii) no se contempla la fusión de información como mecanismo para proporcionar una vigilancia más completa.

Respecto a cómo afrontar el problema de analizar un entorno complejo, sería aconsejable seguir un esquema que permitiera la división del mismo en varios sub-entornos. Un sub-entorno se entiende como la visión particular del escenario a monitorizar obtenida mediante un subconjunto de sensores. De este modo, sería posible abordar este problema de una manera más simple y estructurada, basada en monitorizar entornos simples y, como posible alternativa, en fusionar o agregar el conocimiento obtenido de cada sub-entorno para proporcionar una vigilancia global. En la figura 3.4 se plantea la división del escenario de tráfico urbano previamente analizado (ver figura 3.1) en distintas percepciones o sub-entornos.

Figura 3.4: División del escenario de tráfico urbano en varias percepciones. **a)** Vista global. **b)** Sub-entorno E_1. **c)** Sub-entorno E_2.

Por otra parte, el sistema de comunicaciones de la arquitectura planteada sólo soporta la interacción directa entre agentes. Si se tiene en cuenta la posibilidad de que varios agentes monitoricen una misma amenaza o incluso un mismo objeto, entonces sería deseable que la arquitectura soportara mecanismos de comunicación más flexibles, como los basados en eventos o en un sistema de pizarra [EM88].

3.2. Motivación

Actualmente, el diseño y desarrollo de sistemas de vigilancia inteligente supone un reto debido al incremento de la complejidad de los entornos de

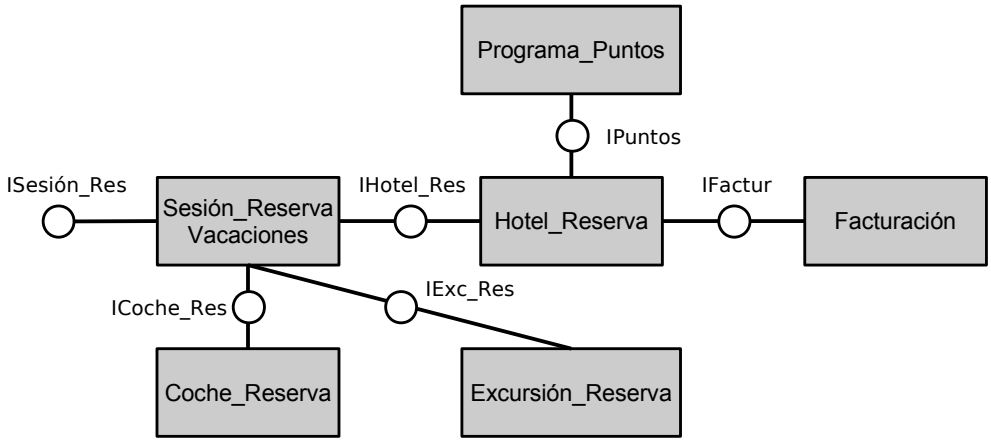


Figura 3.5: Ejemplo de aplicación basada en un modelo de desarrollo basado en componentes (los circulos representan las interfaces).

monitorización y a la necesidad de tratar distintas amenazas o problemas de manera simultánea. Para reducir dicha complejidad y facilitar el diseño y desarrollo de este tipo de sistemas se ha propuesto un **modelo de análisis de normalidad** [AVJL+09] aplicable a entornos susceptibles de ser monitorizados. Este modelo está inspirado en el desarrollo de software basado en componentes, el diseño multi-capa y la división de problemas complejos en sub-problemas que sean más sencillos de tratar.

Desde un punto de vista abstracto, la adopción de un enfoque basado en componentes para diseñar sistemas de vigilancia implica tener en cuenta las siguientes cuestiones: i) identificar los objetos o elementos a monitorizar, ii) determinar los aspectos para monitorizar dichos objetos o elementos, iii) establecer el conjunto de sensores necesarios para recoger la información del entorno, iv) especificar las técnicas requeridas para procesar dicha información (por ejemplo información visual) y v) definir o reutilizar los componentes que permitan la monitorización de los aspectos previamente comentados.

La principal ventaja de adoptar este enfoque basado en componentes reside en que cuando un componente de propósito general se ha diseñado, éste se puede instanciar en tantos entornos de vigilancia como sea necesario (ver esquema abstracto en la figura 3.5). Por ejemplo, si se ha diseñado un componente para definir y analizar trayectorias normales en cualquier entorno, entonces la instanciación de dicho componente implicaría definir las trayectorias normales particulares del entorno monitorizado a través de las definiciones establecidas por el componente. Típicamente, cada componente requerirá un método para obtener el conocimiento necesario para instanciarlo, teniendo en cuenta el problema particular para el cual se está empleando. En este contexto, las herramientas de adquisición de conocimiento o los algoritmos de aprendizaje automático se pueden utilizar para alcanzar dos objetivos principales: i) creación de instancias de componentes en entornos particulares y ii) depuración de la funcionalidad de dichos componentes.

Las características del modelo de análisis de normalidad discutido en [AVJL+09] pueden contribuir a la evolución de los sistemas de vigilancia inteligentes. Esta contribución se debe a la adopción de un enfoque basado en conocimiento experto lo suficientemente general para desarrollar componentes de vigilancia, los cuales se pueden utilizar y desplegar en distintos entornos. A continuación se resumen las principales características de este modelo:

- **Generalidad**. El enfoque adoptado permite modelar y definir cualquier componente de vigilancia de una manera general, es decir, independiente de aspectos o entornos específicos. A la hora de hacer uso de los componentes, éstos han de instanciarse para el entorno concreto de monitorización en función de sus características.
- **Reutilización**. El desarrollo de componentes generales que especifican la normalidad de entornos de manera independiente del dominio de aplicación facilita el reciclaje de conocimiento experto. Esta característica está directamente ligada con la redución del ciclo de desarrollo.
- **Mejora de calidad**. La reutilización de experiencias previas evita la repetición de errores de diseño y agiliza el diseño de nuevos componentes.
- **Fiabilidad**. Si un componente se ha probado exhaustivamente en un entorno particular, el despliegue de dicho componente en otro entorno garantiza una alta probabilidad de éxito en el proceso de vigilancia.
- **Flexibilidad**. Los componentes se pueden utilizar simultáneamente para efectuar distintos tipos de vigilancia dependiendo de las necesidades de monitorización del entorno.

Además de adoptar un enfoque basado en componentes, el modelo de normalidad también se beneficia del **diseño multi-capa** y de la habilidad para afrontar el problema de la vigilancia de una manera estructurada, es decir, dividiendo el problema completo en sub-problemas. Por una parte, el diseño multi-capa proporciona la abstracción necesaria para desarrollar de manera independiente las distintas capas del sistema. De este modo, los cambios en una capa no afectan al resto (siempre que las interfaces de comunicación sean las mismas), lo cual es deseable en sistemas complejos como los sistemas de vigilancia inteligente. Por otra parte, la división en sub-problemas o sub-entornos facilita considerablemente el tratamiento de la complejidad de los actuales entornos de monitorización, ya sean interiores o exteriores. Esta idea hace posible la obtención de resultados parciales que se pueden componer para proporcionar una vigilancia más completa.

Las ventajas de un modelo de vigilancia general implican que la arquitectura que dé soporte a dicho modelo cubra un alto número de necesidades. Así mismo, para hacer uso del modelo en problemas particulares, la arquitectura ha de permitir añadir e instanciar componentes en función de los

requisitos de los aspectos tratados por el sistema de vigilancia. En consecuencia, esta arquitectura ha de afrontar la división del problema global en sub-problemas, lo cual normalmente implica tratar de manera inteligente la información distribuida en el entorno, gestionar las tareas que se desarrollan en cada capa y proporcionar los mecanismos de comunicación necesarios para integrar dichas tareas.

Desde un punto de vista arquitectónico, las propuestas actualmentes existentes no han sido diseñadas para satisfacer estas necesidades generales planteadas por el modelo de normalidad previamente mencionado. En este ámbito, los trabajos más recientes [ERLA06] [VBL+05] [DvdHD+08] [TBH+08] que tienen en cuenta la escalabilidad y flexibilidad de la arquitectura están centrados principalmente en dos cuestiones: la integración de nuevos sensores y la inclusión de módulos de análisis para problemas o entornos particulares. En otras palabras, estos sistemas no están concebidos considerando las ventajas proporcionadas mediante el uso de modelos de conocimiento generales para ofrecer una vigilancia más sofisticada.

Para soportar las necesidades planteadas por modelos de conocimiento aplicados al ámbito de la vigilancia inteligente, como el previamente introducido en [AVJL+09], en esta tesis se propone la arquitectura que permite el despliegue de entidades autónomas que gestionan las tareas de vigilancia de entornos en función de los componentes de análisis de normalidad utilizados para llevar a cabo la monitorización. Así mismo, esta arquitectura ha sido concebida para tratar con el problema de la vigilancia de una manera sencilla teniendo en cuenta la división del entorno monitorizado en sub-entornos, como especifica el modelo de normalidad que inspira su diseño y desarrollo. Para ello, la arquitectura también proporciona los mecanismos de comunicación requeridos para integrar la información generada mediante el análisis de distintos componentes y gestiona los mecanismos de interacción con los dispositivos de vigilancia.

En este contexto, la arquitectura está basada en los Sistemas Multi-Agente [Wei99], estableciendo un *framework* donde estas entidades autónomas o agentes conviven y cooperan llevando a cabo las tareas para las cuales fueron diseñados. El uso de este tipo de sistemas facilita la delegación de tareas, el reparto de responsabilidades, la colaboración y la gestión del conocimiento distribuido; aspectos directamente relacionados con las necesidades del modelo de normalidad adoptado.

Actualmente existen propuestas basadas en el uso de la tecnología de agentes para solucionar problemas particulares en entornos concretos, como las discutidas en [ABC+00], [RSJ04], [CCP09] o [PCP+07]. Sin embargo, estas aproximaciones no son lo suficientemente escalables y flexibles para soportar el modelo general de conocimiento, el cual establece las bases para un sistema de vigilancia independiente de dominio y basado en el análisis de normalidad. Por esta razón, en este trabajo se propone una arquitectura multi-agente en la que los agentes que la componen son los responsables

de tratar con los requisitos establecidos por dichos modelos generales de conocimiento. Las principales ventajas de la arquitectura propuesta son las siguientes:

- **Generalidad**, permitiendo el despliegue de agentes que gestionan el conocimiento derivado del diseño de los componentes de vigilancia.
- **Escalabilidad**, haciendo posible añadir nuevos agentes al sistema para tratar con una tarea de vigilancia que no se consideró inicialmente, como el preprocesamiento de información recogida por un nuevo sensor o la necesidad de incluir un nuevo componente de análisis en el entorno.
- **Diseño multi-capa**, estructurando la funcionalidad del sistema en distintas capas.
- **Enfoque basado en agentes**, facilitando el diseño de sistemas distribuidos complejos donde las distintas entidades autónomas cooperan para alcanzar un objetivo común. En el caso referente a este trabajo, este objetivo es la vigilancia de entornos complejos.
- **Control distribuido**, posibilitando el desarrollo de soluciones robustas y adaptables en lugar de utilizar enfoques centralizados.

En la siguiente sección se resume el modelo de normalidad en el cual se inspira la arquitectura multi-agente propuesta. En dicha sección se justificará la necesidad de los distintos agentes que forman la arquitectura a través de los requisitos funcionales de un caso concreto de modelo general de vigilancia, el cual es general, escalable y flexible.

3.3. Fundamentos teóricos: un modelo de vigilancia basado en el análisis de normalidad

Una de las principales metas de la vigilancia inteligente es la de proporcionar mecanismos para automatizar las diferentes tareas que forman parte de este complejo proceso. En este contexto, una de las líneas de investigación más activas en la actualidad es la interpretación de comportamientos. Para ello, una posible solución consiste en modelar o aprender la normalidad de un entorno con el objetivo de definir las situaciones más comunes que pueden darse en el mismo. Sin embargo, también es posible modelar o aprender el conocimiento necesario para representar situaciones anómalas o amenazas potenciales que puedan producirse en el entorno a monitorizar.

En principio, este segundo enfoque puede resultar adecuado para detectar anomalías puntuales. Sin embargo, carece de escalabilidad a la hora de construir un sistema de vigilancia integral en el que se pretenda detectar

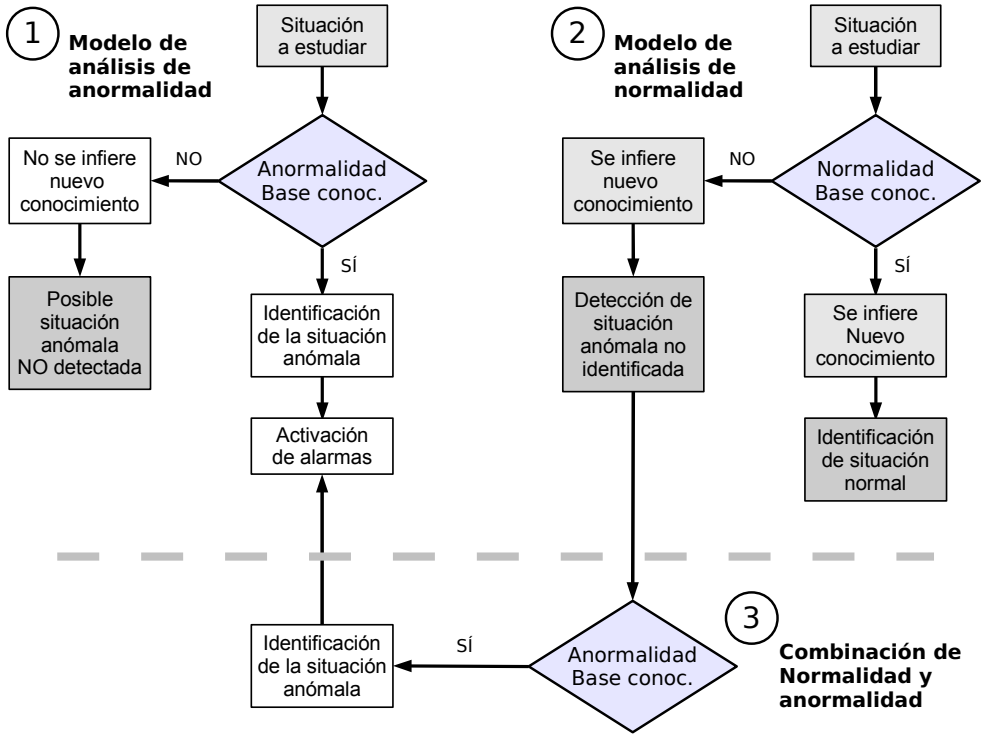


Figura 3.6: Distintos modelos para el análisis y estudio de situaciones en un sistema de vigilancia inteligente: i) modelo basado en el análisis de situaciones normales, ii) modelo basado en el análisis de situaciones anómalas, iii) modelo híbrido que combina ambas aproximaciones.

gran parte de las posibles situaciones anómalas dentro del entorno a monitorizar, ya que existe un elevado número de ellas, muchas incluso impredecibles. Por el contrario, un enfoque basado en la **definición de la normalidad** tiene la ventaja de que ésta se conoce generalmente y está limitada en gran parte de los entornos. De este modo, un sistema de vigilancia basado en este modelo inferiría que una situación detectada y no definida explícitamente en la base de conocimiento es anormal o representa una amenaza. Además, el enfoque basado en la definición de la normalidad se puede complementar fácilmente con la definición de las situaciones anómalas más comunes, con el objetivo de ser capaz de ofrecer respuestas a amenazas particulares y no sólo a cualquier situación no normal. La figura 3.6 compara de manera gráfica estos tres enfoques.

A continuación se describe brevemente un modelo fácilmente acoplable que permite definir e identificar la normalidad en un entorno de acuerdo a diferentes conceptos (estudio de las trayectorias que siguen los objetos, velocidades, relaciones de proximidad, etc), tratando la incertidumbre e imprecisión existente en los datos que proceden de las capas inferiores de la arquitectura.

La arquitectura multi-agente propuesta en este trabajo está inspirada en dicho modelo de análisis de normalidad, estableciendo un marco bien definido para el despliegue de sistemas de vigilancia inteligentes. Este modelo, el cual se discute detalladamente en [AVJL+09], será descrito en esta sección con el objetivo de facilitar la comprensión de la asociación entre modelo y arquitectura.

El **problema de la vigilancia** se puede definir como la interpretación de los distintos sub-escenarios o percepciones (E_i) que componen el entorno global. Estas percepciones se obtienen mediante un conjunto de sensores desplegados en cada una de ellas. De esta forma, el problema de la vigilancia P se puede dividir en una serie de sub-problemas:

$$P = \{E_1, E_2, ..., E_n\} \tag{3.1}$$

La definición y el análisis de normalidad en cada uno de los sub-entornos E_i será más sencilla que si se realizará de manera global sobre un entorno E. En la figura 3.7 se muestra un entorno virtual monitorizado que se divide en cuatro sub-entornos (E_1, E_2, E_3, E_4). Cada uno de estos sub-entornos E_i puede estar vinculado a uno o varios sensores de vigilancia. En el ejemplo de la figura 3.7, cada sub-entorno tiene asociado una cámara de videovigilancia. Esta división en percepciones, a priori independientes entre sí, nos permite abordar el problema de la vigilancia inteligente desde una perspectiva más estructurada, clara y práctica.

A su vez, cada E_i se puede definir mediante una 4-tupla formada por los siguientes elementos:

$$E =< V; O; C; O \times C > \tag{3.2}$$

donde V representa el conjunto de variables de entrada que se utilizan para definir y analizar la normalidad. Por ejemplo, si el sistema quisiera analizar las trayectorias que siguen los objetos, se podrían emplear variables tales como la posición del objeto, tamaño o zona sobre la que se encuentra. Por otro lado, O representa las clases de objetos de interés para el sistema, es decir, aquellos sobre los que el sistema estudia su comportamiento; C engloba el conjunto de conceptos que se utilizan para analizar la normalidad de un entorno y, finalmente, $O \times C$ indica qué conceptos se deben utilizar para estudiar el comportamiento de un tipo concreto de objeto. En otras palabras, cada tipo de objeto tiene asociado la forma en la que se debe estudiar su comportamiento, la cual difiere normalmente con respecto a otros tipos de objetos.

La información asociada a las variables de entrada V se obtiene de los sensores de vigilancia, distinguiéndose dos posibilidades:

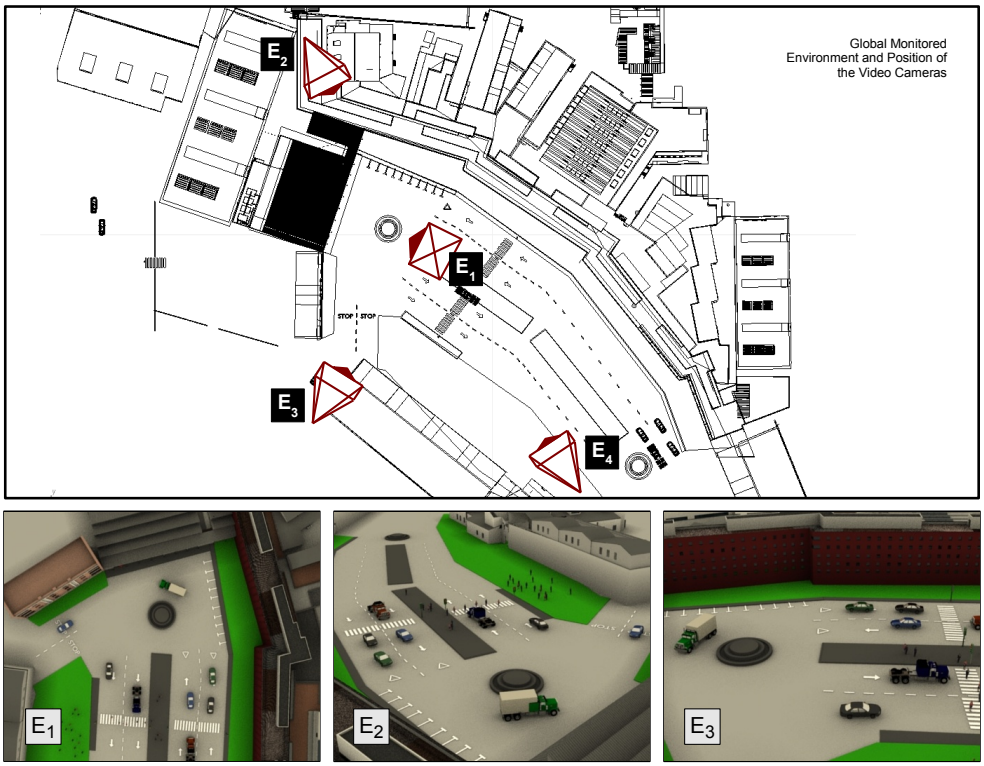


Figura 3.7: División de un entorno virtual de vigilancia en distintos entornos locales (E_1, E_2, E_3 y E_4).

1. que dicha información sea utilizada directamente por el sistema de vigilancia (p.e. sensor de presencia),

2. que sea necesario un procesamiento de la misma para obtener información que otros módulos de más alto nivel puedan utilizar (p.e. cámara de seguridad).

Ante esta segunda necesidad, la arquitectura propuesta permite el despliegue de los ***preprocessing agents***, que son agentes que mantienen un modelo reactivo y aplican una serie de técnicas para procesar la información de uno o varios sensores de vigilancia y generar una determinada salida útil para el sistema de monitorización. En el caso de la videovigilancia, los sensores de un determinado sub-entorno serían las cámaras, mientras que el *preprocessing agent* asociado gestionaría técnicas como la segmentación y el *tracking* [FP02] para identificar y seguir objetos móviles, respectivamente. Finalmente, tanto la información generada por los *preprocessing agents* como por los sensores más simples ha de enviarse a módulos de nivel superior para realizar la vigilancia inteligente.

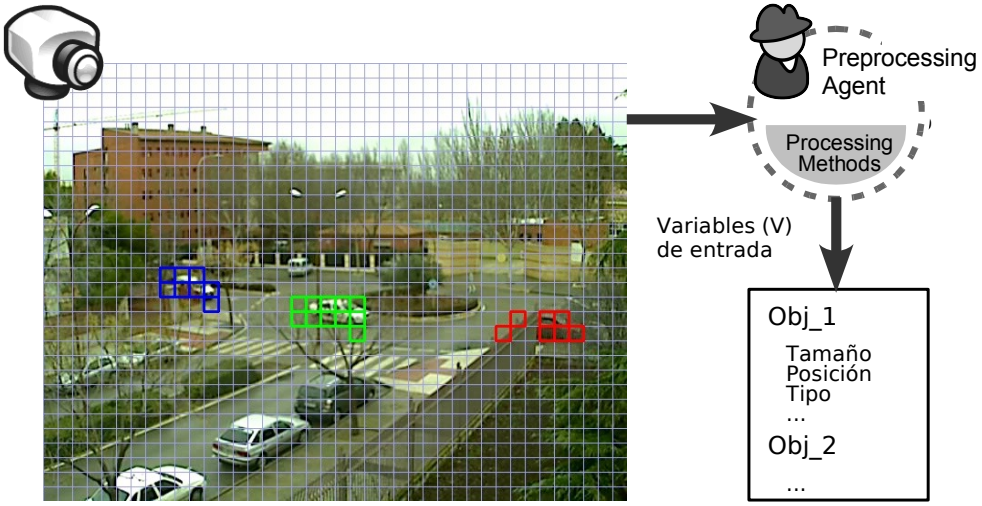


Figura 3.8: Ejemplo de necesidad de procesamiento de la salida de una cámara de vídeo. Las variables (V) de entrada serán las utilizadas en el proceso de análisis de normalidad.

Debido a que el número de sensores, *preprocessing agents* y consumidores de información generada por los dos primeros no se conoce inicialmente y puede variar dinámicamente una vez que el sistema ha sido desplegado, la arquitectura ha de proporcionar un modelo de comunicación flexible y escalable que permita independizar a los consumidores de información (módulos de análisis y monitorización) de los generadores de información (sensores y *preprocessing agents*). Como se describirá en profundidad en la sección 3.4.2, este modelo está basado en **eventos** y canales de eventos. Los primeros reflejan un cambio de estado en el entorno monitorizado mientras que los segundos son canales de comunicación unidireccionales que permiten el envío de eventos de los publicadores a los suscriptores de información.

Una vez que se han definido las variables, los objetos de interés y se han decidido los conceptos que intervendrán en el análisis de normalidad, el siguiente paso es definir cada uno de los conceptos y sus restricciones. Como veremos a continuación, el análisis de normalidad se puede llevar a cabo en base al cumplimiento de un conjunto de restricciones, las cuales determinan la normalidad del entorno de acuerdo a un concepto. Así, un **concepto** se puede definir en base a tres elementos principales:

$$c_i =< V_i; DDV_i; \Phi_i > \tag{3.3}$$

El concepto c_i se define a partir de un conjunto de variables $V_i \subseteq V$, un dominio de definición DDV_i para cada una de estas variables y, finalmente, Φ_i como el conjunto de restricciones o condiciones que se deben satisfacer para que un objeto se comporte de manera normal de acuerdo al concepto

c_i ($\Phi_i = \{\mu_{1i}, \mu_{2i}, ..., \mu_{ki}\}$). A la hora de definir las restricciones es importante tener en cuenta la incertidumbre e imprecisión de los datos que proceden de los niveles inferiores. Un sistema artificial que interactúa con el mundo real no podrá, en muchas ocasiones, afirmar con total certeza si un objeto satisface una restricción totalmente o no. Por esta razón, el modelo debe facilitar los mecanismos necesarios para estudiar el grado en el que se cumplen estas restricciones. De esta forma, una **restricción de normalidad** se define como un conjunto difuso [Zad65] en el dominio $\mathcal{P}(V_i)$ con una función de pertenencia asociada μ_{xi}:

$$\mu_{xi} : \mathcal{P}(V_i) \longrightarrow [0, 1] \tag{3.4}$$

La función devolverá un valor perteneciente al intervalo $[0, 1]$, donde 1 representa el máximo grado de satisfacción de la restricción (máxima pertenencia al conjunto) y 0 el mínimo. Algunos ejemplos de tipos de restricciones son las siguientes (ver [AVJL+09] para una descripción detallada de las restricciones):

- **Restricciones de rol**, las cuales indican qué tipo de objetos pueden realizar una determinada acción.
- **Restricciones espaciales**, las cuales determinan dónde puede situarse el objeto, cuál debe ser su velocidad, qué tipo de movimientos puede realizar, etc.
- **Restricciones temporales**, las cuales establecen cuándo un objeto puede realizar una determina acción.

Una vez que se han definido de manera general los conceptos que se van a emplear en un entorno y el tipo de restricciones para determinar la normalidad, es necesario particularizar estos conceptos y sus restricciones en un entorno concreto (parte del entorno global observado desde uno o varios sensores). Por ejemplo, si se estudia como concepto las trayectorias que siguen los objetos, una particularización de dicho concepto consistiría en definir las trayectorias específicas que se pueden realizar en una región concreta (por ejemplo la entrada a un edificio) del entorno monitorizado por un conjunto de sensores.

De manera formal, una instancia de un concepto c_i en un entorno E_j ($E_j \in P$), denominada c^j_{iy}, se define de la siguiente forma:

$$c^j_{iy} =< V_i; DDV_i; \tilde{\Phi}_i = \{\tilde{\mu}_{1i}, \tilde{\mu}_{2i}, ..., \tilde{\mu}_{zi}\} > \tag{3.5}$$

tal que $\tilde{\Phi}_i$ es un conjunto de restricciones particularizadas de las restricciones definidas de forma general en Φ_i, es decir, $|\Phi_i| \geq |\tilde{\Phi}_i|$. Cada $\tilde{\mu}_{xi} \in \tilde{\Phi}_i$ representa la particularización de $\mu_{xi} \in \Phi_i$.

Una **instancia de una restricción** de normalidad se considera como un conjunto difuso definido en $\mathcal{P}(DDV_i)$, con una función de pertenencia asociada ($\tilde{\mu}_{xi}$):

$$\tilde{\mu}_{xi} : \mathcal{P}(DDV_i) \longrightarrow [0,1] \tag{3.6}$$

tal que si $v_{ki} \in \mathcal{P}(V_i)$ define μ_{xi}, entonces $DDV_{ki} \in \mathcal{P}(DDV_i)$ son utilizados al instanciar $\tilde{\mu}_{xi}$.

Con el objetivo de gestionar el conocimiento de un concepto en un sub-entorno particular, la arquitectura soporta la instanciación de los ***normality agents***. Estos agentes deliberativos mantienen un modelo de conocimiento del sub-entorno que vigilan, siendo capaces de razonar para determinar si un objeto se comporta de manera normal de acuerdo al concepto en el que están especializados. De este modo, se construirá un *normality agent* por cada concepto $c_i \in C$ analizado en cada uno de los sub-entornos E_i en los que se dividió el entorno global. Como se mencionará en la siguiente sección, este conocimiento se puede obtener mediante herramientas de adquisición de conocimiento o se puede aprender mediante técnicas de aprendizaje automático.

Para finalizar la descripción del modelo que estudia la normalidad en entornos monitorizados, es necesario establecer cómo se determina si el comportamiento de un objeto es normal en base a los conceptos y restricciones definidos previamente. La **normalidad de un objeto** obj en un entorno E_j ($E_j \in P$), de acuerdo a una **instancia del concepto** c_i (c_{iy}^j), y denotado como $N_{c_{iy}^j}(obj)$, se calcula a partir de los valores obtenidos de cada $\tilde{\mu}_{xi}$:

$$N_{c_{iy}^j}(obj) = \bigwedge_{x=1}^{|\tilde{\Phi}_i|} \tilde{\mu}_{xi} \tag{3.7}$$

donde $\bigwedge$ es una t-norma, por ejemplo la t-norma del mínimo. De esta forma, es posible determinar si un objeto satisface la normalidad de una instancia concreta de un concepto. Por ejemplo, en el caso de las trayectorias, se podría determinar si un objeto está siguiendo una trayectoria concreta de manera correcta o normal. Para calcular el grado de normalidad de un objeto de acuerdo a un concepto se define la **normalidad de un objeto** obj en un entorno E_j ($E_j \in P$), de acuerdo a un **concepto** c_i, y se denota como $N_{c_i^j}(obj)$. La forma de calcular su valor es la siguiente:

$$N_{c_i^j}(obj) = \bigvee_{y=1}^{w} N_{c_{iy}^j}(obj) \tag{3.8}$$

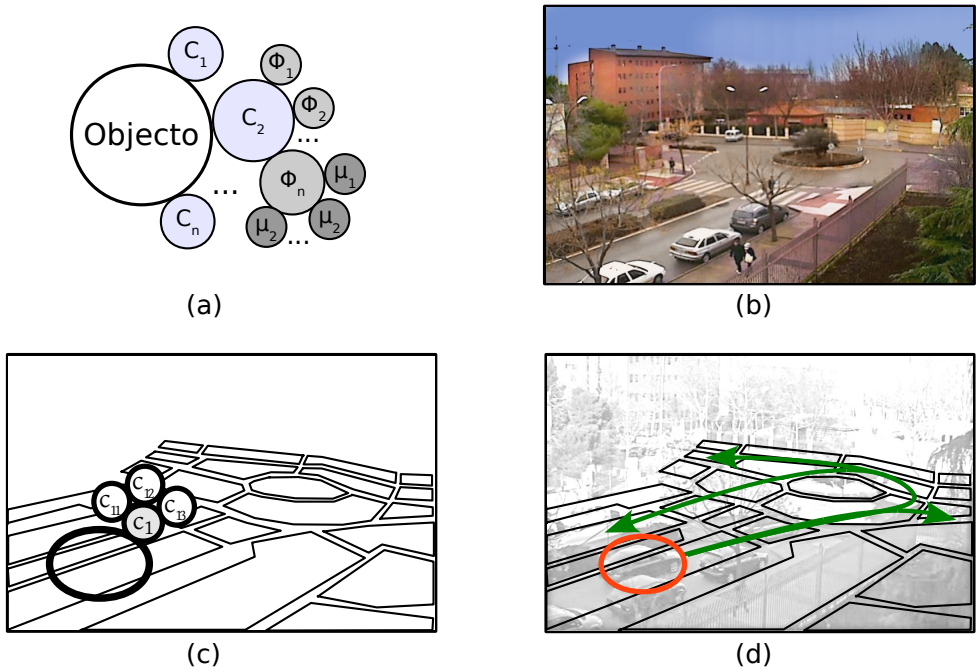


Figura 3.9: Visión gráfica del modelo de análisis de normalidad [AVJL+09].
a) Vista abstracta de *conceptos* (c_i), *restricciones* (Φ_i) y *definición de restricciones* (μ_i).
b) Ejemplo de entorno a monitorizar. **c)** Representación abstracta de la instanciación de un concepto. **d)** Instanciación de distintas trayectorias en el entorno de ejemplo.

donde w es el numero de instancias c_i y $\bigvee$ es una t-conorma, por ejemplo la t-conorma del máximo. Este valor establece la normalidad de un objeto con respecto a un concepto. Si el objeto se comporta de forma normal en este caso, este hecho no implica que se esté comportando normalmente de forma general en el entorno. De hecho, habría que estudiar también cómo es su comportamiento en base a otros conceptos (los especificados en $O \times C$).

La manera más directa de determinar si un objeto se comporta de manera normal de acuerdo a un concepto c_i consiste en establecer un valor umbral α_{c_i} definido por el experto encargado de modelar la normalidad del concepto c_i en el entorno. De este modo, un objeto se comportará de manera normal a c_i siempre y cuando el valor de $N_{c_i^j}(obj)$ supere dicho umbral α_{c_i}:

$$N_{c_i^j}(obj) > \alpha_{c_i} \tag{3.9}$$

Sin embargo, el principal problema de esta definición es que la diferencia entre un comportamiento normal y uno anormal es muy sutil y poco flexible, de manera que el cambio de una situación anómala a una normal, y viceversa, es tremendamente brusco. Por ejemplo, si $\alpha_{c_i} = 0{,}3$, tal y como se puede apreciar de manera gráfica en la figura 3.10.a, un valor de $N_{c_i^j}(obj) = 0{,}295$ implicaría un comportamiento anómalo mientras que un valor de $N_{c_i^j}(obj) = 0{,}305$ implicaría un comportamiento normal.

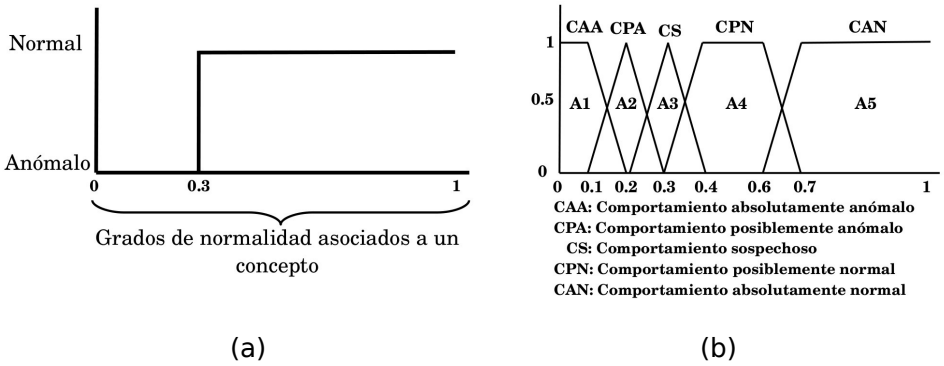


Figura 3.10: a) Función de pertenecia básica para determinar la normalidad del comportamiento de un objeto en base a un concepto. **b)** Definición de la variable difusa *comportamiento* a través de cinco conjuntos difusos que representan distintos grados de normalidad

Para solventar estas limitaciones, el grado de normalidad de un objecto respecto a un concepto c_i se establece mediante una variable lingüística difusa con cinco valores de normalidad (comportamiento absolutamente anómalo, posiblemente anómalo, sospechoso, posiblemente normal y absolutamente normal), tal y como se aprecia en la figura 3.10.b. Esta solución permite una definición más flexible de normalidad, posibilitando la clasificación de una situación monitorizada en uno o más conjuntos difusos.

Es decir, la variable $N_{c_i}^j$ tomará los valores del dominio difuso $DDV_{N_{c_i^j}} = \{A_1, A_2, A_3, A_4, A_5\}$, donde cada A_i representa el conjunto difuso en la posición i, como se aprecia en la figura 3.10.b.

Un objeto se comportará normalmente de acuerdo a un concepto siempre y cuando satisgafa la normalidad de alguna de las instancias del propio concepto. En el caso de las trayectorias, un objeto se comportará normalmente en un determinado sub-entorno E_j siempre y cuando esté siguiendo alguna de las trayectorias definidas como normales.

En la figura 3.11 se muestra gráficamente la evolución de un sistema de vigilancia basado en el modelo de normalidad descrito y su comparación con el razonamiento de un guardia de seguridad. En este caso concreto, se define la normalidad del concepto *trayectorias* en un escenario de tráfico urbano que consiste en una intersección regulada por una glorieta circular. En cada instante de tiempo se muestran las posibles trayectorias asociadas al objeto monitorizado (flechas verdes). Se puede apreciar como en el instante $t = 6$ el vehículo realiza una maniobra que se aleja de las trayectorias normales para ese entorno particular.

Cuando un componente de normalidad analiza el comportamiento de un objeto, genera un valor numérico de salida y se estudia la pertenencia de este

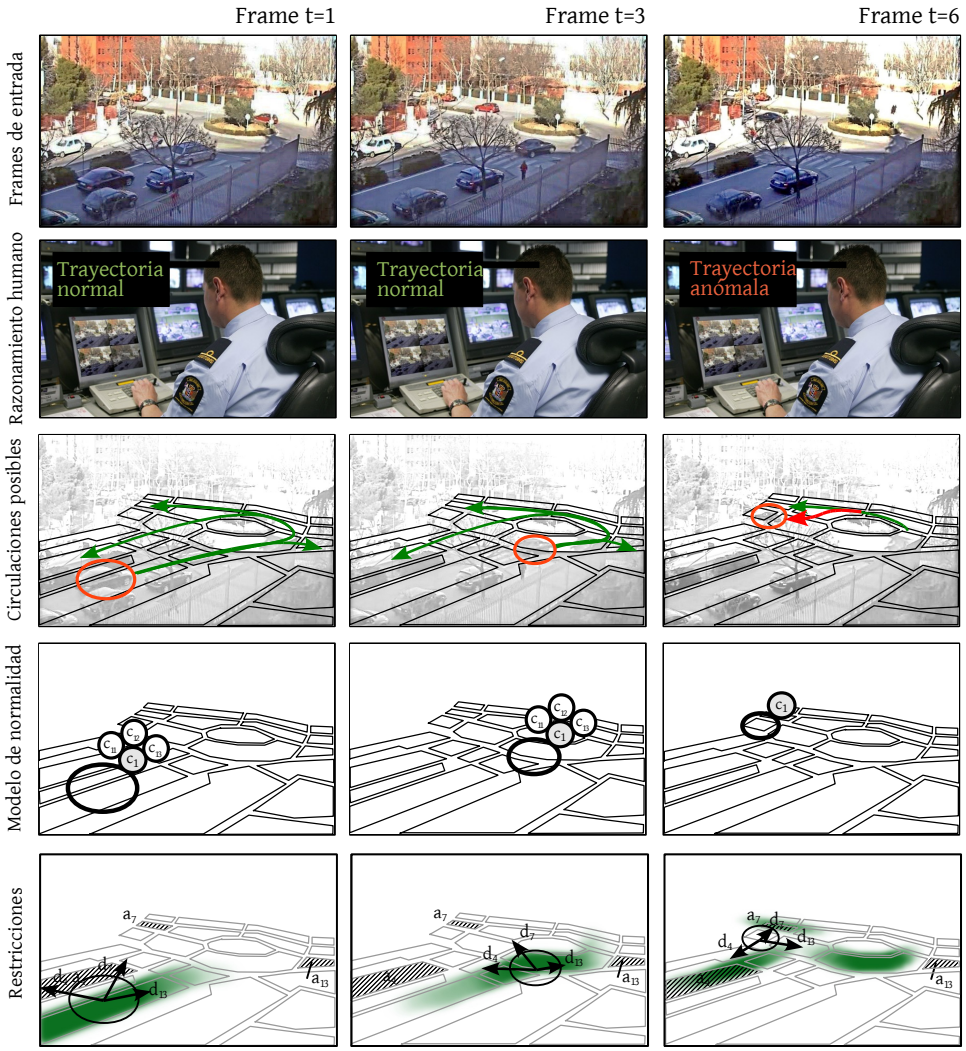


Figura 3.11: Representación gráfica de la evolución temporal del sistema de vigilancia artificial. **Primera fila**: imágenes estáticas obtenidas de la cámara de vídeo utilizada para obtener información visual del entorno. **Segunda fila**: conocimiento inferido por un hipotético guardia de seguridad en cada momento. **Tercera fila:** posibles trayectorias del vehículo monitorizado (las flechas verdes representan trayectorias normales y las rojas representan trayectorias prohibidas). **Cuarta fila:** situación del modelo de normalidad en función del concepto de trayectorias previamente definido por el experto humano. **Quinta fila:** razonamiento visual de las restricciones asociadas a las posibles trayectorias del vehículo monitorizado (las zonas rayadas representan las restricciones de destino, los tonos verde indican el grado de cumplimiento de las restricciones espaciales).

valor a los conjuntos difusos definidos en la Figura 3.10.b. El resultado de dicho estudio permite al sistema de vigilancia ser consciente de la normalidad del comportamiento del objeto con respecto a un determinado concepto. Sin embargo, dicha normalidad no depende únicamente de un concepto, sino de una valoración global de todos los conceptos utilizados para monitorizar un objeto. Por tanto, es necesario establecer una serie de mecanismos que permitan la combinación de los valores de salida de los componentes para obtener un valor global que represente la normalidad del comportamiento de un objeto de manera general.

En este caso concreto, el modelo propone el uso de **operadores de agregación OWA** que pueden comportarse tanto como los operadores t-norma y t-conorma, caracterizándose por su gran flexibilidad en la toma de decisiones a partir de múltiples criterios.

Los operadores de agregación OWA (*ordered weighted averaging*, media ponderada ordenada) fueron propuestos por R. Yager [Yag88] y facilitan la toma de decisiones en base a múltiples criterios con diferentes pesos. Formalmente, un operador de agregación OWA se puede representar mediante una función $F : R^n \rightarrow R$, que tiene asociado un vector de pesos W de longitud n; $W = [w_1, w_2, ..., w_n]$, donde cada $w_i \in [0, 1]$ y $\sum_{i=1}^{n} w_i = 1$.

En el modelo de vigilancia planteado, dichos pesos determinarán la *importancia* de cada concepto dentro del análisis global del comportamiento de un objeto en base a varios conceptos, permitiendo distintas configuraciones entre las que destacan la t-norma mínimo, la t-conorma máximo o una aplicación ponderada de dichos pesos. La elección de los valores de los pesos del vector W dependerá del nivel de vigilancia que se quiera alcanzar.

Una vez que se ha obtenido un valor global de normalidad como resultado de la combinación de los valores de salida de los componentes que han participado en el análisis del comportamiento de un objeto ($OWA(N^j_{c_1}, N^j_{c_2}, ..., N^j_{c_n})$), se estudia la pertenencia de este valor a los conjuntos difusos de la variable *comportamiento* que se muestra en la figura 3.10.b. El valor final de normalidad $N^j(obj_k)$ asociado a un objeto obj_k en un determinado entorno E_j es la suma de los valores de pertenencia del valor de salida del operador $OWA(N^j_{c_1}, N^j_{c_2}, ..., N^j_{c_n})$ a los conjuntos difusos CPN: comportamiento parcialmente normal y CAN: comportamiento absolutamente normal:

$$N^j(obj_k) = \mu_{CPN}(OWA(N^j_{c_1}, N^j_{c_2}, ..., N^j_{c_n})) + \mu_{CAN}(OWA(N^j_{c_1}, N^j_{c_2}, ..., N^j_{c_n})) \tag{3.10}$$

donde $\mu_{CPN}(OWA(N^j_{c_1}, N^j_{c_2}, ..., N^j_{c_n}))$ es la pertenencia del valor de salida del operador OWA al conjunto CPN y $\mu_{CAN}(OWA(N^j_{c_1}, N^j_{c_2}, ..., N^j_{c_n}))$ la pertenencia del mismo valor al conjunto CAN. El valor global de normalidad GN^j en un entorno E_j se calcula como el mínimo de los valores de normalidad N^j calculado para cada objeto obj_k:

$$GN^j = \bigwedge_{k=1}^{|O|} N^j(obj_k) \tag{3.11}$$

donde $|O|$ representa el número de elementos del conjunto O en el que se agrupan los objetos que están siendo analizados por el sistema, y $N^j(obj_k)$ el valor de normalidad calculado para cada objeto obj_k. Si un objeto no se comporta de manera normal, entonces el valor global de normalidad GN^j en el entorno E_j será bajo.

Para llevar a cabo el análisis global de normalidad a nivel de sub-entorno, la arquitectura multi-agente permitirá el despliegue de un ***environmental normality analysis agent*** por cada sub-entorno E_j que forme parte del entorno global, el cual será responsable de comprobar que todo objeto monitorizado obj cumple la normalidad asociada a los conceptos que están siendo monitorizados en E_j.

La justificación de plantear un sistema de vigilancia basado en el análisis de normalidad reside principalmente en que ésta es bien conocida y, por lo tanto, su definición está más acotada que la anormalidad en un entorno particular, como se discutió anteriormente en esta sección. La alternativa de modelar la anormalidad mediante conocimiento experto implicaría definir un alto número de situaciones de manera que si una de ellas ocurre y no fue modelada, entonces el sistema sería incapaz de detectarla. Por el contrario, un sistema basado en el análisis de la normalidad detectaría que algo fuera de lo normal está ocurriendo, aunque no sea capaz de inferir el qué.

Aunque este modelo está basado en la definición de las situaciones normales, suele ser deseable conocer o identificar una situación anómala o peligrosa para poder actuar en consecuencia. Ante esta necesidad, la arquitectura propuesta en este trabajo propone el ***abnormality agent***, el cual es el encargado de gestionar el conocimiento de las situaciones anómalas más comunes que se pueden producir en un escenario vinculadas a un determinado concepto c_i. Este agente es capaz de inferir dichas situaciones empleando tanto su conocimiento como el generado por el *normality agent* que trata la normalidad del mismo concepto. Por ejemplo, si un *normality agent* determina que un vehículo no cumplió la normalidad del concepto *trayectorias* porque violó las restricciones espaciales, el *abnormality agent* asociado podría usar esta información para inferir que la situación anómala concreta fue la invasión de una zona habilitada exclusivamente para peatones.

Por lo tanto, se establece una relación uno a uno entre el *normality agent* y el *abnormality agent* asociados a un concepto c_i en un sub-entorno particular E_j. Esta relación configura la necesidad de un mecanismo de comunicación más directo que el basado en eventos y canales de eventos (ver sección 3.4.2). Por otra parte, la arquitectura ha de proporcionar mecanismos para la búsqueda de agentes por nombre (servicio de páginas blancas) y por funcionalidad (servicio de páginas amarillas), así como para el envío de men-

sajes entre agentes (sistema de transporte de mensajes). Estos tres agentes de gestión han sido y son ampliamente usados en los Sistemas Multi-Agente, existiendo incluso estandarizaciones para los mismos. En este trabajo se ha optado por modelarlos de acuerdo a las guías del comité FIPA [Fou04].

Actualmente, el modelo de normalidad descrito mantiene un modelo de razonamiento local a nivel de sub-entorno E_i, el cual está soportado en la arquitectura mediante el *normality agent*. El siguiente paso consiste en realizar un razonamiento global a nivel de entorno P, siendo necesario un **módulo de fusión** que sea capaz de agregar la información generada por los distintos *normality agents* que llevan a cabo el análisis de un mismo concepto en distintos sub-entornos. Normalmente, dichos sub-entornos están relacionados en términos de proximidad física (ver figura 3.7). Ante esta necesidad, la arquitectura propuesta recoge un módulo de fusión de información que alberga a aquellos agentes encargados de analizar la normalidad de un concepto a nivel de entorno global (***fusion information agents***), a un agente responsable de decidir cuándo fusionar información (***fusion manager agent***) y a un agente (***global fusion agent***) que permite relacionar el conocimiento generado por estos agentes de fusión de información.

Las interacciones existentes en el módulo de fusión de información se pueden abordar mediante un modelo de comunicación basado en una arquitectura de pizarra [EM88], de manera que los *normality agents* publican el conocimiento parcial necesario por los *fusion information agents*, los cuales tienen como objetivo el análisis global de un concepto particular. Así mismo, la arquitectura ha de permitir el despliegue de tantos agentes de fusión de información como sean necesarios, teniendo en cuenta que cada uno de ellos maneja el conocimiento de un concepto c_i desde una perspectiva de análisis global.

Finalmente, y considerando la arquitectura abstracta de un sistema de vigilancia planteada en la figura 1.2, dos de las tareas de alto nivel más relevantes son el soporte a la toma de decisiones y la gestión de crisis. Dichas funciones se pueden concebir mediante módulos de apoyo a la decisión que sugieran determinadas acciones al personal de seguridad responsable, como consecuencia de la detección de una situación anormal o peligrosa. Por ello, en la arquitectura se propone la integración del ***decision support and crisis manager agent*** como agente de apoyo a la toma de decisiones y a la gestión de crisis. Este agente interactuará con todos los agentes encargados del análisis del entorno.

3.3.1. Adquisición del conocimiento del modelo

El modelo de normalidad basado en componentes previamente descrito implica la definición general e independiente de dominio de cómo llevar a cabo la vigilancia de un cierto concepto, sin especificar los detalles asociados en entornos de vigilancia particulares. En este contexto, el modelo sugiere

que esta última tarea, es decir, el proceso de crear instancias de un concepto de normalidad para un entorno concreto, se realice mediante herramientas de adquisición de conocimiento [AVJL+09] o mediante algoritmos de aprendizaje automático [ACSLL+09]. Ambos enfoques comparten el mismo objetivo: instanciar la definición general de un concepto de vigilancia previamente modelado (o aprendido) en entornos de monitorización concretos. Por otra parte, el modelo sugiere que este tipo de herramientas son útiles para depurar el funcionamiento de un componente, permitiendo mejorar su calidad a la hora de reutilizarlo.

Por estas razones, la arquitectura propuesta ha de tener en cuenta la existencia de este tipo de herramientas, las cuales resultan esenciales para crear la base de conocimiento de los agentes que realicen el análisis de la escena y para mejorar su calidad. Estas interacciones definen cómo los agentes obtienen el conocimiento necesario para monitorizar entornos concretos en función de los conceptos utilizados para efectuar la vigilancia inteligente.

El enfoque inspirado por el modelo permite la creación de **componentes autónomos** compuestos por el propio concepto a monitorizar, gestionado por un agente de normalidad, un mecanismo para la adquisición del conocimiento de un determinado concepto que facilite su instanciación en un entorno particular, un mecanismo para depurar dichas instancias teniendo en cuenta la evolución del proceso de vigilancia y, finalmente, un conjunto de reglas para integrar el componente en la arquitectura del sistema de vigilancia. De este modo, es posible concebir un modelo de desarrollo basado en componentes de vigilancia formado por todos los elementos necesarios para su definición, despliegue y depuración en entornos de monitorización complejos. En la figura 3.12 se muestra de manera gráfica este enfoque.

Figura 3.12: Modelo de vigilancia basado en un repositorio de componentes independientes.

3.4. Arquitectura multi-agente para la vigilancia inteligente

Las necesidades planteadas por el modelo de normalidad descrito en la sección anterior se han utilizado como base para diseñar la arquitectura multi-agente para el desarrollo y despliegue de sistemas de vigilancia inteligente. Para ello, es necesario llevar a cabo el diseño de aquellos agentes inteligentes que cubran las necesidades funcionales introducidas por el modelo general de vigilancia de la sección 3.3, teniendo en cuenta su adaptabilidad a otros modelos, y de los mecanismos de comunicación necesarios para proporcionar una vigilancia del entorno a monitorizar.

En la Figura 3.13 se muestra la arquitectura propuesta para un sistema de vigilancia inteligente. Dicha arquitectura está estructurada en **tres niveles**: el nivel reactivo, el nivel deliberativo y el nivel de usuario. Los distintos componentes de cada nivel se comunican entre sí mediante un *middleware* de comunicaciones que permite la interacción de los mismos y la integración de nuevos elementos. A continuación se resumen las principales características de cada nivel:

- El **nivel reactivo** está compuesto por los dispositivos de vigilancia utilizados para monitorizar los distintos sub-escenarios en los que se divide el entorno global de monitorización y por aquellos agentes que mantienen un esquema reactivo.
- El **nivel deliberativo** está compuesto por aquellos agentes que mantienen y utilizan un modelo de conocimiento, realizando el análisis de normalidad y anormalidad del entorno desde distintos puntos de vista (componentes). Así mismo, en este nivel residen aquellos agentes que llevan a cabo la fusión de la información obtenida de las distintas percepciones (sub-entornos) y la toma de decisiones.
- El **nivel de usuario** permite la interacción del personal de seguridad a través de herramientas de monitorización.

Además de los agentes propuestos para solucionar el problema de la vigilancia en un entorno desde un punto de vista global, resulta interesante definir una serie de agentes de gestión que faciliten el control del sistema multi-agente y proporcionen ciertos **servicios básicos**, como por ejemplo un servicio de directorio o páginas amarillas o un servicio de comunicación entre agentes. Ante esta necesidad, la arquitectura propuesta define una serie de agentes de gestión de acuerdo al conjunto de estándares definidos por el comité FIPA [Fou04] para el desarrollo de sistemas multi-agente (ver sección 2.1.4). Dichos agentes son el *Agent Management System (AMS)*, el cual gestiona los eventos que se producen en el sistema multi-agente y controla el estado de cada uno de los agentes, ofreciendo un servicio de páginas blancas; el *Directory Facilitator (DF)*, responsable de proporcionar un servicio de

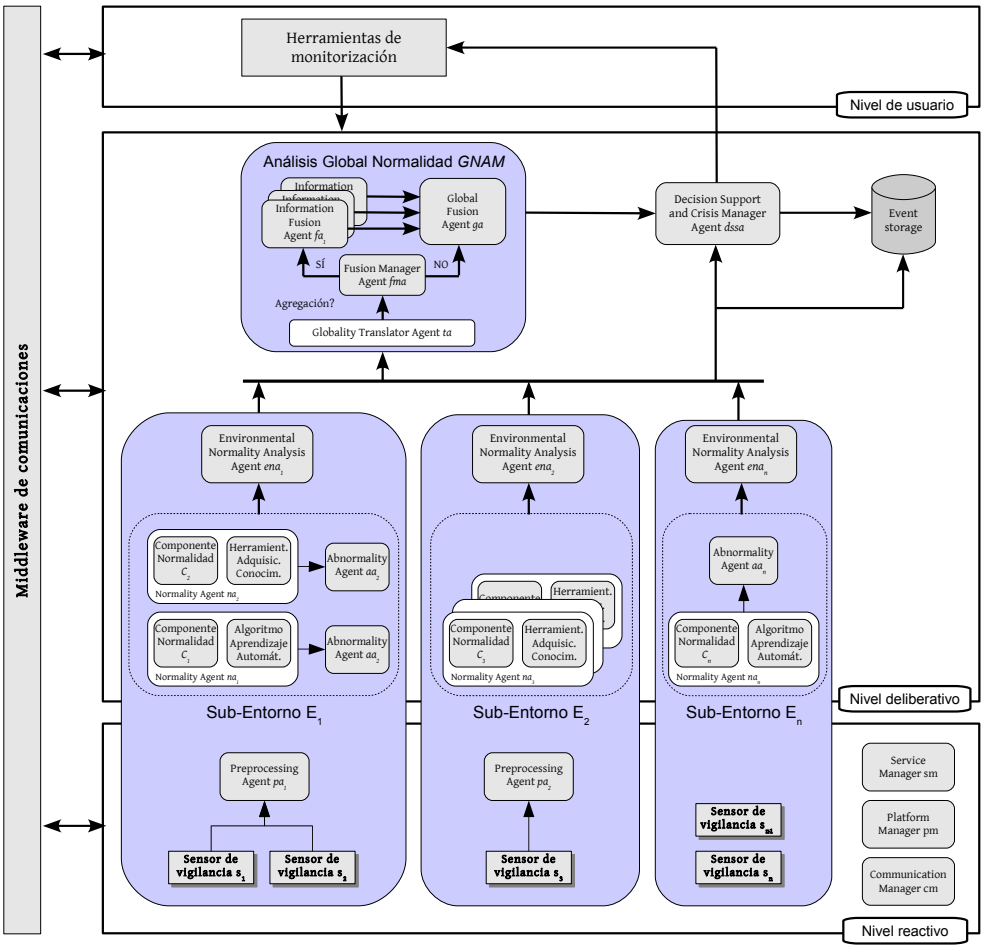


Figura 3.13: Propuesta de arquitectura para sistemas de vigilancia inteligente.

páginas amarillas; y el *Message Transport Service (MTS)*, el cual gestiona el envío y recepción de los mensajes de los agentes.

A continuación se describe cómo fluye la información por los distintos elementos que componen la arquitectura del sistema. En la figura 3.14 aparecen numeradas las principales etapas asociadas a este proceso. El objetivo de esta descripción informal es dar una visión general de cómo la información obtenida por los sensores en el entorno de monitorización se va transformando hasta proporcionar conocimiento de alto nivel, como el análisis de comportamientos o la posible toma de decisiones.

1. **Información obtenida del entorno**. Los sensores de vigilancia son los encargados de obtener dicha información, la cual está vinculada a la especialización del sensor (audio, vídeo, presencia, etc). En el caso de la videovigilancia, esta información consiste en la secuencia de imágenes

estáticas (*frames*) obtenidas por las cámaras de seguridad. La integración de los dispositivos de vigilancia se realizará mediante el *middleware* de comunicaciones que da soporte a la arquitectura del sistema de vigilancia.

2. **Notificación de eventos de vigilancia**. La salida generada por los sensores de vigilancia requiere comúnmente un procesamiento para extraer la información relevante que se utilizará para realizar el análisis del entorno. Si éste es el caso, los *preprocessing agents* procesarán esta información antes de enviarla al nivel deliberativo, como ocurre con la información obtenida de los sensores de vídeo. Sin embargo, también es posible hacer uso de sensores más sencillos cuya información será notificada directamente al nivel deliberativo, como es el caso de los sensores de presencia, los cuales generan normalmente una salida *booleana*. En ambos casos, la información se encapsula en eventos de vigilancia que se enviarán al nivel deliberativo a través de canales de eventos.

3. **Generación de conocimiento para analizar el entorno**. Los eventos de vigilancia enviados desde el nivel reactivo son procesados por los *normality agents*, agentes responsables de analizar la normalidad del entorno monitorizado y capaces de detectar la violación de la normalidad en base a la definición de la misma respecto a múltiples eventos de interés. Todo este conocimiento se comparte gracias a una pizarra en la que los agentes de normalidad van anotando lo que ocurre en el entorno monitorizado. Así mismo, este conocimiento se notifica a los *abnormality agents*, los cuales llevan a cabo la detección de las situaciones anómalas más comunes en los sub-escenarios en los que estén desplegados, y al personal de seguridad a través de las herramientas de monitorización. El conocimiento anotado en la pizarra por parte de los *normality agents* de un escenario es usado por el *environmental normality analysis agent* de dicho escenario para realizar un análisis de normalidad completo a nivel de escenario o sub-entorno, es decir, contemplando la normalidad de todos los aspectos monitorizados en dicho escenario.

4. **Fusión de información**. El conocimiento explícito de la pizarra es utilizado por los distintos *fusion information agents*, cada uno de ellos desplegado por cada aspecto a vigilar en el entorno global, para efectuar un análisis global de normalidad a partir de los resultados parciales obtenidos por los *normality agents*. Con el objetivo de establecer relaciones entre la información obtenida a nivel de escenarios y de entorno global, el *global translator agent* será el encargado de este proceso, ofreciendo una visión homogénea de la información del entorno a los *fusion information agents*. Finalmente, el *global fusion agent* será el responsable de agregar el conocimiento de los distintos agentes de fusión, con la finalidad de proporcionar una vigilancia más sofisticada que relacione distintos conceptos de vigilancia entre sí.

5. **Apoyo a la toma de decisiones**. Tanto el conocimiento anotado en la pizarra como el generado por los *fusion agents* es utilizado por el *de-*

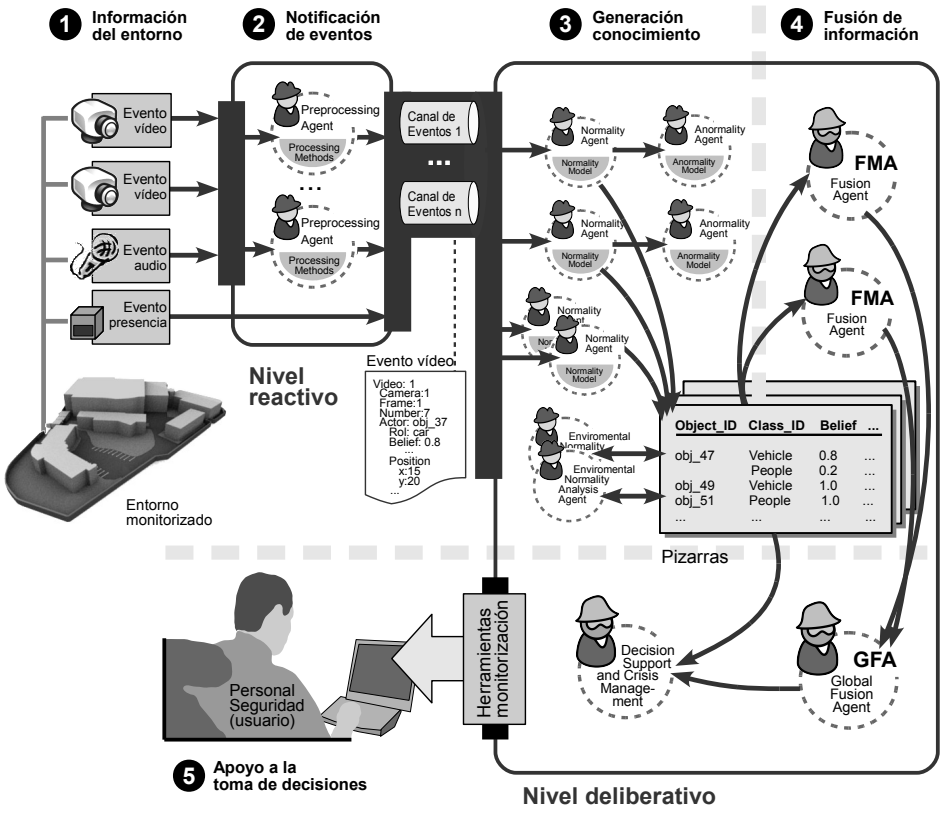


Figura 3.14: Visión general del flujo de información entre los principales componentes de la arquitectura de vigilancia.

cision support and crisis manager agent para ofrecer apoyo al personal de seguridad en lo que a toma de decisiones y gestión de crisis se refiere. La interacción con los mismos se efectúa mediante herramientas de monitorización.

3.4.1. Formalización

A continuación se presenta la formalización de la arquitectura multi-agente propuesta para el problema de la vigilancia inteligente, la cual se puede definir como una tupla:

$$S' =< RL, DL, UL >,$$

donde RL representa el nivel reactivo, DL el nivel deliberativo y UL el nivel de usuario. A continuación se definirán cada uno de estos niveles.

RL representa el **nivel reactivo** de la arquitectura y se define como una tupla $RL =< S, PA, MA >$ (ver figura 3.15), donde:

- S es el conjunto de sensores desplegados en el entorno a monitorizar, $S = \{s_1, s_2, \ldots, s_n\}$.
- PA es un conjunto de agentes encargados de preprocesar las señales generadas por los sensores, $PA = \{pa_1, pa_2, \ldots, pa_m\}$, denominado a partir de ahora como el conjunto de agentes de preprocesamiento (*preprocessing agents*). Cada agente de preprocesamiento $pa_i \in PA$ se define como $pa_i =< S_i, M_i, KB_i, O_i >$, donde:
  - S_i es el conjunto de sensores asociados al agente (verificándose que $S_i \subseteq S$),
  - M_i es el conjunto de técnicas de preprocesamiento utilizadas por el agente ($M_i = \{m_1, m_2, \ldots, m_k\}$, cada $m_j \in M_i$ representa una técnica de preprocesamiento).
  - KB_i es la base de conocimiento del agente que establece cómo emplear las técnicas presentes en M_i para obtener O_i a partir de las señales proporcionadas por S_i.
  - O_i es la salida del agente una vez procesada la información propocionada por S_i empleando las técnicas presentes en M_i.
- MA es el conjunto de agentes necesarios para llevar a cabo la gestión del sistema multi-agente, $MA = \{pm, sm, cm\}$, donde:
  - pm es el agente encargado del registro de todos los agentes y de proporcionar un servicio de páginas blancas al mismo (representado por el *Agent Management System*, según FIPA).
  - sm es el agente gestor de servicios del sistema. Su principal misión es la de proporcionar un servicio de páginas amarillas al resto de agentes (representado por el *Directory Facilitator*, según FIPA).
  - cm es el agente gestor de comunicaciones, el cual actuará como intermediario en el proceso de envío y recepción de mensajes por parte del resto de agentes del sistema (representado por el *Message Transport System*, según FIPA).

DL representa el **nivel deliberativo** del sistema y se define mediante la tupla $DL =< NAA, AAA, ENAA, GNAM, DSM, ES >$ (ver figura 3.16) donde:

- NAA es el conjunto de agentes de análisis de normalidad (*normality agents*) encargados de llevar a cabo la vigilancia del entorno, $NAA = \{na_1, na_2, \ldots, na_k\}$. Cada agente de normalidad na_i se define como:

$$na_i =< IDna_i, c, IA_i, \widetilde{KB}_i, IE_i, QM_i, O_i >$$

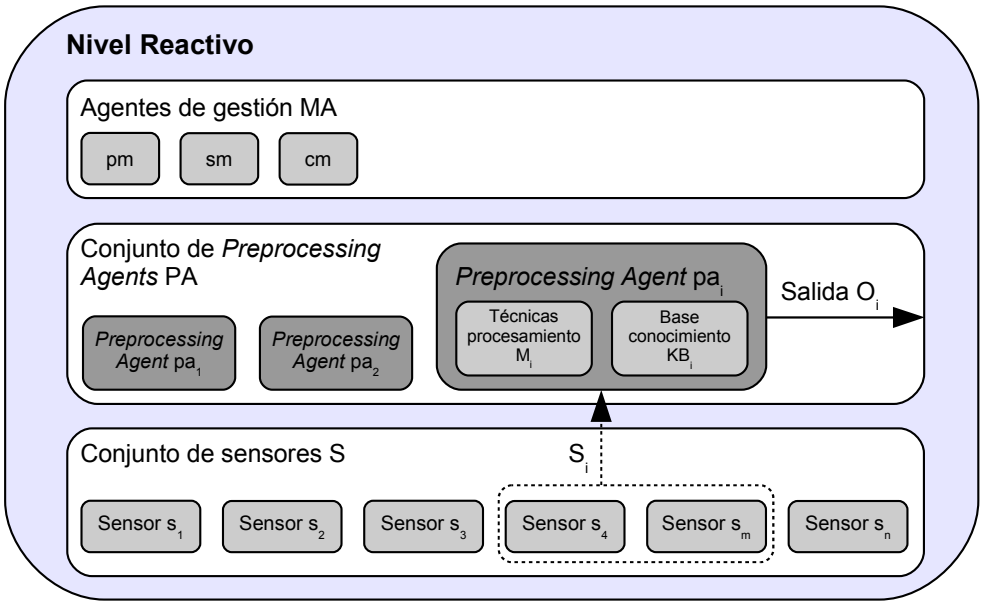


Figura 3.15: Representación gráfica del nivel reactivo.

donde:

- $IDna_i$ es el identificador único del agente en el sistema.
- c es el tipo de análisis o concepto analizado por el agente ($c \in C$ siendo C el conjunto de aspectos o amenazas a vigilar en el entorno).
- IA_i es el conjunto de agentes de preprocesamiento o sensores que le proporciona información al agente para realizar el análisis, $IA_i \subseteq PA \cup S$.
- $\widetilde{KB}_i$ es la base de conocimiento del agente que permite realizar el análisis de normalidad en el entorno.
- IE_i es el motor de inferencia del agente.
- QM_i es el gestor de consultas de agente, el cual permite consultar el progreso del análisis de normalidad del agente.
- O_i es la salida del agente, la cual consiste en una decisión sobre si una situación es normal o no y una explicación de tal decisión.

- AAA es el conjunto de agentes encargados de analizar la anormalidad (*abnormality agents*) más frecuente que ocurre en el entorno, $AAA = \{aa_1, aa_2, \ldots, aa_k\}$. Estos agentes estarán asociados a los agentes de normalidad. Cada agente de anormalidad aa_i se define como:

$$aa_i =< IDaa_i, IDna_j, \widetilde{KB}_i, IE_i, QM_i, O_i >$$

donde:

- $IDaa_i$ es el identificador único del agente en el sistema.
- $IDna_j$ es el identificador del agente de normalidad al que está asociado el agente, utilizando su salida para efectuar el análisis de anormalidad.
- $\widetilde{KB}_i$ es la base de conocimiento del agente que define el conjunto de situaciones anómalas más comunes que tienen lugar en el entorno en base a la salida proporcionada por el agente de normalidad al que está asociado.
- IE_i es el motor de inferencia del agente.
- QM_i es el gestor de consultas de agente, empleado para consultar el progreso del análisis de anormalidad que realiza el agente.
- O_i es la salida del agente, la cual consiste en una decisión sobre si una situación es anormal o no y una explicación de tal decisión.

- $ENAA$ es el conjunto de agentes encargados de analizar la normalidad del entorno (*environmental normality analysis agents*), $ENAA = \{ena_1, ena_2, \ldots, ena_k\}$. Cada agente de análisis de la normalidad del entorno ($ena_i \in ENAA$) estará asociado a un conjunto de agentes de normalidad y anormalidad, y se encargará de combinar las salidas de dichos agentes para estudiar la normalidad del sub-entorno conforme a la información que éstos proporcionan. Cada agente de análisis de la normalidad del entorno ena_i se define como:

$$ena_i =< NA_i, AA_i, \widetilde{KB}_i, IE_i, QM_i, O_i >$$

donde:

- NA_i es el conjunto de agentes de normalidad asociados al agente (se verifica que $NA_i \subseteq NAA$).
- AA_i es el conjunto de agentes de anormalidad conocida asociados al agente (se verifica que $AA_i \subseteq AAA$).
- $\widetilde{KB}_i$ es la base de conocimiento del agente que establece la normalidad del entorno en base a las salidas de los agentes $na_j \in NA_i$ y $aa_j \in AA_i$.
- IE_i es el motor de inferencia del agente que establece cómo combinar las salidas de NA_i y AA_i.
- QM_i es el gestor de consultas de agente, el cual informa de cómo va progresando el análisis de normalidad del entorno que realiza el agente.
- O_i es la salida del agente, la cual consiste en una decisión sobre si una situación en un entorno es normal, anormal conocida o simplemente una situación no conocida y una explicación de tal decisión (que incluye los tipos de los agentes cuya salida se emplea).

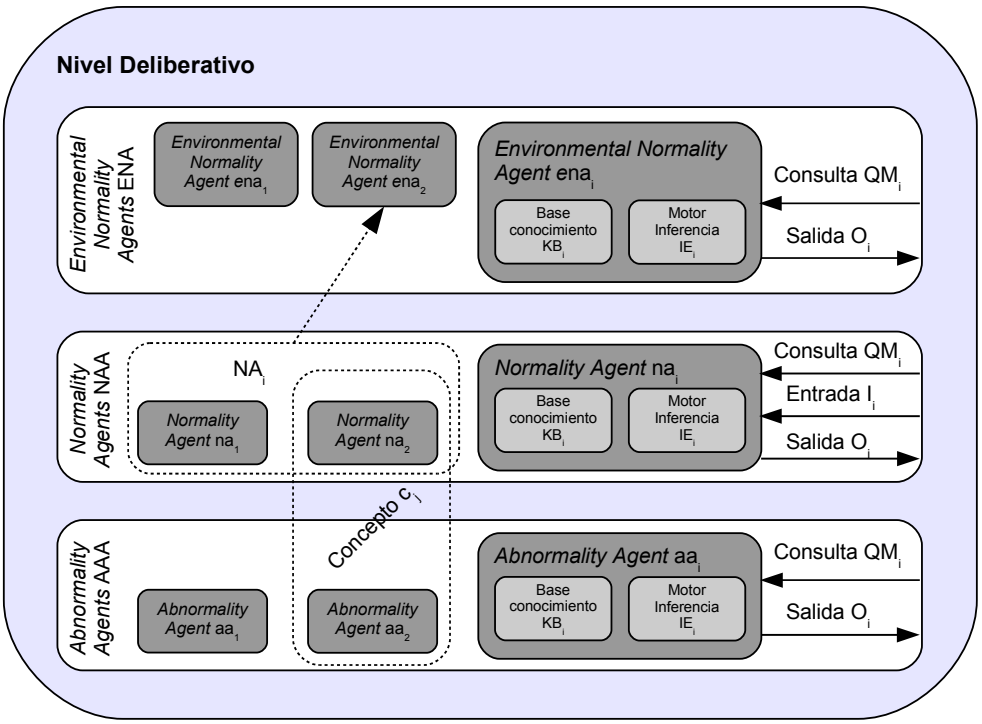


Figura 3.16: Representación gráfica del nivel deliberativo (1).

Cada ena_i permite dividir la complejidad del problema de vigilancia global en una serie de sub-problemas. Por medio de $ENAA$ se crean sub-entornos E_i donde se despliegan una serie de sensores cuyas salidas pueden ser o no preprocesados por agentes de preprocesamiento antes de ser analizados por los agentes de normalidad (y en algunos casos de anormalidad).

- $GNAM$ es el módulo para el análisis de la normalidad global del entorno y tiene como entrada las salidas de los análisis de normalidad de cada uno de los sub-entornos E_i que componen el entorno global, $GNAM =< ta, fma, FA, ga >$, donde:

  - ta (*translator agent*) es el agente encargado de transformar las percepciones locales de cada agente de análisis de la normalidad del sub-entorno en una percepción global.
    Se define como $ta =< ENAA, M_{gt} >$ donde $ENAA$ es el conjunto de agentes encargados de analizar la normalidad de cada sub-entorno. Por otra parte, $M_{gt} = \{m_1, m_2, \ldots, m_x\}$ representa los mecanismos definidos para transformar la información local a cada sub-entorno, empleada por los agentes de normalidad en sus análisis, en información con validez en el entorno global.
  - fma (*fusion manager agent*) es un agente encargado de decidir si es necesario o no agregar el conocimiento que generan los agentes

de normalidad del entorno, qué agentes de normalidad o anormalidad se complementan en base al tipo del análisis realizado y quién será el agente de fusión cuyos servicios serán requeridos. Se define como $fma =< NA', AA', \widetilde{KB}_{fma}, FA >$. La base de conocimiento definirá una función del tipo: $(N|A)A' \times (N|A)A' \times c \longrightarrow FA$.

- FA es el conjunto de agentes encargados de hacer fusión (*fusion information agents*) de la información obtenida por los agentes de normalidad en los niveles inferiores, $FA = \{fa_1, fa_2, \ldots, fa_k\}$. Cada agente fa_i se define como

$$fa_i =< IDxa_i, IDxa_j, c, \widetilde{KB}_i, IE_i, QM_i, O_i >,$$

donde $IDxa_i$ y $IDxa_j$ son los identificadores de los agentes de normalidad o anormalidad cuya información se va a agregar con la finalidad de obtener una visión más precisa del entorno. Se verifica que ambos agentes realizan el mismo tipo de análisis, es decir, éste corresponde al concepto c. $\widetilde{KB}_{fa_i}$ es la base de conocimiento que establece cómo se debe agregar la información. El resto de elementos mantienen las mismas definiciones vistas en anteriores agentes.

- ga (*global fusion agent*) es el agente encargado de agregar la información complementada o no que se genera en los niveles inferiores o por los agentes de fusión, con el objetivo de proporcionar una información global más precisa que permita una vigilancia más sofisticada. Este agente se define como

$$ga =< FA, NA, AA, ENAA, \widetilde{KB}_{ga}, O_{ga} >,$$

donde $\widetilde{KB}_{ga}$ es la base de conocimiento que establece cómo se agrega el conocimiento de distintos agentes de vigilancia y O_{ga} la salida del agente.

- DSM es el módulo para el soporte a la decisión. Su misión consistirá en sugerir posibles acciones al operador de seguridad humano ante la detección de una situación de riesgo. Este módulo estará formado por un único agente $dssa$ (*decision support and crisis manager agent*), definido como ($dssa =< \widetilde{KB}_{dssa}, QM_{dssa}, R >$). En este contexto, $R = \{r_1, r_2, \ldots, r_y\}$ es la salida del agente, consistente en una recomendación sobre un conjunto de acciones a llevar a cabo ante la detección de una situación anómala. $\widetilde{KB}_{dssa}$ es la base de conocimiento que permite relacionar las entradas al agente (salida de ga) con el resultado. QM_{dssa} es el módulo encargado de justificar la decisión recomendada.

- ES es el lugar donde se almacenará la información que se va produciendo en el sistema y que podrá ser empleado para depurar el funcionamiento de los agentes o para realizar un análisis forense de lo que ha ocurrido en el sistema.

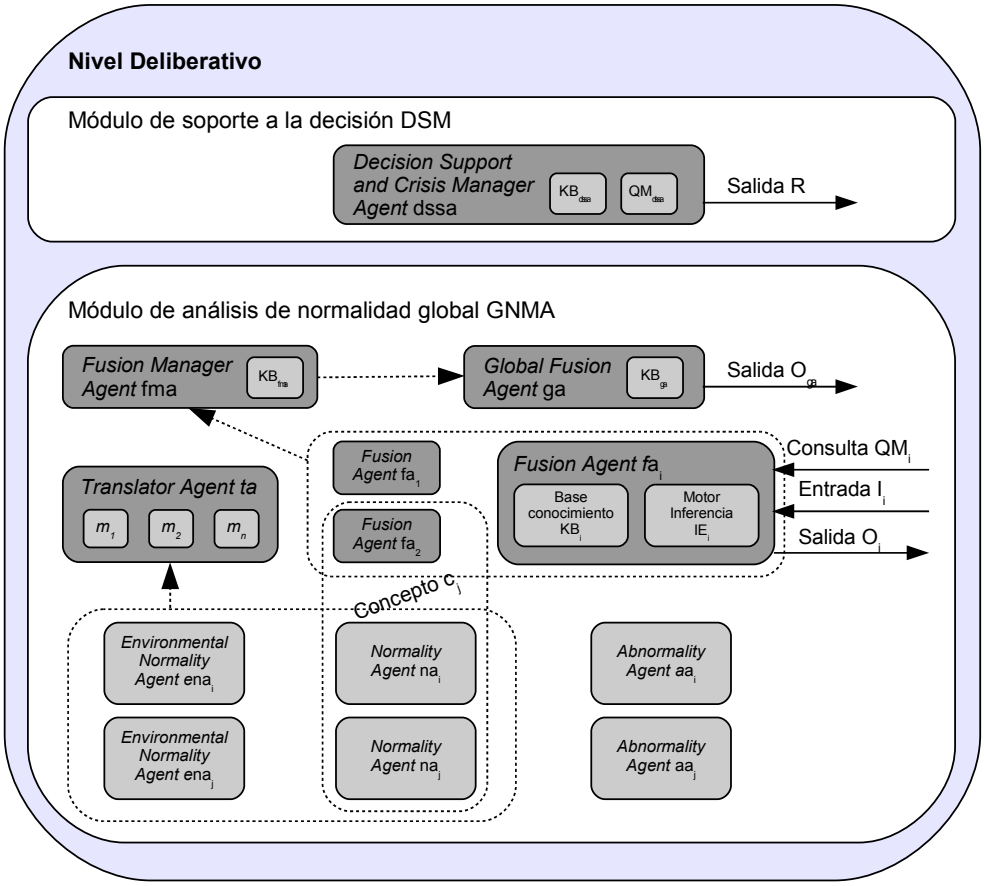


Figura 3.17: Representación gráfica del nivel deliberativo (2).

A modo de ejemplo, y con el objetivo de relacionar la formalización de la arquitectura con el caso particular del modelo presentado en la sección 3, se va a mostrar en qué consisten las bases de conocimiento $\widetilde{KB}_i$ de los agentes de normalidad na_i de tipo c_i. Las bases de conocimiento se definen como $\widetilde{KB}_i =< V_i, DDV_i, \tilde{\phi}_i >$ y representan la instanciación de la base de conocimiento KB_i que analiza en general un concepto c_i y que se define como $KB_i =< V_i, DDV_I, \phi_i >$:

- $V_i = \{v_{1i}, v_{2i}, \ldots, v_{mi}\}$ es el conjunto de variables empleadas para almacenar información del entorno, generalmente proporcionadas por los sensores asociados al agente o construidas por los agentes de preprocesamiento, a partir de la información recogida por los sensores.
- $DDV_i = \{DDV_{1i}, DDV_{2i}, \ldots, DDV_{ki}\}$ es el conjunto de dominios de definición de cada una de las variables $v_{ji} \in V_i$.
- $\phi_i = \{\mu_{1i}, \mu_{2i}, \ldots, \mu_{mi}\}$ es el conjunto de restricciones utilizadas para definir la normalidad del concepto c_i.

de manera que $\widetilde{KB}_i$ será la base de conocimiento instanciada para realizar el análisis de normalidad del concepto c_i en un entorno concreto y para un problema específico, siendo $\tilde{\phi}_i = \{\tilde{\mu}_{1i}, \tilde{\mu}_{2i}, \ldots, \tilde{\mu}_{mi}\}$.

Finalmente, UL representa el **nivel de usuario** y está constituido por un conjunto de herramientas para gestionar y monitorizar tanto los eventos de vigilancia como el conocimiento generado por los agentes de razonamiento.

3.4.2. Mecanismos de comunicación

Como ya se introdujo en la sección 1.1, una de las principales características de los sistemas de vigilancia inteligente de última generación es el incremento considerable de los dispositivos de vigilancia. Esta tendencia implica el tratamiento de tres aspectos fundamentales: i) la necesidad de técnicas de vigilancia automática para reducir la dependencia respecto a operadores humanos, ii) la aplicación de métodos para agregar la información distribuida en el entorno a monitorizar, y iii) el uso de mecanismos de comunicación que se adapten a esta naturaleza distribuida. En esta sección se discutirán las soluciones planteadas como respuesta a esta tercera cuestión.

En la figura 3.13 se planteó de manera gráfica la arquitectura multi-capa diseñada para dar soporte a sistemas de vigilancia inteligentes. Como se puede apreciar, existe un alto número de sensores de vigilancia S de distinta naturaleza (generadores de información) y una serie de agentes encargados de procesar esta información NAA (consumidores de información). Así mismo, la arquitectura se plantea de manera que exista una independencia entre ambos elementos, al mismo tiempo que se posibilita la inclusión de nuevos sensores de vigilancia o agentes de preprocesamiento PA. Por lo tanto, la primera necesidad del núcleo de comunicaciones del sistema de vigilancia ha de ser la **escalabilidad** del mismo, es decir, la adopción de mecanismos de comunicación que garanticen la extensibilidad del sistema de vigilancia.

Otro requisito funcional a tener en cuenta en el modelo de comunicaciones está directamente relacionado con la diversidad de sensores de vigilancia existentes en la actualidad. Aunque los más comunes son las cámaras de seguridad, la inclusión de otro tipo de sensores, como los micrófonos o los detectores de presencia, complementan la vigilancia y permiten partir de una información más rica para llevar a cabo la monitorización de un entorno. En este contexto, sería deseable poder clasificar la información dependiendo del tipo de sensor que la obtuvo del entorno. Para ello, el *framework* de comunicaciones ha de proporcionar los mecanismos necesarios para desplegar canales de comunicación en función de los tipos de sensores utilizados. De este modo, se puede plantear un esquema basado en canales de comunicación en los que se identifican dos roles principales:

1. **Publicador**, es decir, aquel componente del sistema que notifica información siguiendo unos determinados criterios.
2. **Suscriptor**, es decir, aquel componente del sistema que recibe información de uno o más publicadores.

Esta aproximación permite clasificar los elementos del sistema de vigilancia y garantiza la escalabilidad debido a que es posible desplegar los canales de comunicación necesarios y asociarlos tanto a los publicadores como a los suscriptores de información.

Comunicación mediante eventos

En esta sección de discute el modelo de comunicación general de la arquitectura de vigilancia inteligente planteada en este trabajo. Dicho modelo responde a las necesidades funcionales planteadas en la sección anterior, con el objetivo de garantizar la escalabilidad del sistema respecto al número de generadores y consumidores de información. Este esquema es indepen-dinte del nivel funcional en el que se encuentren (reactivo, deliberativo o de usuario). A continuación se exponen los principales conceptos asociados a dicho modelo.

Un **evento** se define como la unidad de información básica de comunicación del sistema y refleja un cambio de estado en uno de los elementos que forman parte del sistema de vigilancia. Un ejemplo podría ser un evento de vídeo, el cual recoge la información visual del entorno en un determinado instante de tiempo (ver figura 3.18.a). Otro evento podría resumir el conocimiento generado por un agente de análisis de normalidad después de procesar la información de un evento de vídeo.

Un **canal de eventos** se define como el mecanismo de comunicación que posibilita el envío de eventos entre publicadores y suscriptores de información. Por lo tanto, el canal de eventos proporciona la funcionalidad necesaria para i) asociar publicadores de información, ii) suscribir consumidores de información y iii) permitir que los publicadores envíen eventos asíncronos de manera que los suscriptores los reciban de manera transparente (ver figura 3.18.b). Así mismo, el canal de eventos está tipado, es decir, tiene asociada una propiedad que determina el tipo de eventos que fluyen a través de él. Un ejemplo podría ser el canal de eventos de audio por el que fluiría la información recogida por los sensores de sonido y a la que accederían aquellos agentes del nivel deliberativo responsables de analizar la normalidad de un concepto utilizando dicha información.

Un **enlace** se define como el puente entre dos canales de eventos y permite que los eventos que llegan a un canal de eventos se reenvíen a otro canal, con el que previamente se estableció dicho enlace. La arquitectura también posibilita la jerarquización de canales de eventos y la asociación de costes

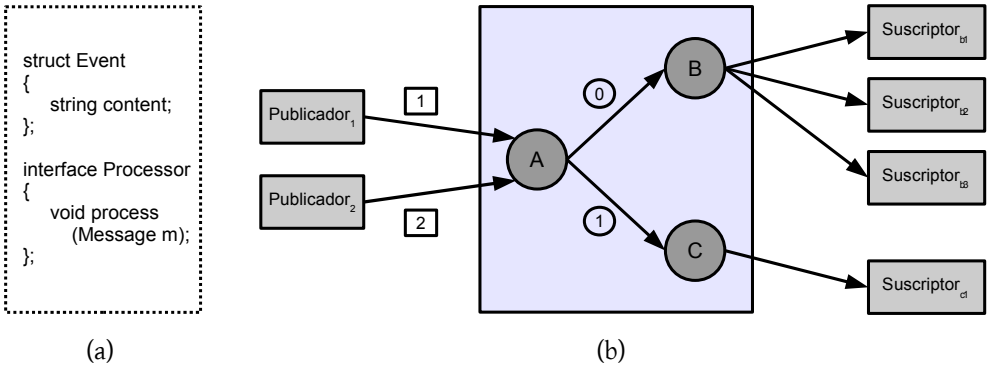


Figura 3.18: Arquitectura abstracta del modelo de comunicación basado en eventos. **(a)** Especificación de un evento y consumidor de información en un lenguaje de descripción abstracto. **(b)** Esquema de comunicación basado en el paradigma publicador-suscriptor. A, B y C son tres canales de eventos.

a los eventos e incluso a los propios canales. Este mecanismo posibilita el reenvío de eventos en función de determinados criterios, como por ejemplo la prioridad o urgencia de los mismos.

Comunicación entre agentes

En determinadas situaciones puede ser necesario establecer una conversación entre dos agentes, como por ejemplo en el caso de dos agentes que analizan las trayectorias de objetos en un mismo sub-escenario pero con distintas perspectivas (posiblemente distintas cámaras de vigilancia). En esta situación, los dos agentes podrían llevar a cabo un proceso de negociación para intentar convencerse el uno al otro sobre qué información tiene asociada un mayor grado de certeza o veracidad.

Una posible solución a esta necesidad funcional es el despliegue de un canal de eventos tal y como se describió en la sección anterior, donde ambos agentes serían publicadores y suscriptores de información de manera simultánea. Sin embargo, es más práctico, eficiente y flexible utilizar un mecanismo que permita la comunicación directa entre agentes en determinadas situaciones.

La solución adoptada en la arquitectura propuesta en este trabajo consiste en utilizar un servicio de consulta tanto por nombre de agente como por la funcionalidad que proporciona para establecer una comunicación entre agentes. Tradicionalmente, estos servicios se han denominado servicio de páginas blancas y servicio de páginas amarillas, respectivamente. Como se discutió en la sección 3.4, los agentes de gestión del sistema de vigilancia multi-agente se han definido de acuerdo al conjunto de estándares definidos por el comité FIPA, por lo que es posible reutilizarlos para proporcionar la funcionalidad previamente descrita.

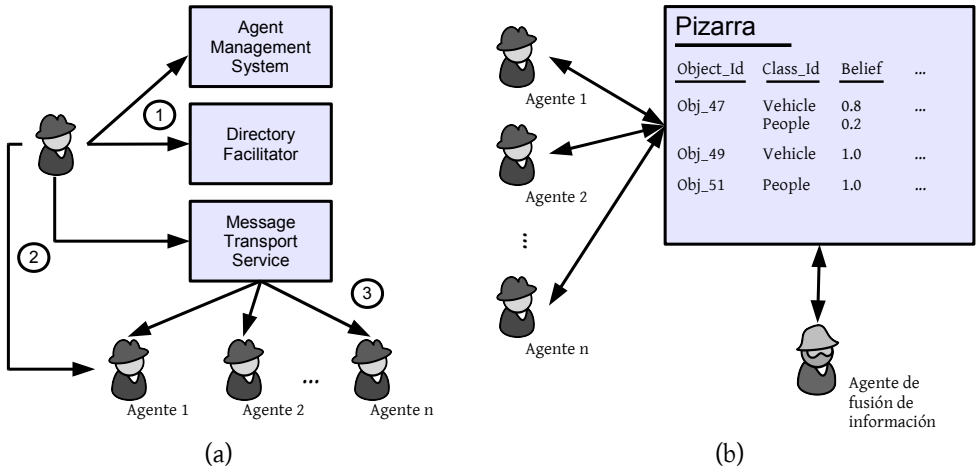


Figura 3.19: (a) Comunicación entre agentes utilizando los mecanismos propuestos por FIPA. **(b)** Comunicación entre agentes utilizando una arquitectura de pizarra.

De este modo, un agente puede establecer una comunicación con uno o varios agentes a través del MTS, indicando el identificador único del agente o agentes destinatarios, o bien recuperando la dirección física de los mismos previa consulta al AMS o al DF (ver figura 3.19.a). Para ello, todos los agentes han de registrarse tanto en el AMS como en el DF.

Arquitectura de pizarra

Un modelo de arquitectura de pizarra [EM88] consiste en hacer uso de una base de conocimiento común, permitiendo que distintos agentes especializados colaboren entre sí y actualicen el contenido de la pizarra para alcanzar una solución global a un problema. En el ámbito de la vigilancia inteligente, este tipo de mecanismos pueden contribuir a solucionar ciertos problemas que requieren la colaboración de distintas fuentes de conocimiento que generalmente están distribuidas en el entorno.

Por este motivo, la arquitectura planteada en este trabajo da soporte al modelo de cooperación basado en la pizarra (ver figura 3.19.b). La aplicación directa de este mecanismo es su uso por parte de los agentes de normalidad NAA, posibilitando que dos agentes que estén analizando, por ejemplo, distintos conceptos en un mismo escenario, compartan la información de un mismo objeto para generar conclusiones con la mayor certidumbre posible.

La funcionalidad proporcionada por la pizarra consiste en leer y escribir de la misma, siendo responsabilidad de los agentes la composición de la información que en ella se encuentra. Por ejemplo, un agente de análisis de normalidad a nivel de sub-entorno ena_i será el responsable de analizar la normalidad de un sub-entorno E_j en base a la información escrita en la pizarra en un determinado instante.

3.4.3. Orientación a servicios

En la sección 2.3 se analizaron las arquitecturas orientadas a servicios como mecanismo de diseño de componentes autosuficientes y abiertos (servicios) a la hora de desarrollar aplicaciones distribuidas. En esta sección se relacionarán las principales características de este tipo de arquitecturas con la propia arquitectura de vigilancia desarrollada en este trabajo. Para ello, la presente sección justifica cómo la propuesta de arquitectura soporta los principios arquitectónicos más relevantes de la computación orientada a servicios. Por otra parte, también se discuten las soluciones de diseño para dar soporte a los servicios más relevantes existentes en este tipo de enfoques.

Principios arquitectónicos

Según se introdujo en la sección 2.3.2, los principios arquitectónicos fundamentales que debería seguir una arquitectura orientada a servicios son i) mínimo acoplamiento, ii) neutralidad de implementación, iii) configuración flexible, iv) persistencia, v) granularidad y, por último, vi) comunidades [HS05]. A continuación se justificará cómo la arquitectura de vigilancia hace gala de estas características.

El **mínimo acoplamiento** se ha alcanzado mediante el soporte basado en agentes, donde éstos gestionan de manera autónoma los servicios que ofrecen al resto de componentes de la arquitectura. De hecho, los agentes sólo necesitan ser conscientes de su contexto, es decir, de tener algún mecanismo para averiguar qué agentes están desplegados y qué servicios ofrecen. Este servicio está representado en la arquitectura por el agente reactivo *service manager* (sm), diseñado de acuerdo al componente *Directory Facilitator* definido por el comité FIPA.

La **neutralidad en la implementación** se garantiza mediante el uso de un lenguaje de especificación de interfaces que sea independiente de cualquier tecnología. En el prototipo implementado de la arquitectura propuesta, esta característica se obtiene mediante el uso de *Slice*, el lenguaje de especificación de ZeroC ICE, *middleware* utilizado para dar soporte a la arquitectura de vigilancia. En la sección 5.2 se profundiza en el proceso de utilización de tecnologías específicas y su relación con *Slice*.

Respecto a la **configuración flexible** de la arquitectura, hay que distinguir dos aspectos. Por una parte, la base de conocimiento de los agentes puede cambiar sin necesidad de alterar el servicio que ofrecen. Por ejemplo, un agente de análisis de normalidad puede refinar su conocimiento en un determinado momento gracias a una herramienta de depuración sin que se produzcan modificaciones en las relaciones con otros agentes. Por otra parte, los servicios proporcionados por la arquitectura (delegados en agentes) se pueden activar bajo demanda en tiempo de ejecución, flexibilizando su despliegue y proporcionando una gestión eficiente de los recursos utilizados.

La **persistencia** de servicios se traslada, principalmente, a la posibilidad de almacenar la base de conocimiento de los agentes en memoria permanente. Debido a que la arquitectura está inspirada en el modelo de conceptos de análisis de normalidad, también es posible plantear un repositorio de ontologías que los agentes utilizan para recuperar el conocimiento necesario para analizar una escena.

La **granularidad** de los servicios proporcionados por los agentes se refleja directamente en agentes esenciales en la arquitectura, como por ejemplo el agente de normalidad. Dicho agente mantiene una interfaz sencilla de grano grueso basada en operaciones de consulta, notificación de eventos de vigilancia y destrucción. En la sección B.1.1 se muestra la interfaz de dicho agente.

Finalmente, el principio de **comunidades** se alcanza mediante la propia naturaleza multi-agente de la arquitectura y el uso de modelos de comunicación como el sistema de pizarra, fomentando el control distribuido y el desarrollo de soluciones basadas en la comunicación y la cooperación.

Servicios relevantes

Dos de los servicios relevantes dentro de la Computación Orientada a Servicios son la **localización** y la **publicación**. En la arquitectura propuesta, el primero de ellos está representado por los agentes *platform manager* (pm) y *service manager* (sm), diseñados de acuerdo a los componentes *Agent Management System* y *Directory Facilitator* de FIPA, respectivamente.

El segundo de estos servicios está soportado por los canales de eventos proporcionados por la arquitectura, los cuales se han desplegado para alcanzar objetivos específicos en el problema particular de la vigilancia inteligente. Un ejemplo es el despliegue de canales de eventos para enviar la información del escenario monitorizado desde el nivel reactivo (sensores y *preprocessing agents*) al nivel deliberativo (*normality agents*).

Finalmente, otro servicio muy interesante es la **composición** de servicios. En este caso, la arquitectura aborda este problema mediante las interacciones entre los agentes que proporcionan los servicios de vigilancia, ya sea mediante comunicación directa entre agentes o a través de sistemas de pizarra donde los agentes comparten el conocimiento que van generando. Esta solución de modelado se puede interpretar como una aproximación a la composición de servicios.

3.4.4. Requisitos sistemáticos

Además de las características propias de la arquitectura propuesta para la vigilancia inteligente, es decir, de las denominadas características o propiedades funcionales, existe una serie de características o requisitos sistemáticos que una arquitectura de esta naturaleza debería soportar. El objetivo que se pretende alcanzar con este tipo de requisitos es facilitar el despliegue práctico de sistemas de vigilancia inteligente. En esta sección se discutirá cómo la arquitectura propuesta aborda la adquisición de estos requisitos. Para ello, se tomará como referencia principal el artículo de H. Detmold et al. [DvdHD+08], donde los autores presentan un *middleware* para la videovigilancia y enumeran una serie de requisitos sistemáticos a satisfacer. En la sección actual, estos requisitos se extrapolarán a la vigilancia con distintos tipos de sensores. La justificación de esta elección reside en la similaridad de ambas propuestas en cuanto a objetivos finales (esencialmente dar soporte a sistemas de vigilancia inteligente) y en la experiencia de dichos autores[2] en este área.

Estos requisitos se describen brevemente en el siguiente listado[3]:

- *Escalabilidad* (*scalability*), como la capacidad y los mecanismos de la arquitectura a la hora de integrar nuevos componentes.
- *Disponibilidad* (*availability*), relacionada con la robustez del sistema y la aparición de posibles fallos en el sistema y sus consecuencias asociadas.
- *Evolucionabilidad* (*evolvability*), asociada a la respuesta del sistema ante la necesidad de incluir nuevos cambios, ya sea en el hardware o en el software.
- *Integración* (*integration*), como la capacidad de la arquitectura para incluir nuevos dispositivos.
- *Seguridad* (*security*), en relación a los mecanismos proporcionados para resistir contra un uso inadecuado o no autorizado del sistema.
- *Administración* (*manageability*), como la relación con el personal de seguridad a la hora de interactuar con el sistema de vigilancia.

Respecto a la ***escalabilidad***, la arquitectura diferencia entre dos cuestiones. Por una parte, el soporte proporcionado cuando se incorpora un nuevo sensor al entorno de monitorización y, por otra, la necesidad de estudiar o analizar un nuevo concepto o amenaza. Así mismo, a la hora de analizar un nuevo concepto, también es posible diferenciar entre análisis de normalidad y análisis de anormalidad.

[2] http://www.acvt.com.au/research/surveillance/

[3] La traducción de algunos de los términos se ha realizado en función del concepto más utilizado en castellano.

La incorporación de un nuevo sensor de vigilancia en el entorno monitorizado se realiza en el nivel reactivo de la arquitectura y, dependiendo del tipo de sensor, será necesario asociar un *preprocessing agent* o no (ver sección 3.3). De cualquier modo, en este contexto la escalabilidad se alcanza mediante el mecanismo de comunicación basado en eventos que comunica los niveles reactivo y deliberativo. Como ya se discutió en la sección 3.4.2, los canales de eventos independizan el número de productores de información (sensores o *preprocessing agents*) del número de consumidores (*normality agents*). Esta cuestión está directamente ligada con la ***integración***, por lo que no se discutirá posteriormente.

A nivel de análisis de conceptos, la escalabilidad se alcanza mediante los *normality agents* y los *abnormality agents*, respectivamente, en función del tipo de análisis de normalidad. En el caso de que sea necesario un proceso de fusión de información, la arquitectura soporta el despliegue del *fusion agent* asociado. Así mismo, la escalabilidad a nivel de comunicación se logra mediante el despliegue de pizarras, como mecanismo para compartir el conocimiento vinculado a un determinado concepto.

La ***disponibilidad*** se ha gestionado desde dos puntos de vista. Por una parte, este requisito mantiene un enlace con el anterior respecto a los mecanismos de comunicación. En este sentido, la consecuencia de que la ejecución de un productor de información termine de manera inesperada es que los consumidores asociados (mediante un canal de eventos, por ejemplo) dispondrán de menos información para analizar el entorno. En el peor de los casos, no dispondrán de información alguna, pero no ligarán su destino al del productor de información que falló por cualquier motivo.

Por ejemplo, si el agente encargado del análisis de la normalidad na_i del concepto c_j en el sub-entorno E_k deja de funcionar o en la máquina en la que se está ejecutando ocurre algún tipo de problema, dicho fallo repercutirá en el rendimiento del sistema de vigilancia del siguiente modo:

- Si na_i no estaba replicado, el concepto c_i no se analizará en E_k, pero de manera independiente al resto de conceptos analizados en dicho sub-entorno.
- Si existe un agente de anormalidad aa_i que analiza el mismo concepto c_i (pero desde el punto de vista de la anormalidad), dicho agente no podrá llevar a cabo tal análisis utilizando la información generada por na_i debido a que éste no genera nuevo conocimiento. Sin embargo, esta cuestión dependerá del modelo de razonamiento interno de aa_i y de si éste utiliza o no el conocimiento generado por na_i.
- El agente de análisis de normalidad a nivel de sub-entorno ena_i que agrega la información de normalidad de todos los conceptos analizados en el sub-entorno E_k no dispondrá del conocimiento de na_i, por lo que el valor de normalidad global se verá afectado. No obstante, dicho valor para el sub-entorno E_k seguirá siendo correcto para monitorizar la

normalidad global, aunque sin tener en cuenta c_i, debido al esquema de agregación de conocimiento planteado por el modelo de análisis de normalidad de la sección 3.3.

Por otra parte, la arquitectura soporta la replicación de componentes para garantizar la tolerancia a fallos. El esquema adoptado, tal y como se discutirá en profundidad en la sección 4.5, consiste en utilizar un esquema de replicación maestro-esclavos teniendo en cuenta que dichos componentes pueden tener un estado interno (como ocurre con los agentes del nivel deliberativo).

La ***evolucionabilidad*** de la arquitectura se ha afrontado mediante la definición de las interfaces y los tipos de datos que representan a cada uno de los componentes de las misma (ver anexo B.1 para profundizar en esta cuestión). Este enfoque garantiza la independencia respecto a ciertos aspectos típicos asociados a los cambios del sistema, como por ejemplo el software, el hardware o incluso el sistema operativo. En la sección 5.2 se plantea un caso de estudio relativo a la elección de una determinada tecnología para desarrollar e integrar un agente de análisis de normalidad en un sistema de vigilancia soportado por esta arquitectura.

Respecto a los mecanismos de ***seguridad*** manejados por la arquitectura, hay que tener en cuenta la implementación final del propio sistema de vigilancia para evaluar este aspecto. Sin embargo, la arquitectura considera ciertas técnicas como la habilidad para proteger información, asegurar su integridad o verificar las identidades de los componentes que se comunican en un determinado momento. En este contexto, el prototipo desarrollado a partir de la arquitectura soporta el protocolo SSL.

No obstante, existen ciertas cuestiones que se han de tener en cuenta, como la propia seguridad física de los sensores utilizados para monitorizar un entorno. En este ámbito, habría que realizar un estudio de los mecanismos utilizados por el sistema de vigilancia particular para garantizar la integridad física de dichos dispositivos. Del mismo modo, y a nivel de sistema concreto, habría que considerar la propia seguridad de los algoritmos o de las técnicas utilizadas para llevar a cabo la vigilancia inteligente, es decir, las posibles vulnerabilidades de dichas técnicas.

Finalmente, en términos de ***administración*** de la arquitectura se ha prestado especial importancia al despliegue de sistemas de vigilancia (ver sección 3.5). Debido a que la mayoría de funcionalidades de la arquitectura están proporcionadas por agentes, uno de los mecanismos para facilitar su despliegue son las fábricas de agentes (ver sección 5.2.4). Sin embargo, los sistemas de vigilancia desarrollados y desplegados mediante la arquitectura propuesta están guiados por la autonomía, reduciendo la necesidad de administración y/o control directo por parte del personal de seguridad. En este sentido, el principal tipo de interacción por parte de dicho personal es la obtención de información mediante los módulos de consulta de los agentes de análisis de normalidad y anormalidad. En cuestiones de control directo por

parte de los operadores sería necesario llevar a cabo un estudio de los requisitos funcionales y un plan de viabilidad para integrar dichas características a la arquitectura.

En [DvdHD+08], los autores plantean que el rendimiento (*performance*) en tiempo real es identificado como un requisito sistemático por una gran parte de los investigadores. Sin embargo, los mismos autores se posicionan en la necesidad de desarrollar software de vigilancia robusto y eficaz como primera meta, asignándole una mayor importancia que al rendimiento en tiempo real en sí mismo. No obstante, en este trabajo se destacarán algunos de los resultados obtenidos a la hora de evaluar de manera empírica ciertas partes de la arquitectura propuesta:

- La discusión de un sistema multi-agente concreto para vigilar un entorno de tráfico urbano se discute en la sección 5.1.2. En esta misma sección se detalla cómo el prototipo desarrollado soporta dicha monitorización y se evalúa su rendimiento y escalabilidad bajo diversos supuestos.
- La evaluación del sistema de comunicación directa entre agentes se discute en profundidad en la sección 5.3.1.
- La evaluación del consumo de recursos por parte de agentes implementados en distintos lenguajes de programación, con y sin soporte de hibernación, se discute en profundidad en la sección 5.3.2.
- El rendimiento asociado a la implementación de un agente de normalidad se detalla en la sección A.3.
- En [Hen09] se evalúa el *middleware* utilizado para desarrollar el prototipo funcional de la arquitectura de vigilancia. Así mismo, en la propio web de ZeroC ICE se estudian aspectos de rendimiento, como por ejemplo el asociado al servicio de publicación de eventos[4].

3.5. Uso de la arquitectura multi-agente para desplegar sistemas de vigilancia inteligente

Desde un punto de vista abstracto, y atendiendo al caso particular del modelo de análisis de normalidad descrito en la sección 3.3, la construcción de un sistema de vigilancia en base a una serie de componentes implica la gestión de los siguientes aspectos:

1. Identificar el conjunto de sensores S necesarios y/o disponibles en el entorno a vigilar junto con los mecanismos necesarios para procesar la información obtenida de los mismos (PA), cuando así sea necesario.

[4] http://zeroc.com/blogs/matthew/category/performance/

2. Establecer la división del entorno global en base a una serie de percepciones o sub-entornos $P = \{E_1, E_2, \ldots, E_n\}$ con el objetivo de simplificar la vigilancia del mismo. La opción más inmediate consiste en la división física de dicho entorno.

3. Determinar los objetos móviles O que se van a vigilar o, de forma general, a los elementos del entorno a los que hay que prestar especial atención.

4. Diseñar o reutilizar, desde una perspectiva general, aquellos componentes o módulos de análisis desarrollados para monitorizar distintos aspectos en un mismo entorno.

Este conjunto de pasos generales tienen un reflejo directo en la arquitectura propuesta respecto a los tres niveles que la forman. A nivel reactivo será necesario tanto integrar los sensores empleados para la vigilancia como desarrollar los agentes de preprocesamiento asociados a los mismos. A nivel deliberativo será necesario desplegar los agentes de análisis necesarios para estudiar cada concepto $c_i \in C$ monitorizado, así como los mecanismos de comunicación necesarios para la permitir la interacción y cooperación entre los mismos. Finalmente, y a nivel de usuario, tanto las herramientas de monitorización como las de adquisición de conocimiento interactuarán con los agentes responsables del análisis del entorno global.

A continuación, y desde un punto de vista general, se detallan el i) el despliegue y configuración de los agentes de vigilancia y ii) el despliegue del sistema de comunicaciones que da soporte a los aspectos previamente mencionados para la construcción de sistemas de vigilancia.

3.5.1. Despliegue y configuración de los agentes de vigilancia

El despliegue del conjunto de agentes necesarios para realizar la vigilancia inteligente de un determinado entorno dependerá directamente de aquellos conceptos o amenazas que se deseen monitorizar. A continuación se irán detallando, desde un punto de vista general, qué agentes se instanciarán en cada uno de los niveles que forman la arquitectura (nivel reactivo, nivel deliberativo y nivel de usuario).

A **nivel reactivo**, se distinguen dos grupos de agentes: i) aquellos responsables de procesar la información recogida por el conjunto de sensores S desplegados en el entorno, es decir, los *preprocessing agents* PA, y ii) aquellos encargados de las operaciones de gestión básicas del sistema multi-agente, es decir, $MA = \{pm, sm, cm\}$, recordando que pm representa al gestor de la plataforma de agentes, sm es el gestor de servicios o agente de páginas amarillas, y cm es el agente gestor de comunicaciones.

A nivel de sub-entorno E_i, se desplegará un agente de preprocesamiento pa_j por cada tipo de sensor existente en dicho sub-entorno E_i y que requiera un procesamiento de información, como en el caso de los sensores de vídeo o de audio. Por ejemplo, si un sub-entorno E_1 está compuesto de una cámara de seguridad s_1, un micrófono s_2 y un sensor de presencia s_3, entonces típicamente se desplegarán un agente de preprocesamiento de vídeo pa_1 y un agente de preprocesamiento de audio pa_2, los cuales analizarán la información obtenida de los sensores s_1 y s_2, respectivamente. En este caso, s_3 no requerirá el despliegue de otro agente, ya que la salida del mismo es un valor *booleano* que no necesita ningún tipo de procesamiento antes de que dicha información sea notificada al nivel deliberativo. De manera análoga se procederá con el resto de sub-entornos que componen el entorno global.

El despliegue de los agentes de preprocesamiento PA (y de cualquier agente del sistema de vigilancia en general) implica dos acciones básicas de gestión:

1. Registro con el agente de gestión de la plataforma pm para obtener un identificador válido y único dentro de la misma.
2. Registro con el agente de gestión de servicios sm, con el objetivo de que el resto de agentes puedan contactar con el agente que se registra en el caso de que necesiten sus servicios.

Debido a que estos dos agentes de gestión (pm y sm) se han diseñado de acuerdo a las especificaciones del comité FIPA, es posible hacer uso de las ontologías definidas por FIPA para mantener un esquema formal y homogéneo a la hora de compartir información, como por ejemplo la utilizada para llevar a cabo los registros previamente mencionados. Por ejemplo, en el caso del registro de un agente con el gestor de servicios, o *Directory Facilitator* según FIPA, esta información permite definir datos clave como los tipos de servicios ofrecidos por el agente o el sub-entorno en el que se encuentran (ver el tipo de objeto *service-description* del documento *FIPA Agent Management Specification* [Fou04]).

A **nivel deliberativo**, se puede establecer una distinción en función del alcance del análisis de vigilancia llevado a cabo por los agentes que forman parte de este nivel, distinguiendo entre un alcance local a nivel de sub-entorno E_i o a nivel global (vea definición 3.1 en la sección 3.3). Así, los agentes de análisis de normalidad a nivel de concepto NAA, de análisis de anormalidad a nivel de concepto AAA y de análisis de normalidad a nivel de escenario $ENAA$ tienen un alcance local desde la perspectiva de sub-entorno E_i. Por el contratio, los agentes que componen el módulo de fusión de información $GNAM$ y de apoyo a la toma de decisiones DSM tienen un alcance global.

De este modo, a nivel de sub-entorno E_i se desplegará un agente de análisis de normalidad na_j por cada aspecto c_k que se desee analizar en di-

cho sub-entorno E_i. Por ejemplo, si en un entorno E_i se pretenden analizar las trayectorias de los objetos móviles c_1 y el comportamiento normal de los vehículos y peatones respecto a un paso de peatones c_2, entonces se desplegarán dos agentes de normalidad na_1 y na_2 para cada aspecto c_1 y c_2, respectivamente. De manera opcional, por cada agente de análisis de normalidad na_j se podrá desplegar un agente de análisis de anormalidad aa_j que complemente el análisis realizado por na_j. Cuando el agente na_j detecte una situación que no se corresponde con la definición de normalidad c_j para el entorno E_i, na_j se comunicará con el agente aa_j para que éste detecte la situación anómala que se produjo (siempre que esté reflejada en la base de conocimiento $\widetilde{KB}_j$ de aa_j).

Así mismo, por cada sub-entorno E_i se desplegará un único agente de análisis de normalidad a nivel de sub-entorno ena_i, responsable de comprobar si todo objeto obj monitorizable cumple la normalidad asociada a los conceptos que están siendo monitorizados en E_i (ver definición 3.10 en la sección 3.3).

Para realizar una vigilancia del entorno de manera global es necesario instanciar los agentes encargados de fusionar el conocimiento obtenido por los agentes previamente mencionados. Este proceso implica desplegar los siguientes agentes:

1. Un agente ta encargado de traducir la información local manejada por los $ENAA$ en información global que pueda ser utilizada por el resto de agentes del módulo $GNAM$ (ver figura 3.13).
2. Un agente fma encargado de decidir si es necesario fusionar la información obtenida de distintos agentes de normalidad o anormalidad, así como de decidir qué agentes de fusión serán los responsables de efectuar tal proceso.
3. Un agente de fusión de información fa_i por cada uno de los conceptos c_i analizados a nivel de sub-entorno E_j, responsable de realizar un análisis más sofisticado del entorno debido a que posee una percepción global del mismo. Un caso típico consistirá en fusionar la información asociada a un mismo objeto obj obtenida por dos agentes de normalidad na_k y na_l desplegados en dos sub-entornos distintos, pero responsables de analizar un mismo aspecto c_i.
4. Un agente de fusión global ga encargado de fusionar la información generada tanto por los agentes de fusión de información FA, como por los agentes de análisis de normalidad NA, $ENNA$ y anormalidad AA para proporcionar una salida global más sofisticada.

Finalmente, se desplegará un único agente de apoyo a la toma de decisiones $dssa$ como mediador entre el sistema multi-agente de vigilancia y el guardia de seguridad encargado de utilizar las herramientas de monitorización en lo que a gestión de alarmas y de posibles crisis se refiere.

3.5.2. Despliegue del sistema de comunicaciones

Una vez abordado el despliegue de los agentes necesarios para vigilar un determinado entorno, dependiendo de las cuestiones a monitorizar y a las funcionalidades requeridas, el siguiente paso consiste en habilitar los mecanismos de comunicación necesarios (basado en eventos, comunicación directa y arquitectura de pizarra) para que dichos agentes puedan cooperar y colaborar para la vigilancia global del entorno.

A **nivel reactivo** (ver figura 3.13), el tipo de comunicación estará basada en eventos y canales de eventos, sirviendo estos últimos como el pegamento entre el nivel reactivo y el nivel deliberativo. De este modo, la información facilitada tanto por los sensores S como los agentes de preprocesamiento PA desplegados representará el flujo de entrada de estos canales de eventos. Por otra parte, los agentes de análisis de normalidad NAA del nivel deliberativo serán los receptores (suscriptores) de esta información. Dependiendo del tipo de información recogida en cada escenario E_i que forme parte del entorno global, se podrán instanciar uno o varios canales de eventos a nivel de escenario E_i. Sin embargo, la práctica más común será la de desplegar un canal de eventos por cada tipo de información (visual, sonora, sensorial...) que se pueda obtener a nivel de escenario E_i. En este mismo nivel se desplegarán una o varias instancias del *Message Transport Service*, encargado de gestionar la comunicación directa entre agentes del nivel deliberativo.

A **nivel deliberativo** (ver figura 3.13) hay que destacar en primer lugar la comunicación entre un agente de normalidad na_i y un agente de anormalidad aa_i en relación a una determinada amenaza o concepto c_i, en el caso de que las características del análisis requieran la detección de las situaciones anómalas más comunes de acuerdo a c_i. Bajo este supuesto, na_i ha de conocer la existencia de aa_i, información que puede conocer preguntando directamente al agente de páginas amarillas para obtener la dirección física de contacto del agente de anormalidad aa_i. Este hecho posibilita la comunicación directa entre ambos agentes.

Dentro de este mismo nivel y en el ámbito de un sub-entorno E_i, se despliega un sistema de pizarra en cada uno de los sub-entornos $E_1, E_2, \ldots, E_n$ que forman el escenario global de vigilancia. De este modo, los agentes de normalidad que forman parte de cada E_i pueden escribir el conocimiento generado en cada paso del proceso de vigilancia en la pizarra. Ésta será utilizada por el agente de análisis de normalidad a nivel de sub-entorno ena_i para estudiar la normalidad de los objetos monitorizables en E_i, de acuerdo al conjunto de conceptos $C_i \subseteq C$ analizados en E_i.

En el contexto del módulo global de fusión de información GNA se despliega un sistema de pizarra en el que los agentes de fusión FA irán anotando el conocimiento generado a nivel de concepto c_i, pero desde una perspectiva global. Mientras tanto, el agente de soporte a la toma de decisiones y a la gestión de crisis $dssa$ hará uso de las distintas pizarras desplegadas en el nivel

deliberativo y de la comunicación directa con el resto de agentes mediante sus respectivos módulos de consulta.

A **nivel de usuario** existen dos formas de comunicación posibles, una es el módulo de consulta integrado en los agentes del nivel deliberativo, y otra es la notificación de eventos por parte de dichos agentes a las herramientas de monitorización utilizadas en la capa de usuario. En el primer caso, los agentes simplemente responden a las peticiones de información realizadas por el usuario desde las herramientas de monitorización, actuando de un modo reactivo. En el segundo caso, los agentes son los que van notificando la información que consideren relevante para la vigilancia del entorno utilizando los canales de eventos. Por este motivo, la práctica más común será desplegar los siguientes canales de eventos:

- Uno por cada entorno $E_i \in E$ en el que publicarán información tanto los agentes de anormalidad AA_i como el agente de normalidad a nivel de sub-entorno ena_i.
- Un canal de eventos en el que publicarán información los distintos agentes del módulo de análisis global de normalidad GNA.
- Un canal de eventos dedicado exclusivamente al agente de soporte a la toma de decisiones y a la gestión de crisis $dssa$, que le permita apoyar al personal de securidad con la sugerencia de posibles acciones en respuesta a la detección de una determinada situación anómala.

3.6. El proceso de vigilancia inteligente

En esta sección se muestra cómo se realiza el proceso de vigilancia una vez desplegado el sistema a partir de la arquitectura multi-agente propuesta. Este proceso se puede entender como un bucle de recepción de eventos de vigilancia de la capa reactiva, de procesamiento de los mismos en la capa deliberativa y, finalmente, de notificación del conocimiento generado tras el análisis de dichos eventos al nivel de usuario mediante las herramientas de monitorización desplegadas. A continuación se muestra cómo se lleva a cabo este proceso a nivel de sub-entornos locales y a nivel de entorno global, mostrando las interacciones entre los agentes de vigilancia propuestos.

3.6.1. Vigilancia a nivel de sub-entorno E_i

La primera parte del proceso de vigilancia consiste en llevar a cabo los análisis de normalidad y anormalidad a nivel de sub-entorno E_s, utilizando para ello un subconjunto de agentes de preprocesamiento $PA_s \subseteq PA$, un

subconjunto de agentes de normalidad $NA_s \subseteq NA$, un subconjunto de agentes de anormalidad $AA_s \subseteq AA$ y un agente de análisis de normalidad del entorno ena_s. En la figura 3.20 se muestra el flujo normal de interacción de un sub-entorno E_i con un agente de cada tipo de los recientemente mencionados. En esta sección supondremos el caso particular de la videovigilancia para detallar el funcionamiento de los agentes que forman parte del proceso de vigilancia inteligente.

Bajo dicha suposición, el agente de preprocesamiento pa_i sería el responsable de recibir la información capturada de una cámara de vigilancia y de efectuar los procesos de segmentación y *tracking*, de manera que cada vez que dicho agente reciba información de la cámara de vídeo sea capaz de especificar qué objetos monitorizables se encuentran dentro del campo de visión del sensor de vídeo, en qué posición se encuentran y de qué tipo son. Esta información básica, especialmente la clase a la que pertenece un objeto, reflejará comúnmente cierta incertidumbre debido a que en determinadas ocasiones no es posible asegurarla con total certeza. Por ejemplo, pa_i podría establecer que un objeto identificado con la etiqueta obj_1 se encuentra en la posición $[320, 460]$, siendo 320 el pixel que representa la localización en el eje x de obj_1 y 460 el pixel de su localización en el eje y, y pertenece a la clase *coche* con un valor de pertenencia de $0{,}8$ (sobre $1{,}0$) y a la clase persona con un valor de $0{,}2$, respectivamente.

A continuación, el agente de preprocesamiento pa_i envía dicha información mediante un canal de eventos que recibirán los suscriptores, en este caso representados por el agente de normalidad na_j en la figura 3.20. Este paso de información se refleja gráficamente y de manera abstracta por la operación *análisis*. Con esta información, na_j es capaz de utilizar su base de conocimiento para inferir cómo progresa la monitorización de su sub-entorno asociado mediante su base de conocimiento $\widetilde{KB}_j$ sobre el concepto c_k. Si este agente hace uso del modelo discutido en la sección 3.3, el proceso *normality analysis* sería el encargado de efectuar los cálculos asociados a la ecuación 3.8 para todo objeto identificado por el agente de preprocesamiento pa_i.

El conocimiento generado por na_j se notifica a las herramientas de monitorización que permitan la visualización de este tipo de información, es decir, la evolución del proceso de vigilancia a nivel de sub-entorno. Así mismo, dicho conocimiento se anota en la pizarra asociada al sub-entorno de na_j, de modo que ena_j pueda realizar el análisis a nivel de sub-entorno. Este último agente, en el proceso *análisis de normalidad del sub-entorno* de la figura 3.20 y de acuerdo al modelo de la sección 3.3, es responsable de efectuar los cálculos asociados a la ecuación 3.11 para todos los objetos identificados a nivel de sub-entorno E_i por todos los agentes de preprocesamiento PA_i vinculados.

Finalmente, el análisis de normalidad para el concepto c_k analizado por na_j se puede complementar con el análisis de anormalidad efectuado por el agente aa_j para el mismo concepto c_k. En este caso, y mediante la operación

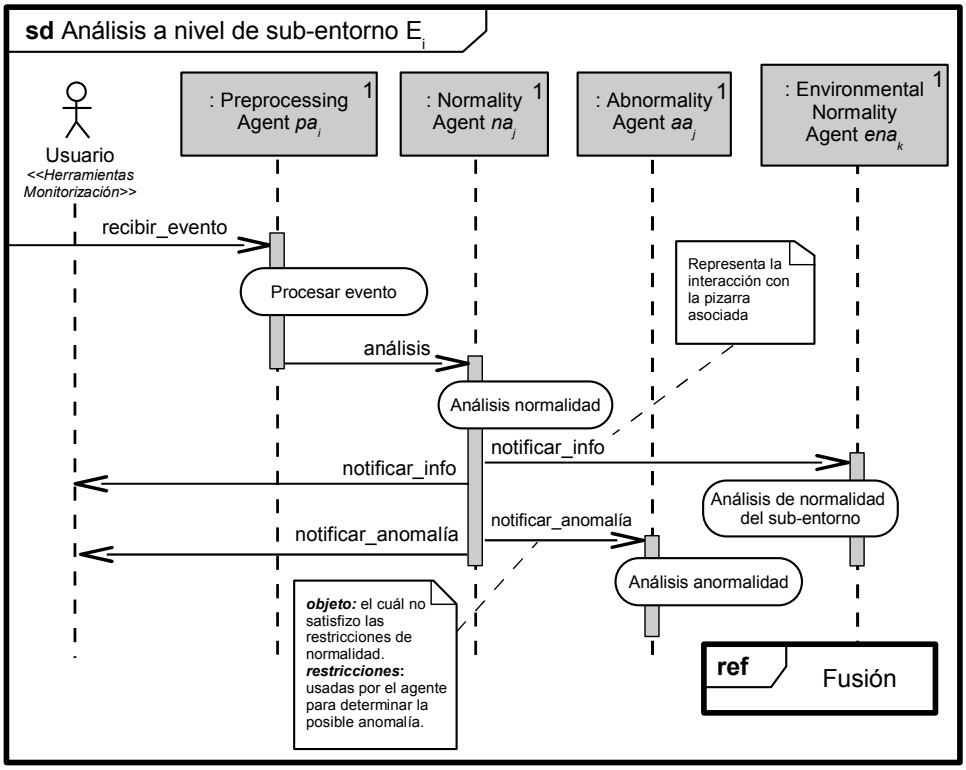


Figura 3.20: Diagrama de secuencia en notación AUML del análisis de un sub-entorno E_i.

análisis normalidad, aa_j recibe la información necesaria para identificar la situación anómala siempre y cuando esté recogida en la base de conocimiento $\widetilde{KB}_j$ de aa_j. Esta información consiste en los valores de cumplimiento de las restricciones de normalidad de todas las instancias del concepto c_k asociadas a un objeto particular obj en el último instante de tiempo en el que se analizó la normalidad del sub-entorno. Con esta información y con la información temporal anterior almacenada en su estado interno, aa_j intentará detectar si el comportamiento del objeto obj en los últimos instantes de tiempo se asemeja a un comportamiento anómalo definido explícitamente en su base de conocimiento.

3.6.2. Vigilancia a nivel global

La segunda parte del proceso de vigilancia se define como la fusión de información obtenida desde los distintos entornos $E_1, E_2, \ldots, E_n$ que componen el escenario completo de vigilancia. Esta información está representada principalmente por los agentes de análisis de normalidad a nivel de entorno $ena_1, ena_2, \ldots, ena_n$, de manera que cada ena_k conoce los conceptos o amena-

zas $C_i \subseteq C$ que se monitorizan en cada sub-entorno E_i y qué objetos monitorizables se están comportando o no de acuerdo a la definición de normalidad de dichos conceptos. En la figura 3.21 se muestra el diagrama de interacción que refleja cómo los agentes que participan en esta etapa de fusión se comunican para llevar a cabo la vigilancia global de un escenario.

En primer lugar, y debido a que cada ena_k maneja información a nivel local o de sub-entorno E_k, es necesario llevar a cabo un proceso de traducción de información local a información global, realizado por el agente de traducción global gta. En el caso particular de la videovigilancia, este proceso implicaría, entre otras cuestiones, traducir las posiciones locales de un objeto obj_x de los sub-entornos $E_j \subseteq E$ en una posición global, siempre que la relación física entre los entornos E_j lo permita.

A continuación, cada ena_k notifica la información de análisis de normalidad a nivel de sub-entorno E_k al agente de gestión de fusión fma responsable de decidir si la fusión de información es necesaria. En tal caso, fma establece qué agente de fusión de información fa_i será el encargado de agregar el conocimiento de los distintos agentes de análisis de normalidad. En el caso de la videovigilancia, la decisión de fusionar información o no podría definirse en base a la relación física entre los sub-entornos analizados, es decir, en base a si es posible relacionar la información de posicionamiento de un subconjunto de sub-entornos $E_j \subseteq E$, normalmente vinculada a las relaciones de adyacencia física existentes en los mismos.

Como se introdujo en la sección 3.5, por cada concepto $c_k \subseteq C$ analizado se despliega un sistema de pizarra utilizado por los distintos agentes de análisis de normalidad a nivel de sub-entorno, con el objetivo de mantener un punto común en el que publicar el conocimiento que se va generando. De este modo, el agente de fusión de información fa_i asociado al concepto o amenaza c_k puede leer de la pizarra y agregar el conocimiento obtenido por los distintos agentes de normalidad locales. Ante esta situación se plantean dos problemas inherentes a los sistemas de pizarra:

1. Es necesario algún mecanismo de exclusión mutua que permita el correcto acceso a la pizarra, ya sea de lectura o de escritura.
2. Los agentes de fusión de información han de estar dotados de algún mecanismo que les permita determinar cuándo efectuar la fusión de información en función del contenido de la pizarra.

El primer problema se ha solucionado haciendo uso de un mecanismo de exclusión mutua (monitor) para garantizar el acceso correcto a la pizarra. El segundo problema depende directamente del concepto c_k manejado por el agente de fusión de información y del estado actual de la pizarra antes de llevar a cabo la fusión propiamente dicha, por lo que la base de conocimiento $\widetilde{KB}_i$ de cada fa_i deberá reflejar algún mecanismo para tratar este problema. En el caso de la videovigilancia, nuestra solución se basa en agregar informa-

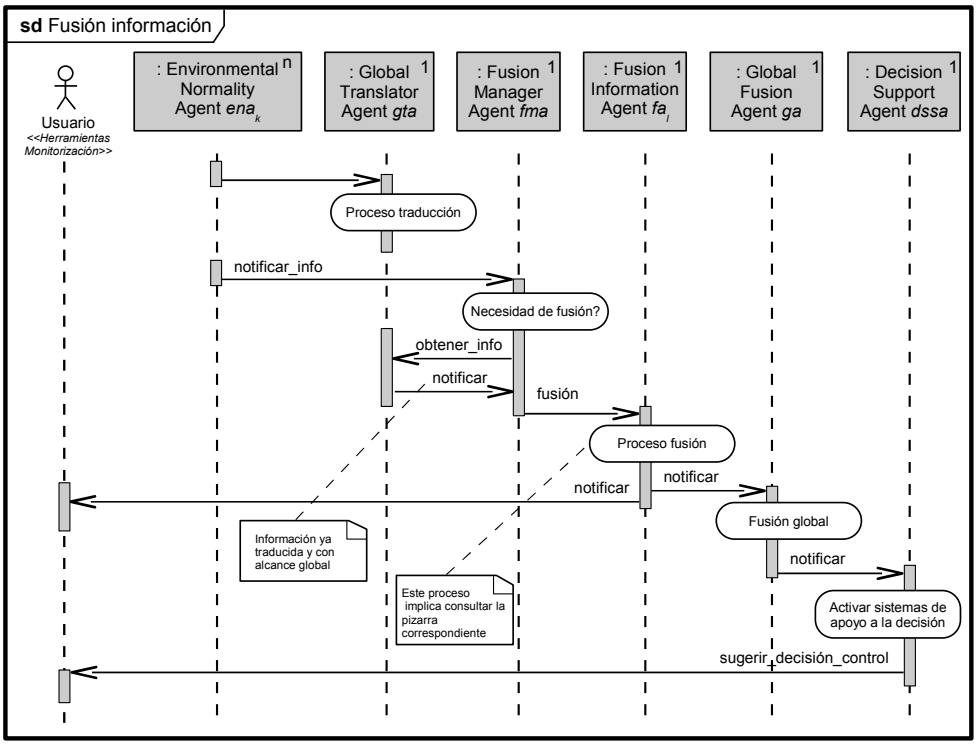


Figura 3.21: Diagrama de secuencia en notación AUML del proceso de fusión de información.

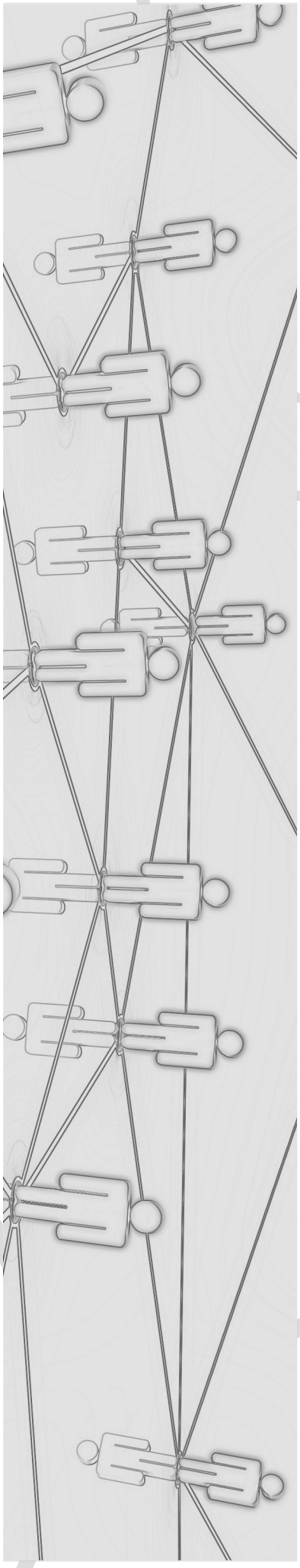
ción tan pronto como sea posible, es decir, cuando el contenido de la pizarra asociada a un determinado concepto c_k lo permita, obteniendo de este modo conclusiones globales que evolucionan temporalmente cuando se añade más conocimiento a la pizarra.

Finalmente, la arquitectura contempla la interacción entre los distintos agentes de fusión FA desplegados y un agente responsable de la fusión global ga a partir de las conclusiones obtenidas de manera individual por parte de dichos agentes de fusión. Típicamente, la tarea de ga será la de intentar relacionar el análisis de normalidad global de uno o varios conceptos o amenazas para efectuar una vigilancia más sofisticada. En el caso de la videovigilancia, este agente global podría aportar servicios de valor añadido en casos de emergencia, donde la normalidad asociada a un determinado concepto c_k se ve alterada por la excepcionalidad de la situación. Por ejemplo, si se detecta una situación de anormalidad en el estado de un inmueble debido a un incendio, y otra situación de anormalidad debido a la conducción temeraria de un vehículo en una zona cercana al incendio, este agente podría ser capaz de inferir que, dentro de la excepcionalidad de la situación, el vehículo se está comportando de manera normal ya que probablemente esté huyendo del incendio o se esté acercando para prestar ayuda (en el caso de que sea un vehículo de emergencias como por ejemplo un camión de bomberos).

Todo el proceso de análisis de normalidad y anormalidad de un entorno, ya sea local o global, tiene como responsable final al agente de soporte a la decisión y a la gestión de crisis, como mecanismo para sugerir posibles acciones al operador humano que utiliza las herramientas de monitorización o al guardia de seguridad. Este agente se tiene que entender como un apoyo al personal humano y no como un sustituto del mismo, debido a la gran responsabilidad que supone tomar una decisión ante una situación potencialmente peligrosa en un escenario de vigilancia. Para proporcionar este soporte, el agente de apoyo a la toma de decisiones $dssa$ utiliza el conocimiento albergado en las distintas pizarras y los módulos de consulta de los distintos agentes de análisis, interactuando directamente con las herramientas de monitorización en el caso de sugerir una posible acción. Una posible materialización de este agente podría consistir en un sistema basado en reglas en el que los consecuentes de dichas reglas serían las acciones a sugerir en un lenguaje cercano al natural.

Capítulo 4

Plataforma Multi-Agente



Capítulo 4
Plataforma Multi-Agente

En este capítulo se discuten las soluciones generales que se extraen a partir de la arquitectura multi-agente para la vigilancia inteligente, las cuales han sido utilizadas y extendidas para diseñar y desarrollar la plataforma de agentes propuesta a continuación. Dicha plataforma ha sido diseñada siguiendo los principios marcados por el comité FIPA (Foundation for Intelligent Physical Agents) para el desarrollo de sistemas multi-agente.

"A pessimist sees the difficulty in every opportunity; an optimist sees the opportunity in every difficulty."

Sir Winston Churchill

4

Plataforma Multi-Agente de Propósito General

En este capítulo se discuten las soluciones de diseño generales que se extraen a partir de la arquitectura multi-agente para la vigilancia inteligente que se describió en profundidad en las secciones 3.4, 3.5 y 3.6. La materialización de dichas soluciones son aquellos servicios de la arquitectura que se pueden reutilizar en diversos dominios, los cuales pueden estar o no relacionados con el área de la vigilancia inteligente. La principal ventaja de este enfoque es que hace posible su utilización directamente a la hora de desplegar sistemas multi-agente como si se tratara de componentes *plug&play*, facilitando las tareas de administración, configuración, desarrollo e interacción entre agentes al desarrollador final del sistema multi-agente que haga uso de ellos. Estos servicios generales se han utilizado para construir una plataforma que facilite dichas tareas [VAM+09]. En este capítulo se hará uso del término **plataforma** para referirse a la extensión de estas soluciones generales.

Para el diseño de estos servicios generales, normalmente ligados a tareas de gestión del sistema multi-agente, se han seguido las especificaciones propuestas por el comité FIPA (*Foundation for Intelligent Physical Agents*) [Fou04] para el desarrollo de sistemas multi-agente. Esta adopción se justifica mediante dos aspectos fundamentales: la madurez de dichas especificaciones y la interoperabilidad proporcionada con otros desarrollos que hagan uso de este conjunto de estándares.

Por otra parte, para desarrollar e implementar el prototipo de la plataforma de agentes se ha utilizado el *middleware* ZeroC ICE (ver sección 2.4.2). La elección de esta moderna alternativa a la hora de desarrollar aplicaciones distribuidas ha motivado ciertas decisiones de diseño que contribuyen a la creación de una nueva alternativa en relación a las plataformas de agentes existentes.

4.1. Motivación

Actualmente, los Sistemas Multi-Agente (SMA) están aplicándose como solución a una gran diversidad de problemas [Wei99], como planificación, control de sistemas de tiempo-real, robótica y otros campos relacionados con la industria. Esta expansión en el uso de SMA también se refleja en la existencia de modelos de Ingeniería del Software basados en el uso de agentes autónomos diseñados para solucionar problemas complejos [Jen00], la definición de metodologías de desarrollo para SMA [WC01], el uso de lenguajes específicos para su programación [BBD+06] y la adopción de estándares [Fou04] para fomentar la interoperabilidad.

Esta evolución también se puede apreciar en las plataformas de desarrollo software para SMA, tal y como se expuso en la tabla 2.2. Gracias a este tipo de herramientas, los desarrolladores de sistemas orientados a agentes pueden centrar su trabajo principalmente en la aplicación multi-agente, en lugar de tratar con cuestiones de gestión. Actualmente, **JADE** (Java Agent DEvelopment framework) [BCPR08] es probablemente el *framework* orientado a agentes más extendido, y se puede entender como un entorno distribuido y modular que facilita el desarrollo de aplicaciones basadas en agentes. Esta herramienta implementa el ciclo de vida de los agentes y la lógica de gestión básica, proporcionando además herramientas administrativas para desplegar, monitorizar y depurar sistemas multi-agente [BCG07].

Normalmente, los desarrolladores suelen hacer uso de algún *framework* existente para realizar el despliegue del sistema y no emplear mucho tiempo en la gestión y configuración del mismo. Las principales ventajas de este enfoque son las facilidades proporcionadas por el *middleware* para llevar a cabo las tareas anteriormente mencionadas. Sin embargo, la principal desventaja es que el desarrollador estará condicionado por las características de la plataforma escogida. De hecho, la mayoría de estas herramientas, aunque concebidas desde un punto de vista general y aplicables a un rango amplio de problemas, padecen de ciertas **limitaciones** que están directamente ligadas a sus desarrollos particulares:

- La mayoría están vinculadas a alguna tecnología que limita su expansión a otras plataformas (debido a restricciones hardware o de consumo de recursos).
- La utilización de tecnologías interpretadas en máquinas virtuales hace que decremente el rendimiento de la aplicación frente a implementaciones en código nativo de los dispositivos (ver resultados experimentales en la sección 5.3).

De este modo, algunos dominios de aplicación, como los sistemas que requieren un tiempo de respuesta corto, necesitan soluciones que solventen estas limitaciones. Para afrontar este reto, la comunidad de agentes ha de

proporcionar alternativas que contribuyan a la evolución de las plataformas para el desarrollo de sistemas multi-agente. Por lo tanto, uno de los principales motivos que se pretende alcanzar con el diseño de la plataforma de agentes planteada en esta sección es ofrecer una nueva alternativa que se sustente sobre un moderno *middleware* de comunicaciones [Hen04b] y que permita la extensión de la plataforma propuesta a distintos lenguajes de programación, sistemas operativos y entornos hardware.

Otra de las cuestiones relevantes a la hora de afrontar este diseño consiste en garantizar la **interoperabilidad** entre distintas plataformas. Por este motivo, la propuesta que se irá discutiendo en las siguientes secciones está basada en las especificaciones proporcionadas por el comité FIPA para el desarrollo de SMA.

Finalmente, dicha propuesta está basada en principios de Ingeniería del Software ampliamente aceptados y utilizados, como el uso de patrones de diseño y de plantillas. Por otra parte, se ha utilizado un *middleware* orientado a objetos que proporciona herramientas, APIs y bibliotecas para construir aplicaciones distribuidas cliente-servidor. La justificación de utilizar un enfoque orientado a objetos se debe a dos cuestiones principales:

- los agentes se pueden entender como una evolución de los objetos con nuevas características como la autonomía y la proactividad,
- las especificaciones propuestas por FIPA muestran una semántica orientada a objetos para describir conceptos y ontologías.

A continuación se resumen las **características** más importantes de la propuesta:

- Implementación del agente básico de gestión en distintos lenguajes de programación (C++, Java y Python pero extensible a C#, Visual Basic y Ruby) para que el desarrollador pueda elegir la opción más adecuada para su sistema.
- Escalabilidad en el número de agentes y tiempos de respuestas reducidos, como se puede apreciar en los resultados mostrados en la sección 5.3.2.
- Uso de un *middleware* de comunicaciones moderno y eficiente para garantizar la interoperabilidad entre distintos lenguajes de programación, sistemas operativos, plataformas hardware y redes de comunicaciones.
- Propuesta basada en las especificaciones de FIPA para el desarrollo de SMA.
- Despliegue automático de los servicios de gestión de FIPA: *Agent Management System*, *Directory Facilitator* y *Message Transport System*.

- Desarrollo basado en patrones de diseño y en plantillas, para facilitar el despliegue de los componentes del SMA.
- Servicio de localización transparente a todos los elementos del SMA.
- Robustez gracias a los mecanismos de replicación automática proporcionados.
- Autenticación, encriptación e integridad de los mensajes gracias a la posibilidad de utilizar SSL como protocolo de transporte.
- Interfaz gráfica de administración para facilitar la gestión del SMA.

4.2. Visión general de la plataforma de agentes

Desde un punto de vista arquitectónico, la propuesta planteada está compuesta de un registro y una serie de nodos. A nivel administrativo, existe una coordinación entre el **registro** y los nodos para llevar a cabo la gestión de la información y los procesos que componen la aplicación distribuida. Básicamente, cada aplicación asigna una serie de servidores a los nodos. El registro es el encargado de mantener de manera persistente esta información mientras que los **nodos** se encargan de la activación y la monitorización de sus servidores asociados. De este modo, tanto los servicios FIPA como los agentes se despliegan en los nodos de la aplicación, acción que se realiza a partir de la configuración deseada por el usuario. Cada nodo se ejecuta en un dispositivo físico, el cual proporciona los recursos hardware necesarios para albergar los servidores, siendo posible el despliegue de más de un nodo en un mismo dispositivo físico. Del mismo modo, el desarrollador es el responsable de decidir qué dispositivos o máquinas utilizar a la hora de realizar el despliegue del sistema.

En la figura 4.1 se puede apreciar de manera gráfica la arquitectura de la propuesta, en la que se distinguen los conceptos de registro, nodo y host o dispositivo físico. De manera adicional a las recomendaciones de FIPA en relación a los servicios básicos (AMS, DF y MTS) y al agente básico de gestión, se incluye la *fábrica de agentes* para facilitar el despliegue y la administración de los distintos agentes que forman parte del sistema final. Este componente, el cual implementa el patrón de fábrica abstracta [GHJV95], facilita enormemente la instanciación de los agentes y la correcta gestión de los recursos asociados a los mismos. La funcionalidad de cada una de estas capas se resume a continuación:

- **Núcleo de ejecución del middleware**: esta capa hace referencia al núcleo de ejecución del middleware usado para dar soporte al SMA.
- **Núcleo de la plataforma**: esta capa contiene los componentes que dan soporte a la plataforma multi-agente y consiste en un *registro* y en un

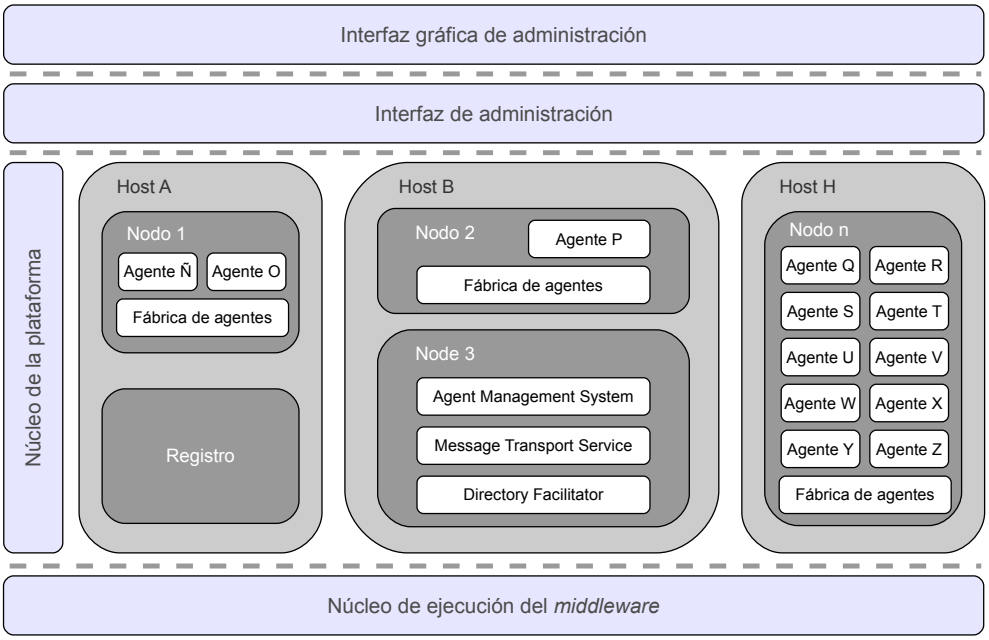


Figura 4.1: Arquitectura de la plataforma multi-agente de propósito general.

número indeterminado de *hosts*, responsables de alejar los nodos que albergarán los distintos elementos de la plataforma.

- **Interfaz de administración**: esta capa proporciona las interfaces de programación para realizar tareas administrativas a un nivel de abstracción medio.
- **Interfaz gráfica de administración**: esta capa permite interactuar y administrar el SMA desde una herramienta de más alto nivel, proporcionando un mayor nivel de abstracción a la hora de controlar y monitorizar el SMA.

4.3. El agente

Según FIPA [Fou04], el agente es la entidad de gestión básica de la plataforma de agentes y representa el proceso computacional que implementa las funciones y los mecanismos de comunicación necesarios para cumplir una determinada tarea.

Teniendo en cuenta la nomenclatura que se suele utilizar a la hora de utilizar un *middleware* de comunicaciones, este agente de gestión estará representado por un *proxy*, el cual actuará como el embajador de dicho agente, ofreciendo la interfaz pública del mismo y permitiendo que el núcleo de comunicaciones de ICE traslade las peticiones a la implementación física del

Agente FIPA	***Proxy***
name	identidad del objeto
addresses	direcciones físicas del *proxy*
resolvers	direcciones físicas del servicio de localización (registro)

Tabla 4.1: Relación entre los atributos de un agente FIPA y un *proxy* ICE.

agente. En este contexto, el *proxy* de un agente recoge la información de gestión básica del mismo y, al mismo tiempo, es posible establecer una relación entre dicha información y la especificada por FIPA para nombrar a un agente. Esta relación se puede apreciar en la tabla 4.1, donde se muestra la semántica utilizada por FIPA en la ontología *fipa-agent-management* [Fou04].

De este modo, el *proxy* encapsula toda la información asociada al agente: identificador único, direcciones de contacto para interactuar con él y las direcciones del servicio de localización para cualquier consulta que requiera la búsqueda de agentes o de otro servicio dentro de la plataforma de agentes. Esta decisión de diseño simplifica la plataforma debido a que se ha utilizado un elemento ya existente en el propio *middleware* de comunicaciones para representar e interactuar con los agentes.

Para soportar el **ciclo de vida** de los agentes se ha adoptado un enfoque basado en una máquina de estados, considerando las distintas transiciones existentes entre los estados definidos por FIPA. Una cuestión a tener en cuenta es la correcta gestión del acceso concurrente al estado de un agente. Por ejemplo, el AMS podría cambiar el estado de un agente mientras otro intente obtenerlo de manera simultánea. Para que no se produzca ningún problema de concurrencia se utiliza un monitor como mecanismo básico de exclusión mutua para este tipo de operaciones.

Otra característica relevante del agente es el soporte para la **persistencia**. De este modo, se permite que el agente mantenga su estado interno en caso de detener su ejecución con el objetivo de recuperlo cuando se vuelva a activar. También es posible establecer explícitamente qué datos del agente han de ser persistentes mediante el uso de metadatos junto con el código fuente de gestión asociado, el cual tiene un impacto mínimo en la aplicación.

Interfaz del agente de gestión FIPA

```
1  interface Agent {
2
3    void setState (AgentState st);
4    idempotent AgentState getState ();
5    ["ami"] void receiveMessage (Message m);
6    void dfAdv (DFAction act, DfAgentDescription desc);
7    void destroy ();
8
9  };
```

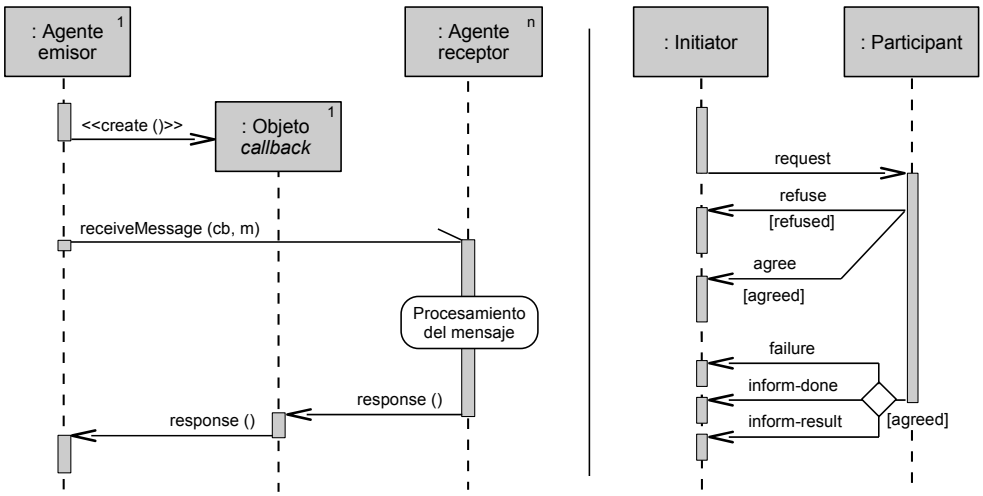


Figura 4.2: Protocolo de interacción *FIPA Request Interaction Protocol* (derecha) modelado a través de un objeto *callback* (izquierda).

Actualmente, el modelo de comunicación utilizado por los agentes está basado en el uso de invocaciones remotas mediante el protocolo de comunicación proporcionado por el middleware, el cual puede utilizar TCP/IP, UDP o SSL, y que ofrece un gran rendimiento y robustez [Hen09]. Sobre esta capa de comunicación se construyen los protocolos de interacción definidos por FIPA para que los agentes se comuniquen entre sí.

En este contexto, un agente puede recibir mensajes de manera síncrona y de manera asíncrona (ver metadatos *["ami"]* en la operación *receiveMessage* de la interfaz del agente). Gracias al modelo de **comunicación asíncrona**, el emisor de un mensaje no permanece bloqueado hasta que el receptor del mismo haya terminado su procesamiento. En realidad, dicho emisor se libera una vez que ha transferido el contenido del mensaje, sin la necesidad de esperar directamente a su gestión por parte del receptor, la cual puede llevar un cierto tiempo. La pregunta surge a la hora de plantearse cómo es posible que el emisor conozca la reacción del receptor después de haber procesado el mensaje. La solución a este problema son los objetos de retrollamada (*callback*), utilizados para notificar el resultado del envío de un mensaje por parte de un agente, ya sea para enviar información o para notificar que se produjo algún contratiempo en el procesamiento del mismo.

Este enfoque basado en objetos de retrollamada nos permite modelar fácilmente los protocolos de interacción de FIPA, debido a que éstos suelen incluir mensajes para aceptar solicitudes de un agente y mensajes para notificar el resultado de procesar dichas solicitudes. En la figura 4.2 se puede apreciar gráficamente cómo se ha modelado el protocolo de interacción *FIPA Request Interaction Protocol* mediante un objeto de retrollamada.

4.4. La fábrica de agentes

La fábrica de agentes es la responsable de la creación de agentes y se puede entender como el gestor de los sirvientes asociados a los mismos. En este contexto, un sirviente representa la implementación física de un agente en un determinado lenguaje de programación. Aunque FIPA no define de manera explícita a dicho componente, éste resulta interesante para gestionar de manera adecuada los recursos software utilizados por los agentes. Además, permite plantear un enfoque que garantice la escalabilidad del sistema cuando el número de agentes crece. Por ejemplo, no sería muy práctico tener una aplicación compuesta por miles de agentes almacenados en la memoria principal de la máquina o máquinas sobre las que se ejecutan, ya que es más razonable hacer uso de algún tipo de mecanismo que permita la activación de agentes bajo demanda, es decir, cuando realmente se les necesite y sin que todos estén en ejecución de manera simultánea.

La fábrica de agentes se ha diseñado con este objetivo. Para llevarlo a cabo, sólo almacena en memoria principal un determinado número de agentes configurable en función de los requisitos de la aplicación multi-agente, ofreciendo un mecanismo de activación de agentes cuando así sea necesario y una política de reemplazo de los mismos en memoria principal proporcionada por el middleware utilizado. Actualmente, el mecanismo utilizado para reemplazar agentes se basa en el enfoque *menos recientemente usado*, aunque es posible extender la fábrica de agentes con otro tipo de enfoques. En la sección 5.3.2 se muestran los resultados obtenidos al analizar el rendimiento de este componente a partir de un extenso conjunto de pruebas.

A continuación se muestra la interfaz ofrecida por la fábrica de agentes, la cual está compuesta por dos operaciones que permite la creación de los mismos de manera individual o colectiva:

Interfaz de la fábrica de agentes

```
1   interface Agent;
2   sequence<Agent*> AgentSeq;
3
4   exception AgentExists {};
5   exception AlreadySubscribed {};
6
7   interface AgentFactory {
8
9     Agent* create (string name) throws AgentExists;
10    AgentSeq createSeq
11       (Ice::StringSeq names,
12        out Ice::StringSeq namesAlreadyRegistered);
13
14  };
```

Un aspecto a discutir sobre esta interfaz es la no inclusión de operaciones para destruir agentes. Si la fábrica permite su instanciación, parece lógico

que también permita su destrucción. Sin embargo, esta decisión de diseño no es acertada debido a varias razones. La más relevante se justifica de la siguiente forma: si un componente del sistema ha de interactuar con un agente, entonces necesita mantener dos *proxies* o referencias de manera simultánea, el del propio agente y el de la fábrica de agentes que se utilizó para instanciarlo. Si se tiene en cuenta que puede ser necesario desplegar un alto número de agentes para determinadas aplicaciones de dominio, entonces se necesita un enfoque con más cohesión y menos propenso a errores.

La solución planteada en este trabajo pasa por incluir la funcionalidad de destrucción en el propio agente. Además de solventar esta cuestión de diseño, la ventaja de que el agente se encarga de esta funcionalidad le permite decidir de manera autónoma cómo actuar si alguien solicita su destrucción, sin tener que delegar dicha decisión a otros componentes externos (como a la fábrica que lo creó).

4.5. Servicios de gestión de FIPA

4.5.1. *Agent Management System*

El ***Agent Management System*** (AMS) es el componente FIPA encargado de gestionar el control y el acceso a la plataforma de agentes [Fou04]. Sólo se permite un AMS por plataforma, el cual es responsable de mantener un directorio con los agentes registrados en la misma. En otras palabras, el AMS proporciona un servicio de páginas blancas al resto de componentes del sistema.

A continuación se muestra la interfaz del AMS, la cual contiene operaciones para registrar y eliminar el registro de agentes, además de permitir su búsqueda por nombre y de acceder a la descripción de la plataforma de agentes.

Interfaz del *Agent Management System*

```
1    interface AMS {
2
3      void register (Agent* aid) throws AgentExists;
4      idempotent void deregister (Ice::Identity id);
5      idempotent Agent* search (Ice::Identity id);
6      idempotent ApDescription getDescription ();
7
8    };
```

Debido a que el AMS es un elemento central de la plataforma, su diseño ha de guiarse por la **robustez** y la **tolerancia a fallos**. Para alcanzar este objetivo, la solución planteada pasa por replicarlo mediante un esquema

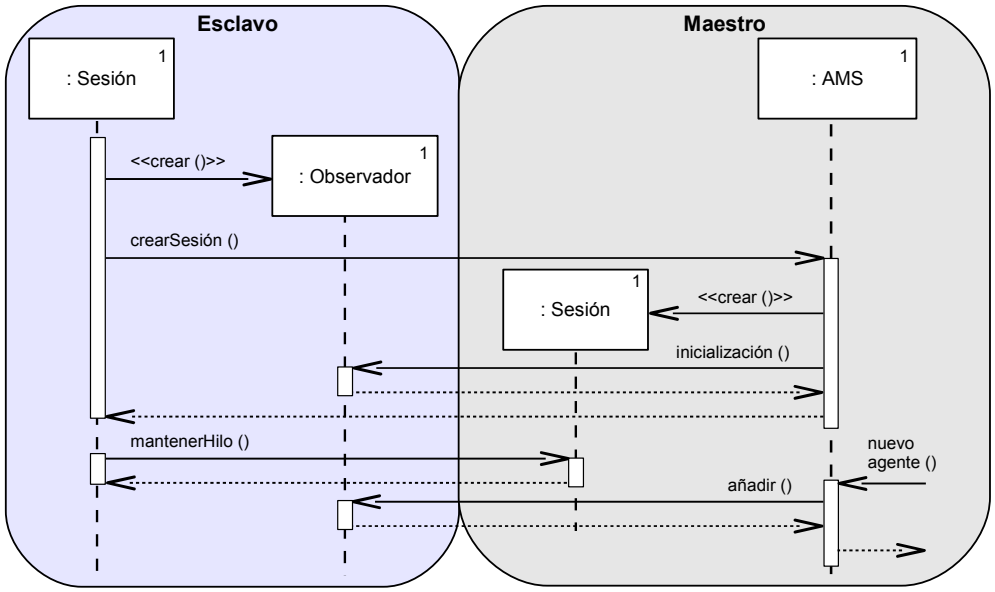


Figura 4.3: Mecanismo de replicación maestro-esclavo utilizado por el *Agent Management System* para garantizar la robustez del SMA.

maestro-esclavo, como se refleja en la figura 4.3. Básicamente, una de las réplicas del AMS es considerado el maestro y el resto esclavos. Cada esclavo hace uso del patrón observador [GHJV95] para establecer una conexión con el maestro, lo cual les permite observar el estado de este último. De este modo, si el AMS maestro actualiza su estado, es decir, el directorio de agentes, entonces envía notificaciones a los esclavos. Así mismo, cada esclavo mantiene un directorio interno que sólo se puede actualizar mediante las notificaciones del maestro. Si éste no es capaz de enviar notificaciones, entonces los esclavos tendrán que destruir y volver a establecer las sesiones con el maestro, con la finalidad de garantizar la sincronización entre los esclavos y el maestro.

En este contexto, otro aspecto a considerar es la promoción de un esclavo a maestro cuando este último sufre algún tipo de problema o simplemente la máquina en la que se ejecuta deja de funcionar correctamente. Dos posibles soluciones son seleccionar de manera aleatoria un esclavo para que éste promocione a maestro, o hacer uso de un esquema basado en prioridades, donde cada esclavo está personalizado con número que determina su prioridad a la hora de promocionar. Actualmente, el esquema utilizado es este último, de manera que cuando el maestro deja de funcionar, los esclavos se comunican entre sí para descubrir al que tiene una mayor prioridad y, por lo tanto, el que pasará a ser el nuevo maestro.

4.5.2. *Directory Facilitator*

El ***Directory Facilitator*** (DF) es un componente definido por FIPA que se puede incluir de manera opcional en la plataforma de agentes. Su principal responsabilidad es la de ofrecer un servicio de páginas amarillas y de proporcionar un servicio de suscripción a los agentes. Dicho servicio gestiona las notificaciones del movimiento (registro, modificación de los servicios ofrecidos, etc) de agentes dentro de la plataforma. Para poder hacer uso de estos servicios, un agente ha de registrarse con el DF. Así mismo, FIPA contempla la posibilidad de desplegar distintos DFs en un mismo sistema, pudiendo formar una jerarquía entre ellos.

El diseño del DF es idéntico al del AMS y también se basa es un esquema de replicación maestro-esclavo. Para extender la funcionalidad del DF, en este trabajo se ha utilizado un protocolo de descubrimiento de servicios [VVM+07] orientado a la escalabilidad y a su uso en entornos compuestos de sensores inalámbricos, minimizando al máximo la necesidad de configurar a los elementos que hacen uso del mismo. La justificación de esta extensión se basa en hacer posible la interacción con un alto número de componentes, ya sean agentes o no, que proporcionan servicios dentro de un determinado contexto.

La interfaz del DF se muestra a continuación y resume la funcionalidad previamente descrita.

Interfaz del *Directory Facilitator*

```
1   interface DF {
2
3     void register (DfAgentDescription desc)
4        throws AgentExists;
5     idempotent void deregister (Ice::Identity id);
6     void modify (DfAgentDescription desc);
7     idempotent AgentSeq search (DfAgentDescription desc);
8
9     void subscribe (Agent* aid) throws AlreadySubscribed;
10    idempotent void unsubscribe (Agent* aid);
11
12  };
```

4.5.3. *Message Transport Service*

Finalmente, el *Message Transport Service* proporciona la funcionalidad necesaria para efectuar el envío de mensajes entre agentes. Al igual que los otros dos componentes básicos de la plataforma, este servicio ha sido replicado para obtener una mayor robustez. No obstante, el principal motivo de esta replicación es la posibilidad de distribuir la carga de trabajo entre las distintas réplicas en dominios en los que el tráfico de mensajes sea elevado.

En la sección 5.3.1 se analizan los resultados obtenidos a la hora de evaluar el rendimiento de este componente en entornos con un alto número de mensajes.

Interfaz del *Message Transport Service*

```
1   interface MTS {
2
3     ["ami"] void receiveMessage (Message m);
4
5   };
```

4.6. Despliegue de sistemas multi-agente

El despliegue de la plataforma multi-agente se realiza a través de las siguientes operaciones:

1. Creación de los directorios necesarios para almacenar los datos persistentes.
2. Arranque de un nodo para soportar los servicios básicos de gestión de FIPA (AMS, DF y MTS) y el registro del sistema. Dichos servicios se activan bajo demanda cuando reciben alguna petición.
3. Arranque de un nodo para desplegar tres fábricas de agentes, cada una de ellas implementada en un lenguaje de programación diferente (C++, Java y Python).
4. Despliegue del descriptor de la aplicación, es decir, del archivo XML que especifica qué servicios se desplegarán, cuáles son sus propiedades y qué nodos se utilizarán para darles soporte. En la página siguiente se muestra un fragmento del descriptor utilizado para realizar las pruebas de rendimiento de la sección 5.3.1.

El conjunto de pasos previamente comentados representa la funcionalidad por defecto de la plataforma a la hora de desplegar una aplicación multi-agente. Sin embargo, es posible personalizar el proceso de despliegue en función de los requisitos de la aplicación de usuario y atendiendo a los siguientes criterios:

- Número de dispositivos hardware.
- Número de nodos (en el mismo o distintos dispositivos).
- Servidores a desplegar en cada nodo.

- Cuestiones técnicas como el número de hilos por servidor o el tamaño de la cola de agentes activos en cada fábrica de agentes.

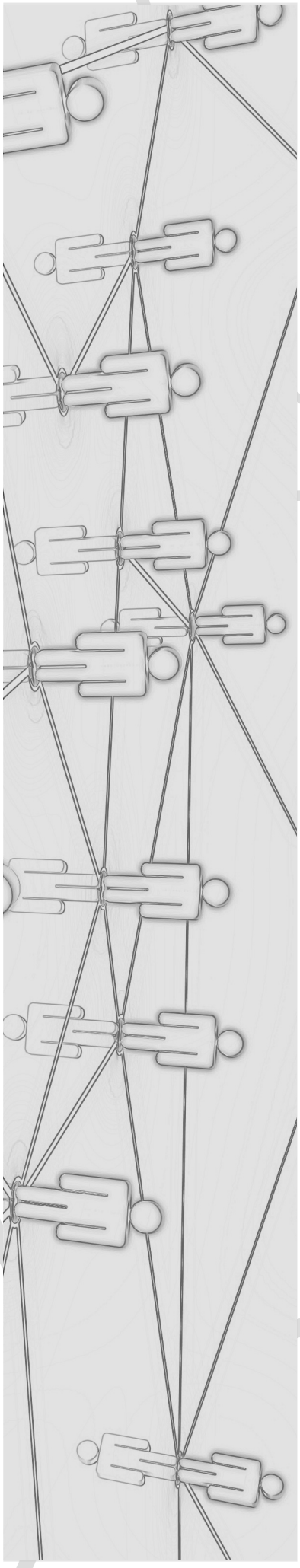
El uso de plantillas facilita tanto el despliegue como la configuración de la plataforma de agentes propuesta. De hecho, se han desarrollado plantillas de servicios y servidores para todos los elementos que forman parte de la arquitectura general con el objetivo de facilitar esta tarea a los desarrolladores finales. En el descriptor que se muestra a continuación se puede apreciar un ejemplo de cada uno de estos tipos de plantillas. Así mismo, en la sección 5.4.5 se detalla cómo realizar el despliegue de un sistema multi-agente concreto.

Ejemplo de descriptor para desplegar un SMA de evaluación

```
1
2   <icegrid>
3    <application name="FIPA">
4      <service-template id="MTSServiceTemplate">
5      <parameter name="id" default="MTSService.${server}"/>
6      <service name="${id}" entry="MTSServer:createMTSServer">
7       <properties>
8        <property name="adapter" value="${id}.Adapter"/>
9       </properties>
10      <adapter name="${id}.Adapter" endpoints="default"
11       id="${server}.${service}.${id}.Adapter"
12       replica-group="MTSServiceReplicaGroup"/>
13    ...
14    <server-template id="IceBoxMTSServerTemplate">
15     <parameter name="index" default="0"/>
16     <parameter name="instance-name"
17                default="${application}.IceBoxMTSServer"/>
18     <icebox id="${instance-name}.${index}"
19             activation="on-demand" exe="icebox">
20      <properties>
21       <property name="IceBox.InstanceName" value="${server}"/>
22       <property name="IceBox.ServiceManager.Endpoints"
23                 value="tcp -h 127.0.0.1"/>
24       <property name="Ice.ThreadPool.Server.Size" value="8"/>
25       <property name="Ice.ThreadPool.Client.Size" value="8"/>
26      </properties>
27       <service-instance template="MTSServiceTemplate"/>
28     </icebox>
29    </server-template>
30    ...
31    <node name="FIPAServices">
32     <server-instance template="IceBoxFIPAServerTemplate"/>
33     <server-instance template="IceBox"/>
34    </node>
35    <node name="Node1">
36     <server-instance template="IceBoxAgentFactoryTemplate"
37                      index="0"/>
38     <server-instance template="IceBoxMTSServerTemplate"/>
39     <server-instance template="SnifferServerTemplate"/>
40     <server-instance template="SpammerFactoryServerTemplate"
41                      index="0"/>
42    </node>
43    </application>
44  </icegrid>
```

Capítulo 5

Resultados



Capítulo 5
Resultados

En este capítulo se discute la evaluación de la arquitectura de vigilancia inteligente propuesta mediante el despliegue de un sistema de monitorización en un entorno real de tráfico urbano. En dicho capítulo se discute un caso de estudio que consiste en la integración de un agente de análisis de normalidad dentro de la arquitectura. Así mismo, también se presente un extenso conjunto de tests de rendimiento realizados sobre la plataforma multi-agente de propósito general, finalizando con el despliegue de un sistema multi-agente en el ámbito de la coordinación multi-robot.

"Movimiento es el paso de la potencia al acto."
Aristóteles (Siglo IV a.C.)

5
Resultados

En este capítulo se presentan los resultados obtenidos como consecuencia de desplegar un sistema de vigilancia inteligente en un entorno de monitorización concreto y de evaluar la plataforma de agentes propuesta en la presente tesis.

La sección 5.1 discute la solución particular desarrollada para vigilar un escenario de tráfico urbano en el que circulan tanto vehículos como peatones con el objetivo de ilustrar el enfoque propuesto en el capítulo 3. El prototipo utilizado para verificar la funcionalidad de la arquitectura propuesta para la vigilancia inteligente ha sido desarrollado haciendo uso del *middleware* ZeroC ICE [Hen04b]. Dicho prototipo se ha utilizado para realizar distintas pruebas de evaluación y rendimiento en el escenario de tráfico monitorizado (para más detalle consultar el anexo B.1).

La sección 5.2 analiza la integración de un agente de normalidad de análisis de trayectorias en un sistema de vigilancia particular (el desplegado para monitorizar el escenario de la sección 5.1). Este proceso cubre el diseño del propio concepto de normalidad, la definición de los tipos de datos necesarios y el despliegue del agente en el sistema.

A continuación, la sección 5.3 discute en profundidad la evaluación de la plataforma de agentes, estableciendo una comparación con JADE (la plataforma de agentes más utilizada actualmente). Dicha evaluación se puede extrapolar a la arquitectura multi-agente para la vigilancia inteligente, justificando la propuesta desarrollada en términos de eficiencia, escalabilidad y robustez.

Finalmente, las secciones 5.4 y 5.5 analizan el uso de la plataforma de agentes propuesta como soporte en el desarrollo, despliegue y administración de aplicación de la tecnología de agentes en otros dominios de aplicación.

5.1. Despliegue de un sistema concreto de vigilancia inteligente

5.1.1. Descripción del entorno de monitorización

La figura 5.1 muestra el escenario elegido para desplegar un sistema de vigilancia inteligente diseñado de acuerdo al modelo de análisis de normalidad descrito en la sección 3.3. Dicho escenario consiste en una intersección regulada por una glorieta en la que los vehículos que se encuentran circulando en ella tienen prioridad sobre los que pretenden incorporarse. Todos los carriles que entran y salen de la glorieta son de un único sentido, de manera que cada rama de la intersección tiene un carril de entrada y otro de salida. En este caso particular, el escenario global (ver figura 5.1.a) se ha dividido en tres sub-entornos o percepciones: E_1 (ver figura 5.1.b), E_2 (ver figura 5.1.c) y E_3 (ver figura 5.1.d). La posición de las cámaras viene determinada por aquellos lugares que ofrecen mejor visibilidad y son de fácil acceso desde el laboratorio de investigación ORETO de la Universidad de Castilla-La Mancha.

Figura 5.1: Escenario de tráfico urbano monitorizado. **a)** Visión global del escenario. **b)** Sub-entorno E_1 monitorizado desde la cámara s_1. **c)** Sub-entorno E_2 monitorizado desde la cámara s_2. **b)** Sub-entorno E_3 monitorizado desde la cámara s_3.

Actualmente es interesante realizar la vigilancia del escenario de la figura 5.1 mediante el análisis de **dos componentes** o aspectos independientes. Uno de ellos es el de trayectorias de objetos móviles, denotado como c_1, y el otro es el análisis de velocidad de dichos objetos, denotado como c_2. Para cubrir estas necesidades de vigilancia, estos dos componentes se han diseñado de manera independiente haciendo uso de las definiciones proporcionadas por el modelo de normalidad. Posteriormente, será posible instanciarlos en cada uno de los entornos (E_1, E_2 y E_3) en los que se ha dividido el escenario global, hecho que implicará el despliegue de los distintos agentes de vigilancia asociados.

Por una parte, con el **análisis de trayectorias** se monitoriza la normalidad de los objetos móviles en la escena (viandantes y vehículos) cuando se desplazan en la misma. El componente de normalidad de trayectorias se detalla en [AVJL+09], donde se discute en detalle las decisiones de diseño del mismo y las restricciones definidas para expresar la normalidad de dicho componente.

Por ejemplo, en el caso de los vehículos el comportamiento normal implica la definición de las posibles circulaciones que puede efectuar el objeto monitorizado cuando se va aproximando a la glorieta. Para ello se hace uso de una herramienta de adquisición de conocimiento que facilita cuestiones como la división en zonas del sub-entorno en el que se instancia el componente, como se muestra en la figura 5.2.

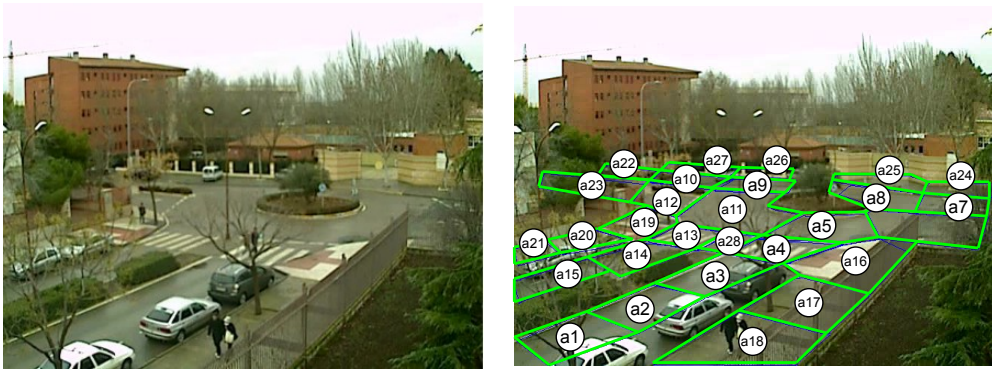
Básicamente, estas circulaciones normales tienen en cuenta la entrada de un vehículo en la escena y la salida por uno de los carriles (derecha, izquierda y cambio de sentido) siempre en el sentido correcto. No obstante, la definición de normalidad para las trayectorias ha de tener en cuenta más cuestiones, además de las zonas por las que va pasando un vehículo (restricciones espaciales). Dicha normalidad ha de contemplar que las circulaciones se realizan en un cierto intervalo de tiempo (restricciones temporales) o que los vehículos tienen como objetivo alcanzar un destino (restricciones de destino) y no se dedican a merodear por el escenario.

Por otra parte, con el componente de **análisis de velocidad** de objetos móviles se pretende detectar, especialmente, a aquellos objetos móviles, generalmente vehículos, que se desplacen demasiado rápido en el entorno y, por lo tanto, infrinjan los límites de velocidad impuestos en el escenario de tráfico urbano monitorizado.

En la tabla 5.1 se resumen las instancias de los componentes de normalidad c_1 y c_2 que se desplegarán en cada uno de los sub-entornos E_1, E_2 y

Concepto / Sub-entorno	E_1	E_2	E_3
Trayectorias (c_1)	c^1_{1y}	c^2_{1y}	c^3_{1y}
Velocidad (c_2)		c^2_{2y}	c^3_{2y}

Tabla 5.1: Instancias desplegadas a partir de los conceptos de normalidad de trayectorias (c_1) y análisis de velocidad (c_2) monitorizados en E_1, E_2 y E_3.



(a) (b)

Figura 5.2: a) Imagen tomada desde la cámara que monitoriza el sub-entorno E_2. **b)** Conjunto de zonas relevantes definidas en E_2 por un usuario experto.

E_3. El despliegue de dichas instancias en los sub-entornos que componen el escenario de tráfico determinará la arquitectura concreta del sistema de vigilancia inteligente para el caso práctico tratado en esta sección. Esta arquitectura particular de agentes, junto con los mecanismos de comunicación necesarios para proporcionar una vigilancia global, es objeto de estudio en el siguiente apartado.

5.1.2. Discusión del sistema multi-agente desplegado

La figura 5.3 muestra de manera gráfica la arquitectura concreta del sistema multi-agente de vigilancia inteligente desplegado para monitorizar trayectorias y velocidades en el escenario de tráfico urbano de la figura 5.1. El proceso de vigilancia inteligente se ha realizado dividiendo dicho escenario en tres sub-entornos o percepciones distintas (E_1, E_2 y E_3), lo cual simplifica el análisis global del mismo y permite la composición de los estudios de las distintas vistas parciales del entorno. Este proceso ha sido posible gracias a las funcionalidades proporcionadas por el conjunto de agentes que a continuación se detallan y a los mecanismos de despliegue y comunicación soportados por la arquitectura propuesta en este trabajo.

En el **nivel reactivo** se despliegan los agentes pm, sm y cm, responsables de proporcionar servicios básicos (de acuerdo a las especificaciones propuestas por FIPA) al resto de agentes de la plataforma, independientes de la división del escenario en sub-entornos ya que realizan tareas de propósito general. En el sub-entorno E_1 se ha instalado una cámara de seguridad, identificada en la arquitectura como el sensor s_1. La salida de dicho dispositivo es procesada por el agente de preprocesamiento pa_1, el cual recibe la información visual mediante el *middleware* de comunicaciones transversal a todos los niveles del sistema de vigilancia. Para enviar la información al nivel deliberativo se ha desplegado un canal de eventos en el que pa_1 irá pu-

blicando la información asociada al análisis de los *frames* de vídeo (objetos detectados, posiciones de los mismos y clasificación con cierta incertidumbre). En los sub-entornos E_2 y E_3 se despliegan las cámaras s_2 y s_3 y los agentes de preprocesamiento pa_2 y pa_3, respectivamente. De manera análoga a E_1, la comunicación con el nivel deliberativo se materializa con otros dos canales de eventos, uno por cada sub-entorno.

En el **nivel deliberativo**, los primeros agentes encargados de la vigilancia del escenario son los que efectúan el análisis a nivel de sub-entorno. En el sub-entorno E_1 se despliega el agente de normalidad na_1, cuyo modelo de conocimiento está representado por la instancia c^1_{1y} del concepto de trayectorias y cuya entrada de información es el canal de eventos desplegado en E_1. En este sub-entorno sólo se lleva a cabo el análisis de este componente debido a que el análisis de velocidad se realizará de manera más general en el sub-entorno E_2. El conocimiento generado por na_1 se escribe en la pizarra desplegada en E_1 y sobre la que leerá el agente de normalidad a nivel de sub-entorno ena_1. En este sub-entorno en particular, ena_1 sólo tiene en cuenta el análisis de normalidad para el componente de trayectorias (c_1). Por otra parte, en E_1 se ha desplegado el agente de anormalidad aa_1, que recibirá información directa de na_1 si éste detectó que un objeto móvil no se comporta correctamente de acuerdo a las restricciones de c^1_{1y}.

En E_2, además de desplegar el agente de normalidad na_2 que maneja la base de conocimiento c^2_{1y} utilizada para vigilar las trayectorias respecto a la glorieta, se ha desplegado na_3 con el objetivo de monitorizar la velocidad de los objetos móviles detectados. En este caso, la base de conocimiento de na_3 (c^2_{2y}) representa la instancia del componente de análisis de velocidad para el sub-entorno E_2. Estas bases de conocimiento (como las del resto de agentes de normalidad de este sistema) se han generado con una herramienta de adquisición de conocimiento [AVJL+09], en el caso de las trayectorias, y con un algoritmo de aprendizaje automático [ACSLL+09], en el caso de la velocidad, integradas en el nivel de usuario. La comunicación entre el agente de normalidad y estas herramientas se realiza mediante el *middleware* de comunicaciones.

Al igual que ocurría en E_1, en este sub-entorno se despliega una pizarra en la que na_2 y na_3 irán escribiendo el conocimiento generado como consecuencia de monitorizar E_2 cada vez que reciban información del agente de preprocesamiento pa_2. Este conocimiento será utilizado por ena_2 para estudiar la normalidad de cualquier objeto detectado en E_2 como agregación de la normalidad analizada de manera independiente por na_2 y na_3. En cuanto al sub-entorno E_3, el análisis de normalidad es el mismo que en E_2 y sus encargados son los agentes de normalidad na_4 y na_5 y el agente de normalidad a nivel de sub-entorno ena_3.

En este punto, los agentes ena_1, ena_2 y ena_3 gestionan el conocimiento asociado al análisis de normalidad de acuerdo a los sub-entornos E_1, E_2 y E_3, respectivamente, el cual se envía al módulo de análisis global de norma-

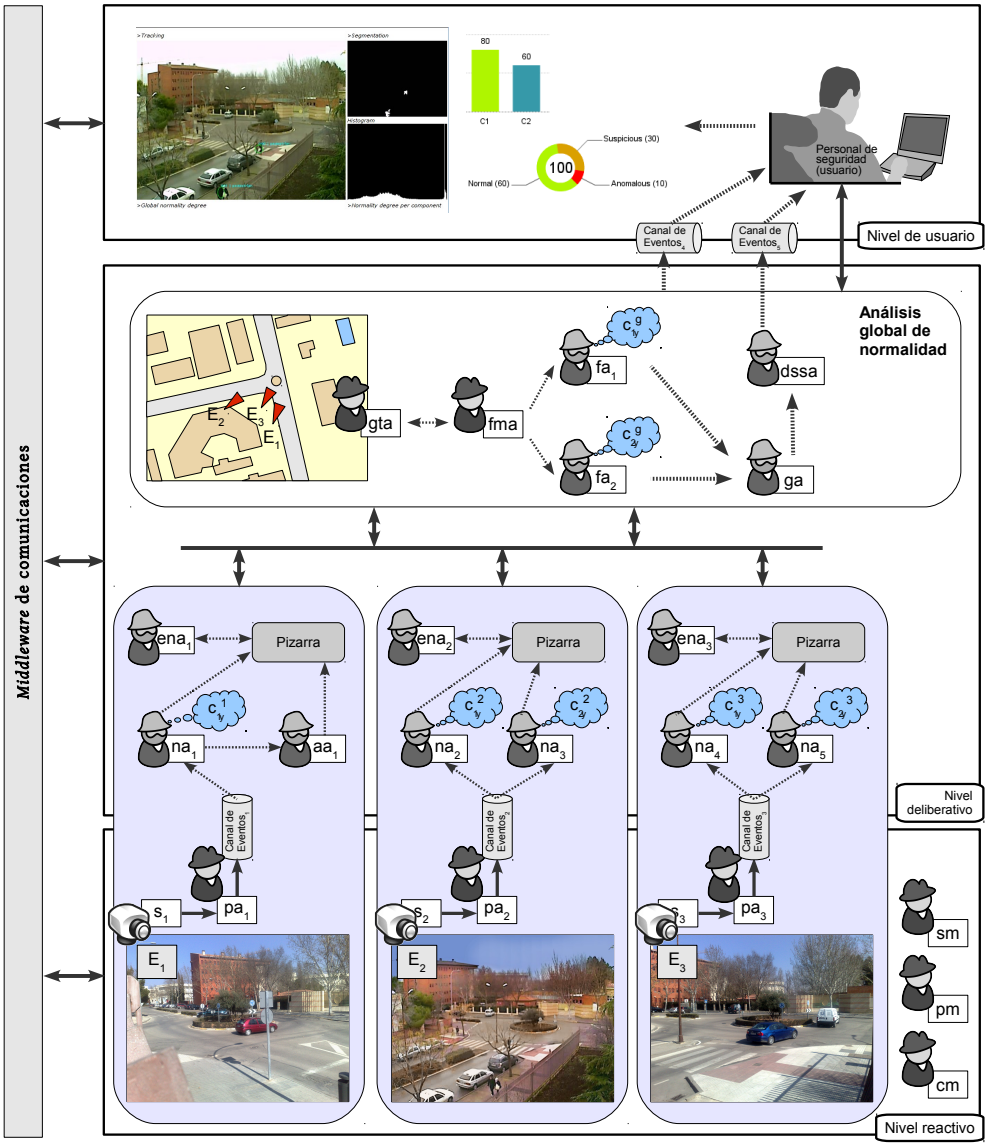


Figura 5.3: Arquitectura concreta del sistema de vigilancia desplegado para monitorizar el entorno de la figura 5.1.

lidad. La interfaz de este módulo está representada por el agente de gestión de fusión fma, el cual decide si es necesaria la fusión de información asociada a algunos de los conceptos c_1 o c_2 analizados en E_1, E_2 y E_3. En el sistema de tráfico urbano se han instanciado dos agentes de fusión fa_1 y fa_2, responsables de analizar la normalidad de los conceptos de trayectorias c_1 y velocidad c_2 desde un punto de vista global. La información global de los vehículos y los peatones la proporciona el agente de traducción ta, permitiendo identificar de manera global, por ejemplo, un peatón que esté cruzando el paso de peatones monitorizado simultáneamente en los sub-entornos E_2 y E_3. Para compartir este conocimiento se hace uso de la pizarra en el módulo de análisis de globalidad.

Finalmente, el agente de fusión global ga se ha instanciado con el objetivo de reunir el conocimiento obtenido por los agentes de fusión fa_1 y fa_2. En este sistema de vigilancia particular, ga sólo recoge las conclusiones de dichos agentes mediante la pizarra y las notifica al agente de apoyo a la toma de decisiones $dssa$. Este último agente utiliza el canal de eventos desplegado entre los niveles deliberativo y de usuario para informar al personal de seguridad. Actualmente, $dssa$ sólo activa alarmas en la herramienta de monitorización suscrita a dicho canal (ver figura 5.3).

5.1.3. Flujo de información

Para ejemplificar cómo fluye la información entre los distintos componentes del sistema multi-agente desplegado para monitorizar el escenario de la figura 5.1, a continuación se estudiarán en detalle algunas de las interacciones existentes entre los agentes del sub-entorno E_2 (ver figura 5.3).

Inicialmente, en cada instante de tiempo t_i, el agente de preprocesamiento pa_2 recibe una o más imágenes (*frames*) capturadas por la cámara de seguridad representada por el sensor s_2. En el caso particular del sistema analizado en esta sección, este agente de preprocesamiento lleva a cabo las siguientes tareas[1]: i) obtención de **parámetros básicos** para el análisis de la imagen, principalmente vinculados a los objetos móviles detectados en la misma, ii) **identificación de la zona** o zonas en las que se encuentran cada uno de los objetos móviles detectados, y iii) **clasificación** de los objetos móviles en función de sus caracteristicas físicas en una o varias clases (peatón, vehículo, etc). Esta información se notificará posteriormente a los agentes de normalidad na_2 y na_3 mediante el canal de eventos desplegado en E_2.

La obtención de parámetros básicos por parte de pa_2 implica obtener la siguiente información a partir de la salida proporcionada por s_2 por cada uno de los objetos móviles detectados:

[1]El objetivo de esta sub-sección no consiste en detallar cómo se realizan estas tareas, sino en dar un ejemplo de las entradas y salidas de cada uno de los agentes involucrados en las mismas.

Cámara	Frame	ID	x	y	Anchura	Altura	hd	vd	gd
1	148	obj0	429	348	19	34	0	0	0
1	148	obj2	230	260	44	25	0	0	0
1	148	obj3	272	309	10	11	0	1	1
1	149	obj0	429	348	19	34	0	0	0
1	149	obj2	230	260	44	25	0	0	0
1	149	obj3	272	310	10	11	0	0	0
1	150	obj0	429	348	19	33	0	0	0
1	150	obj2	231	261	45	25	0	0	0
1	150	obj3	272	310	10	11	0	0	0

Tabla 5.2: Información básica utilizada por el agente de preprocesamiento pa_2 para la identificación objetos en zonas físicas y clasificación de los mismos. *ID* representa el identificador asociado al objeto móvil detectado; *hd*, *vd* y *gd* representan los desplazamientos horizontales, verticales y global, respectivamente, de dicho objeto en el sub-entorno.

- Identificador del *frame* analizado.
- Posición del objeto en los ejes x e y, en función de la resolución de s_2.
- Anchura y altura del objeto.
- Desplazamiento horizontal, vertical y absoluto del objeto respecto a posiciones anteriores. Esta información será utilizada posteriormente por el agente na_3 de análisis de velocidad.

Definición de un evento con el IDL Slice

```
1   struct Zone {
2     string zone; float belief;
3   };
4   sequence<Zone> ZoneSeq;
5
6   struct Role {
7     string actor; float belief;
8   };
9   sequence<Role> RoleSeq;
10
11  struct Actor {
12    string id;
13    ZoneSeq zones;
14    RoleSeq roles;
15  };
16  sequence<Actor> ActorSeq;
17
18  struct Frame {
19    int number;
20    ActorSeq actors;
21  };
22  sequence<Frame> FrameSeq;
23
24  struct VideoEvent
25  {
26    int camera;
27    FrameSeq frames;
28  };
```

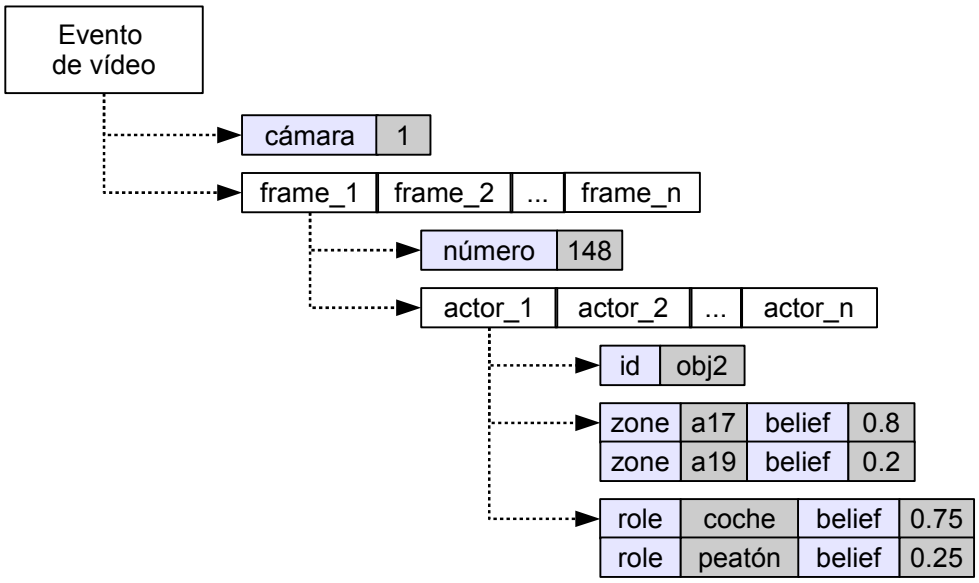


Figura 5.4: Descripción gráfica de un evento de vídeo.

En la tabla 5.2 se muestra la información básica generada por el agente de preprocesamiento pa_2 como resultado de analizar tres *frames* de vídeo consecutivos.

Con la información de posicionamiento de cada objeto y la definición física de las zonas del sub-entorno E_2, previamente realizada mediante una herramienta de adquisición de conocimiento a través de una interfaz gráfica (ver figura 5.2.b), pa_2 es capaz de establecer en qué zona o zonas se encuentran cada uno de los objetos. Es posible que un objeto se encuentre en más de una zona de manera simultánea. Un ejemplo podría ser un vehículo que tiene su parte delantera en el paso de peatones y su parte trasera en la calzada.

Así mismo, pa_2 es el responsable de la clasificación de objetos, utilizando para ello el algoritmo de aprendizaje desarrollado en [ACSLL+09], siendo capaz de clasificar un objeto móvil en una o más clases. La información obtenida tras realizar estas tres tareas es encapsulada por pa_2 en un **evento** (ver figura 5.4) que se envía al nivel deliberativo mediante un canal de eventos al que los agentes de normalidad na_2 y na_3 están suscritos.

Como se puede apreciar en la figura 5.3, los eventos de vídeo del canal del sub-entorno E_2 se notifican tanto a na_2 (análisis de trayectorias) como a na_3 (análisis de velocidad). En la tabla 5.3 se muestra el estado interno de na_2 en relación al análisis de las restricciones de normalidad que definen el concepto de **trayectorias**. Cada entrada de esta tabla representa la asociación de una trayectoria a un objeto móvil en un determinado *frame*, siendo especialmente relevantes los campos *RC*, *SC*, *TC* y *NV*. Los tres primeros representan el grado de satisfacción de normalidad de las restricciones de clase, espaciales

y temporales vinculadas a la circulación asociada al objeto. El último, es decir, *NV*, representa el grado de normalidad de dicha circulación, calculado en este caso como el mínimo de los valores de las restricciones previamente comentadas (en la sección 3.3 se discute el modelo de normalidad en el que se basa el motor de inferencia de na_2).

Por otra parte, en la tabla 5.4 se muestra el estado interno de $na3$ respecto al análisis de normalidad del componente de **velocidad**. En este caso, *NV* representa el grado de normalidad de la velocidad de cada uno de los objetos monitorizados en un determinado *frame*.

Finalmente, en la tabla 5.5 se resume el estado de la **pizarra** desplegada en el sub-entorno E_2 para compartir el conocimiento asociado al análisis de normalidad de las trayectorias y velocidades por parte de los agentes na_2 y na_3, respectivamente. La información escrita por estos agentes la usa el agente de análisis de normalidad ena_2 a nivel de sub-entorno para calcular el valor global de normalidad en cada uno de los instantes analizados.

Cámara	Frame	ID	Trayectoria	RC	SC	TC	NV
1	148	obj0	c8	0.881	1	1	0.881
1	148	obj0	c10	0.881	1	1	0.881
1	148	obj0	c13	0.881	0.987	1	0.881
1	148	obj2	c23	1	0.768	1	0.768
1	148	obj2	c25	1	0.304	1	0.304
1	148	obj2	c15	1	0.768	1	0.768
1	148	obj2	c22	1	0.652	1	0.652
1	148	obj3	c9	0.958	0.833	1	0.833
1	148	obj3	c12	0.958	0.916	1	0.916
1	148	obj3	c20	0	1	1	0
1	148	obj3	c16	0.958	0.916	1	0.916
1	149	obj0	c8	0.882	1	1	0.882
1	149	obj0	c10	0.882	1	1	0.882
1	149	obj0	c13	0.882	0.987	1	0.882
1	149	obj2	c23	1	0.768	1	0.768
1	149	obj2	c22	1	0.652	1	0.652
1	149	obj2	c25	1	0.304	1	0.304
1	149	obj2	c15	1	0.768	1	0.768
1	149	obj3	c9	0.934	0.833	1	0.833
1	149	obj3	c12	0.934	0.916	1	0.916
1	149	obj3	c20	0	1	1	0
1	149	obj3	c16	0.934	0.916	1	0.916

Tabla 5.3: Estado interno del agente de análisis de trayectorias na_2 en relación al análisis del *frame* 148. *ID* representa el identificador del objeto móvil; *RC*, *SC* y *TC* representan los grados de satisfacción de las restriciones de clase, espaciales y temporales, respectivamente; *NV* representa el grado de satisfacción de la trayectoria vinculada al objeto.

Cámara	Frame	ID	NV
1	148	obj0	1
1	148	obj2	1
1	148	obj3	1
1	149	obj0	1
1	149	obj2	1
1	149	obj3	1
1	150	obj0	1
1	150	obj2	1
1	150	obj3	1

Tabla 5.4: Estado interno del agente de análisis de velocidad na_3 en relación al análisis de los *frames* 148-150. *ID* representa el identificador del objeto móvil; *NV* representa el grado de satisfacción del componente de velocidad en relación al objeto.

Cámara	Frame	ID	C1	C2	Global
1	148	obj4	1	1	1
1	148	obj3	0.916	1	0.933
1	148	obj2	0.768	1	0.814
1	148	obj0	0.881	1	0.904
1	149	obj4	1	1	1
1	149	obj3	0.916	1	0.933
1	149	obj2	0.768	1	0.814
1	149	obj0	0.882	1	0.905
1	150	obj4	1	1	1
1	150	obj3	0.912	1	0.929
1	150	obj2	0.783	1	0.826
1	150	obj0	0.883	1	0.906

Tabla 5.5: Estado de la pizarra compartida por los agentes de normalidad na_2 (análisis de trayectorias) y na_3 (análisis de velocidad) y el agente de análisis de normalidad a nivel de sub-entorno (E_2) ena_2 durante los *frames* 148-150. *ID* representa el identificador del objeto móvil; *C1* representa el grado de satisfacción del componente de trayectorias; *C2* representa el grado de satisfacción del componente de velocidad; *Global* indica el valor global de normalidad calculado por ena_2 en función de los valores de *C1* y *C2*.

5.1.4. Uso del prototipo desarrollado para monitorizar un sub-entorno

En esta sección se discute el sistema de vigilancia particular desplegado mediante el prototipo de arquitectura desarrollado para monitorizar el sub-entorno E_2 (ver figura 5.1). Como se describió en la sección 5.1.2, en dicho sub-entorno se monitorizan dos aspectos: i) las trayectorias y ii) la velocidad de los objetos móviles. A continuación se resumen brevemente las tecnologías empleadas para la implementación del prototipo de arquitectura (ver figura 5.5):

- El *middleware* **ZeroC ICE** se ha utilizado para soportar toda la infraestructura de la arquitectura. En concreto, se han utilizado los servicios

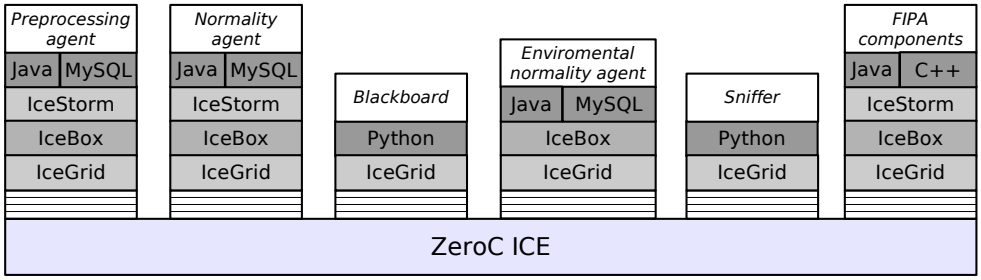


Figura 5.5: a) Tecnologías empleadas para la implementación del prototipo de arquitectura propuesta para la vigilancia inteligente.

IceGrid, para configurar, gestionar y desplegar el sistema; *IceBox*, para gestionar los servicios; *IceStorm*, para desplegar los canales de eventos; e *IceSSL* para utilizar el protocolo SSL.

- El motor de bases de datos relacional **MySQL** para almacenar toda la información generada por los agentes de vigilancia y registrar el tiempo empleado en cada una de las tareas.
- El lenguaje de programación **Java** para la implementación de pa_2, na_2 y na_3.
- El lenguaje de programación **Python** para la implementación del sistema de pizarra y del servicio *Sniffer*, encargado de almacenar el registro de tiempos del sistema.
- El lenguaje de programación **C++** para la implementación de los servicios FIPA.

En dicho sistema, el agente de preprocesamiento pa_2, los agentes de análisis de normalidad na_2 y na_3, y el agente de análisis de normalidad a nivel de sub-entorno ena_2 han sido replicados con un doble objetivo: i) asegurar la robustez del sistema ante posibles fallos y i) distribuir la carga de trabajo del proceso de vigilancia. En la parte izquierda de la figura 5.6 se muestra el diseño de este sistema, el cual se compone de tres nodos distintos denominados *Oreto1*, *Oreto2* y *Oreto3*. Cada uno de estos nodos alberga los agentes de vigilancia encargados de la monitorización del escenario y gestiona los recursos físicos necesarios para ellos. Esta configuración permite desplegar estos nodos en distintas máquinas físicas de manera transparente a los distintos componentes que forman parte de dicha arquitectura específica.

Por ejemplo, el nodo *Oreto1* alberga los siguientes componentes:

- La pizarra *Blackboard.4.1* utilizada por los agentes de análisis de normalidad en E_2.
- El servicio *Oculus.IceStorm* utilizado para desplegar los canales de eventos utilizados en E_2, proporcionado por el *middleware* ZeroC ICE.

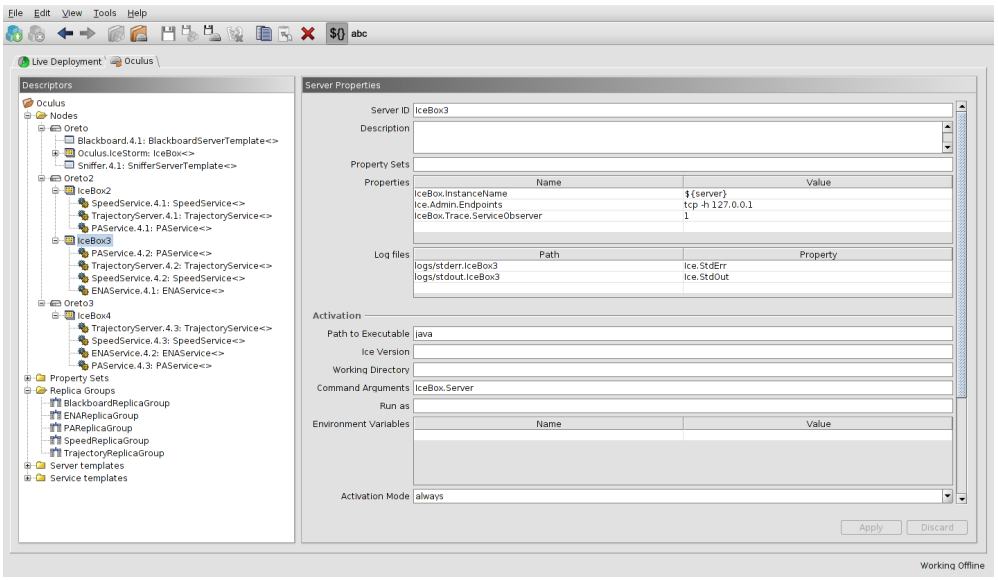


Figura 5.6: a) Diseño del sistema de vigilancia desplegado para monitorizar el sub-entorno E_2.

- El servicio *Sniffer.4.1* utilizado para almacenar los tiempos de comunicación y procesamiento empleados por los distintos agentes desplegados.

Por otra parte, el nodo *Oreto2* alberga los siguientes componentes:

- Dos réplicas, $PAService4{,}1$ y $PAService4{,}2$, del agente de preprocesamiento pa_2.
- Dos réplicas, $TrajectoryService4{,}1$ y $TrajectoryService4{,}2$, del agente de análisis de normalidad de trayectorias na_2.
- Dos réplicas, $SpeedService4{,}1$ y $SpeedService4{,}2$, del agente de análisis de normalidad de velocidad na_3.
- Una réplica, $ENAService{,}4{,}1$, del agente de análisis de normalidad ena_2 a nivel de sub-entorno E_2.

Este nodo mantiene más réplicas con el objetivo de desplegarlo en máquinas potentes en caso de que las demandas de procesamiento del entorno a analizar crezcan considerablemente. Finalmente, *Oreto3* mantiene sendas réplicas de pa_2, na_2, na_3 y ena_2 para incrementar la robustez del sistema. La política de replicación empleada en los tests desarrollados para distribuir el análisis de los eventos ha sido *aleatoria*. De este modo, el procesamiento se irá distribuyendo entre las réplicas de cada agente de manera uniforme a lo largo del tiempo.

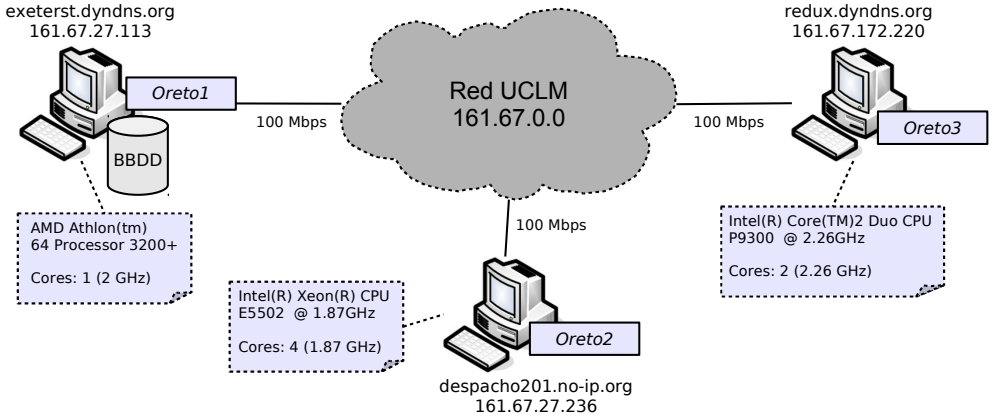


Figura 5.7: Recursos utilizados para desplegar y evaluar el prototipo de arquitectura desarrollado para monitorizar E_2.

Para evaluar la implementación de la arquitectura propuesta en esta tesis se han utilizado los recursos físicos que se muestran en la figura 5.7. Además de demostrar la robustez, la escalabilidad y la disponibilidad de la arquitectura, el conjunto de experimentos desarrollado ha permitido medir los tiempos que se utilizan en las diversas tareas del sistema de vigilancia, como por ejemplo el despliegue, los tiempos de comunicaciones o los tiempos de procesamiento. Así mismo, se pretende establecer una comparativa entre los tiempos empleados cuando el sistema de vigilancia del sub-entorno E_2 se despliega en una única máquina o en varias de acuerdo a la configuración de la figura 5.7.

Las pruebas realizadas se han realizado bajo los siguientes supuestos prácticos:

- La escena analizada tiene una duración de 2 minutos y 33 segundos. Dicha escena tiene un total de 3830 *frames* y fue grabada con una tasa de 25 *frames* por segundo.
- En términos de monitorización se analizaron 3500 *frames* en los que se detectaron objetos móviles de los 3830 *frames* de los que se compone el vídeo.
- Se ha desarrollado un cliente cuya misión es la de enviar los eventos de vigilancia con una determinada tasa, denominada como *tasa de eventos por segundo*. En el conjunto de pruebas desarrollado, cada evento representa la información de un *frame* capturado por la cámara del entorno E_2. Para cada test, esta variable tomará un valor del conjunto 10, 15, 20 ó 25.
- En cada test se obtendrán los tiempos de procesamiento y de comunicación de cada uno de los elementos que conforman el sistema de vigilancia desplegado.

- Se plantean tres tipos de conjuntos de pruebas: i) utilizando una única máquina (*redux.dyndns.org*, ver figura 5.7) sin replicar los agentes; ii) utilizando una única máquina (*redux.dyndns.org*) y replicando los agentes; y iii) utilizando la arquitectura de la figura 5.7 y replicando los agentes.

La tabla 5.6 presenta el resumen de tiempos del conjunto de tests realizados sobre la escena de prueba variando la configuración que determina el despliegue del sistema de vigilancia y la tasa de eventos por segundo (10, 15, 20 y 25) procesada por el mismo. Los tests 1 a 4 comprenden los resultados obtenidos sobre una máquina sin replicación de agentes (tipo **L**), los tests 5 a 8 sobre una máquina con replicación de agentes (tipo **LR**), y los tests 9 a 12 sobre la red de la figura 5.7 (tipo **D**).

En dicha tabla, el segundo bloque de columnas representa los tiempos expresados en segundos empleados por los agentes de preprocesamiento para analizar (PA_{TP}) los 3500 *frames* y comunicarse con los agentes de normalidad (TA_{TC} y SA_{TC}, respectivamente); y los tiempos del procesamiento asociado al análisis de normalidad de trayectorias (TA_{TP}) y de velocidad (SA_{TP}). El tercer bloque de columnas representa los tiempos de escritura en la pizarra por los agentes de normalidad (B_TA_{TC} y B_SA_{TC}, respectivamente) y de procesamiento por parte de la pizarra (B_{TP}). Finalmente, el cuarto bloque de columnas muestra los tiempos de acceso del agente de normalidad ena_2 a nivel de entorno a la pizarra (ENA_{TC}) y de procesamiento para analizar la normalidad de E_2 (ENA_{TP}). La última columna representa el tiempo total transcurrido desde que se envió el primer evento hasta que se procesó el último.

La figura 5.8 muestra gráficamente la fracción de tiempo dedicada a cada tarea en los tests 1, 3 y 4. Como se puede apreciar, la parte más significativa se corresponde con el tiempo empleado por el análisis realizado por el agente de preprocesamiento pa_2. Esto se debe a que dicho agente ha de calcular, por cada evento analizado, i) en qué zona o zonas se encuentra cada objeto detectado y ii) clasificarlo en función de sus características físicas. En la implementación desarrollada, la primera de esas dos tareas es relativamente costosa ya que se estudia a qué zona pertenece cada uno de los píxeles que forman la mitad inferior de la elipse que envuelve a cada objeto. Aunque este algoritmo se puede simplificar obteniendo resultados similares, en el presente prototipo se ha mantenido para justificar la necesidad de distribuir la carga de trabajo entre las réplicas del agente pa_2.

Test	T	Ev/s	PA_{TP}	TA_{TC}	TA_{TP}	SA_{TC}	SA_{TP}	B_TA_{TC}	B_SA_{TC}	B_{TP}	ENA_{TC}	ENA_{TP}	T_{total}
1	L	10	242,12	4,09	1,53	4,07	0,92	5,27	4,87	7,33	6,89	3,41	350
2	L	15	234,62	4,84	1,47	4,93	0,95	5,22	4,86	7,54	6,90	3,16	223
3	L	20	234,04	5,72	1,34	5,70	0,88	5,16	5,06	8,21	6,82	3,20	176
4	L	25	253,56	7,29	1,22	7,11	0,73	5,25	5,08	9,36	6,77	3,22	140
5	LR	10	340,90	5,80	1,34	5,83	1,50	5,16	5,54	8,04	7,38	3,89	350
6	LR	15	371,67	8,05	1,88	6,71	1,69	5,89	5,72	9,69	8,13	4,05	224
7	LR	20	394,70	13,10	1,79	12,35	3,05	6,44	6,26	14,81	8,37	4,21	176
8	LR	25	427,88	15,86	2,65	16,81	2,98	5,87	6,16	19,01	8,33	3,92	141
9	D	10	209,19	16.751	5,939	14.449	5,817	21.216	18.616	27,764	23,623	13,557	351
10	D	15	192.994	16.429	6.521	13,632	6.177	21.525	18,842	28,643	24,975	14,199	224
11	D	20	185,819	16.053	6,62	14.154	6.786	16.146	15,408	26,217	17,775	13,657	255
12	D	25	189,011	20,903	6.731	18.284	6,293	17,994	17,520	32,082	22,730	13,484	140

Tabla 5.6: Resultados del conjunto de pruebas realizado, variando la tasa de *frames* por segundo y la localización de los nodos (**L**=*host* local; **LR**=*host* local con replicación; **D**=distribuido).

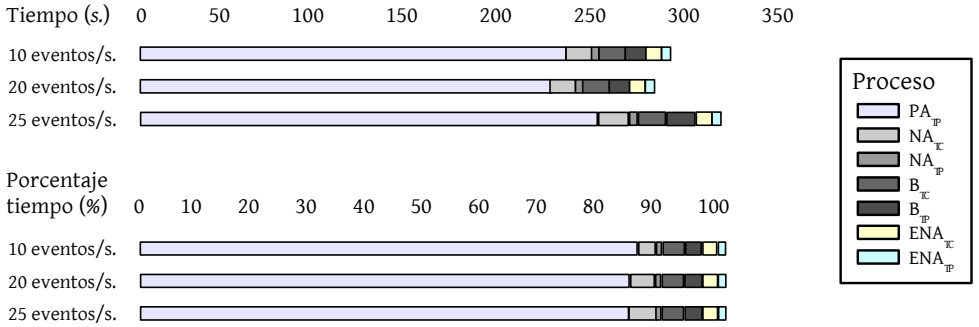


Figura 5.8: Tiempo empleado en segundos por cada uno de los agentes de vigilancia en realizar las distintas tareas (PA_{TC} = tiempo de procesamiento del *preprocessing agent*, NA_{TC} = tiempo de comunicación con los *normality agents*, NA_{TP} = tiempo de procesamiento de los *normality agents*, B_{TC} = tiempo de comunicación con la pizarra, B_{TP} = tiempo de procesamiento de la pizarra, ENA_{TC} = tiempo de comunicación con el *environmental normality analysis agent*, ENA_{TP} = tiempo de procesamiento del el *environmental normality analysis agent*). Tests realizados en una máquina (*redux.dyndns.org*) y sin replicar los agentes.

Por el contrario, los tiempos de procesamiento de los agentes de normalidad de análisis de trayectorias na_2 y na_3 (TA_{TP} y SA_{TP}) son muy bajos, lo cual es positivo para efectuar la monitorización de entornos en términos de análisis en *tiempo real*[2]. Estos agentes están implementados de acuerdo al modelo presentado en la sección 3.3. Por otra parte, los tiempos de procesamiento de la pizarra (B_{TP}) y del agente de normalidad a nivel de sub-entorno (ENA_{TP}) son ligeramente más elevados que los anteriores debido principalmente a la gestión de la concurrencia de la pizarra.

Los tiempos de comunicación son bastante constantes y representan porcentajes de tiempo bastante bajos en relación a los tiempos globales de procesamiento, aún cuando la tasa de eventos por segundo asciende a 25. Por ejemplo, en el test 1 el tiempo de comunicación del agente de análisis de trayectorias na_2 con la pizarra (B_TA_{TC}) es de $5,27$ segundos para los 3500 eventos, es decir, en torno a $1,5$ milésimas por evento.

Otro aspecto importante a destacar en las pruebas realizadas en una única máquina es la configuración de cada una de las réplicas de los agentes de vigilancia. En concreto, el parámetro más relevante es el número de hilos lógicos que manejarán para atender las invocaciones remotas recibidas a través del *middleware* ZeroC ICE. En el conjunto de tests realizado se ha optado por establecer este parámetro a 2, debido a que el rendimiento final del sistema está limitado por el número de elementos de procesamiento físicos de la máquina en la que se ejecutan los agentes, que en este caso es 2 como consecuencia de la arquitectura de la máquina *redux.dyndns.org* (Intel(R) Core(TM)2 Duo CPU).

[2]En este contexto, *tiempo real* se refiere a un tiempo lo suficientemente bajo para proporcionar una respuesta inmediata al procesamiento de un evento.

Una cuestión vinculada a este aspecto es el tiempo de procesamiento total por parte del sistema de vigilancia en relación a la duración del vídeo analizado. Por ejemplo, en el test 4, con una tasa de 25 eventos por segundo, la duración de la escena sería aproximadamente 140 segundos. En dicho test, la suma de los tiempos de procesamiento y de comunicación es mayor a estos 140 segundos. Sin embargo, este hecho se debe a que la arquitectura de la máquina en la que se ejecutan los agentes permite el procesamiento paralelo, siendo posible llevar a cabo el análisis de la escena bajo demanda, es decir, con la capacidad suficiente para analizar cada uno de los eventos en *tiempo real*.

La figura 5.10 muestra gráficamente la fracción de tiempo empleada por los agentes de vigilancia cuando se despliega el sistema planteado en la figura 5.7. Desde un punto de vista general, existen tres principales diferencias con respecto a los resultados obtenidos al utilizar una única máquina:

1. Reducción en el tiempo empleado por los agentes de preprocesamiento.
2. Aumento en el tiempo empleado por los agentes de normalidad.
3. Aumento en los tiempos de comunicación.

La primera diferencia se justifica mediante el esquema de replicación planteado, gracias al cual el tiempo de procesamiento se distribuye entre las réplicas del agente de preprocesamiento. Este hecho garantiza la respuesta de la arquitectura frente a altas cargas de trabajo, siendo posible incrementar el número de máquinas físicas cuando la complejidad de la escena a monitorizar crezca. En la figura 5.9 se muestra el gráfico de consumo de ciclos de CPU de la máquina *despacho201.dyndns.org* mientras se analiza el sub-entorno E_2.

El incremento en el tiempo de análisis de normalidad empleado por los agentes na_2 y na_3, en principio inesperado, se debe a que el código asociado incluye la escritura en la base de datos en la que se almacena toda la información generada. Debido a que dicha base de datos se encuentra en una máquina distinta (*exeterst.dyndns.org*) a la de los agentes de normalidad, el tiempo de análisis sufre dicho incremento. Si la persistencia en base de datos se suprime, los tiempos de análisis de dichos agentes son similares a los obtenidos con una única máquina. En este conjunto de pruebas se ha optado por acceder a la base de datos con el objetivo de posibilitar futuros análisis forenses de dicho registro.

Finalmente, el aumento en los tiempos de comunicación es la consecuencia directa de desplegar réplicas en distintas máquinas físicas. Por ejemplo, si se comparan los tiempos de comunicación del agente de análisis de normalidad ena_2 a nivel de sub-entorno con la pizarra entre el segundo y el tercer conjunto de pruebas (tests 5 a 8 y tests 9 a 12 de la tabla 5.6) cuando existe replicación, se puede apreciar que el incremento de tiempo varía entre un

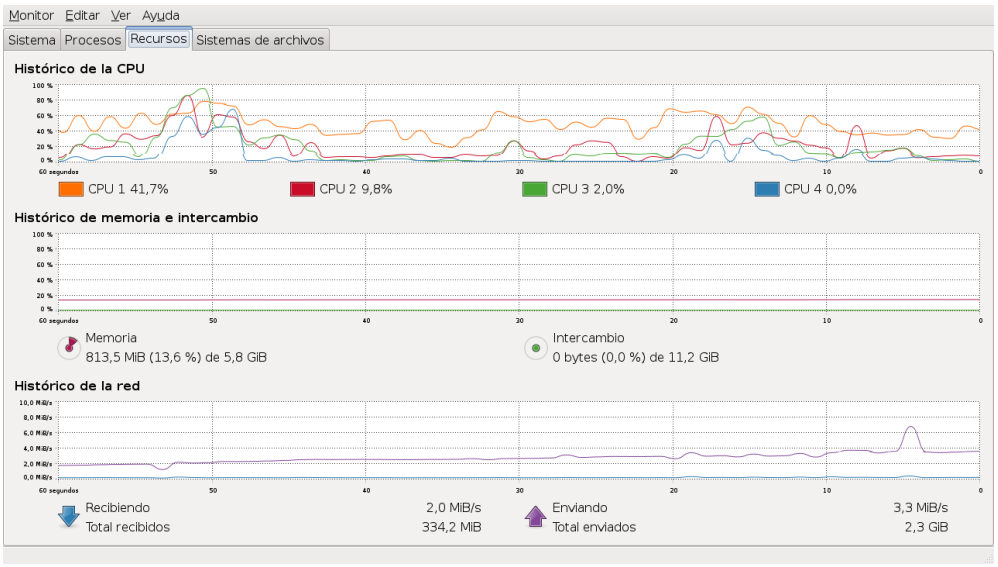


Figura 5.9: Gráfica de consumo de ciclos de CPU y ancho de banda de la máquina *despacho201.dyndns.org* durante la monitorización de E_2.

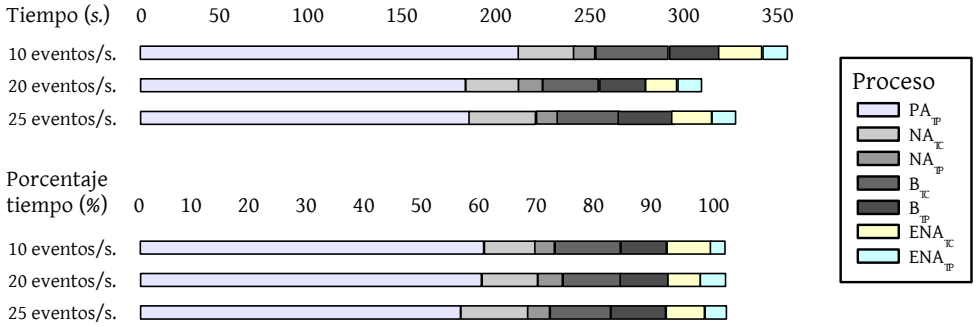


Figura 5.10: Tiempo empleado en segundos por cada uno de los agentes de vigilancia en realizar las distintas tareas (PA_{TC} = tiempo de procesamiento del *preprocessing agent*, NA_{TC} = tiempo de comunicación con los *normality agents*, NA_{TP} = tiempo de procesamiento de los *normality agents*, B_{TC} = tiempo de comunicación con la pizarra, B_{TP} = tiempo de procesamiento de la pizarra, ENA_{TC} = tiempo de comunicación con el *environmental normality analysis agent*, ENA_{TP} = tiempo de procesamiento del el *environmental normality analysis agent*). Tests realizados en la red de la figura 5.7.

factor de 2 a 3. En otras palabras, la penalización por el uso de invocaciones remotas hace que los tiempos de comunicación se dupliquen o incluso se tripliquen. El resto de tiempos de comunicación siguen esta misma tendencia.

Otra cuestión a considerar es la tasa deseable o aconsejada de eventos por segundo. Esta variable dependerá en primer lugar de la escena a monitorizar, ya que en determinados entornos será necesario analizar más información por unidad de tiempo para proporcionar una vigilancia más exacta. En cualquier caso, las pruebas realizadas han contemplado la posibilidad de analizar hasta 25 eventos por segundo, un valor bastante exigente si tenemos en cuenta que esta es la tasa media de grabación de las cámaras de seguridad. Incluso bajo este valor, el prototipo desarrollado permite obtener el conocimiento asociado al análisis del entorno de manera prácticamente inmediata.

5.2. Caso de estudio: integración de un agente de normalidad

En esta sección se analiza la integración de agentes de normalidad dentro de un sistema para la vigilancia inteligente desplegado de acuerdo a la arquitectura propuesta. El objetivo que se persigue es ilustrar los pasos necesarios para desplegar agentes de normalidad encargados de analizar un determinado tipo de concepto o amenaza. Así mismo, estos pasos se reflejaran en el caso concreto del concepto de *trayectorias* para facilitar la comprensión de la sección actual, pero se pueden extrapolar a cualquier otro agente de normalidad específico.

A continuación se enumeran los pasos necesarios para integrar un agente dentro de un sistema de vigilancia inteligente desplegado a partir de la arquitectura multi-agente desarrollada en este trabajo (dichos pasos serán objeto de estudio en las sucesivas sub-secciones):

1. Definición del concepto de análisis mediante un modelo de análisis de normalidad.
2. Definición de los tipos de datos vinculados al concepto de normalidad definido.
3. Desarrollo del agente encargado de gestionar el análisis del concepto.
4. Despliegue del agente en el sistema de vigilancia inteligente.
5. Registro en el sistema multi-agente e integración en el sistema de comunicaciones.

5.2.1. Definición del concepto de normalidad

Como ya se introdujo en la sección 3.3.1, la propuesta para adquirir el conocimiento mediante el modelo basado en conceptos de normalidad refleja dos ideas fundamentales:

1. un mecanismo para crear o instanciar conceptos,
2. un mecanismo para refinar o depurar las instancias de dichos conceptos previamente desplegados para realizar la vigilancia de cada concepto particular.

Este enfoque nos permite crear **componentes autocontenidos** que están formados por el concepto a vigilar, que se manifiesta a través de un *normality*

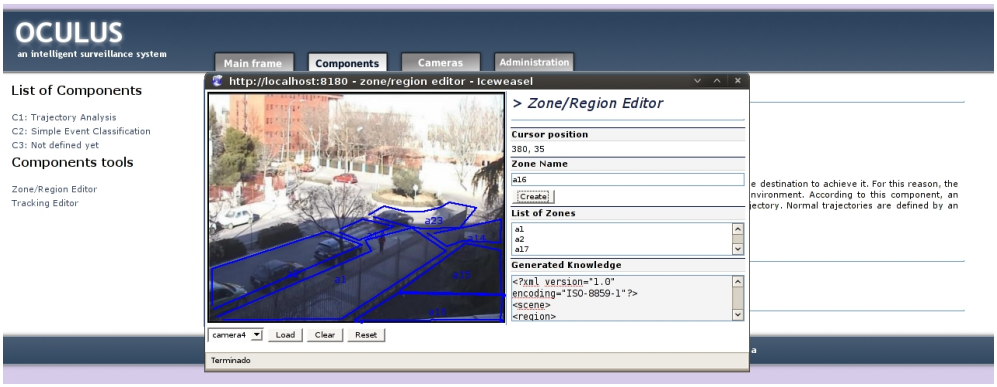


Figura 5.11: Herramienta gráfica de adquisición de conocimiento basado en el modelo de normalidad previamente introducido en la sección 3.3.

agent, un mecanismo de adquisición de conocimiento ligado a dicho concepto y que permite su instanciación en entornos particulares, un mecanismo para refinar dicha instancia teniendo en cuenta la evolución de la vigilancia asociada al concepto y, finalmente, unas reglas básicas de integración del componente en la arquitectura del sistema de vigilancia. De este modo, es posible plantear un modelo de desarrollo basado en componentes de vigilancia que albergan todos los elementos necesarios para especificar y extender la monitorización inteligente de entornos.

La alternativa más directa para crear o instanciar conceptos de normalidad, la cual se tendrá en cuenta para el caso particular de las trayectorias utilizado en esta sección, es el uso de **herramientas de adquisición de conocimiento** por parte de un experto. Este tipo de herramientas facilita la definición explícita del conocimiento propio del entorno a monitorizar, como las zonas físicas en las que se divide, y del conocimiento asociado al concepto a vigilar, como las trayectorias normales de los objetos móviles. Como se ha mencionado anteriormente, es necesaria la creación de una herramienta de adquisición de conocimiento cada vez que se diseñe un nuevo concepto de vigilancia. Sin embargo, una vez creada, la herramienta se puede utilizar para desplegar o instanciar dicho concepto en cualquier entorno que requiera una vigilancia inteligente.

La figura 5.11 muestra una captura de una herramienta de adquisición de conocimiento asociada al concepto de trayectorias, la cual permite la creación de instancias de dicho concepto en cualquier entorno de vigilancia. En [AVJL+09] se detalla en profundidad el proceso de definición del concepto de *trayectorias*, manejado por el agente de normalidad que representa el caso de estudio en esta sección.

La otra alternativa contemplada por el modelo para crear o instanciar conceptos de normalidad es el uso de un algoritmo o técnica de adquisición automática de conocimiento. Esta opción posibilita el aprendizaje a partir de una observación previa del entorno particular y de una fase de entrenamien-

to para poblar la base de conocimiento del *normality agent* asociado. Las principales ventajas de esta alternativa sobre las herramientas de adquisición de conocimiento son la reducción de la dependencia respecto al experto humano y la simplicidad a la hora de generar el conocimiento experto. En [ACSLL+09] se hace una descripción detallada del algoritmo de aprendizaje supervisado que se puede utilizar para generar el conocimiento de un componente de normalidad. Este algoritmo es capaz de obtener un conjunto de reglas que permiten clasificar los eventos simples que se pueden producir en un entorno monitorizado, en base a la clase a la que pertenece un objeto, su velocidad (obtenida a partir de su desplazamiento entre frames) y la zona en la que se encuentra.

5.2.2. Definición de los tipos de datos

Una vez definido el concepto de normalidad, resulta necesario definir los tipos de datos que permiten representar los distintos elementos del concepto de normalidad con un lenguaje neutro que posibilite su traducción a las distintas tecnologías involucradas en el proceso (ver figura 5.12). Por ejemplo, en el caso del concepto de *trayectorias* se asumen los siguientes supuestos:

- Dichos tipos de datos están asociados a las restricciones (incluyendo las distintas clases) que definen cada una de las trayectorias de un entorno.
- La herramienta de adquisición de conocimiento está desarrollada con un lenguaje de programación *A*, por ejemplo Java, o soportada por un sistema web desarrollado con una tecnología *B*, como PHP.
- El agente de análisis de normalidad está desarrollado con un lenguaje de programación compilado *C*, como por ejemplo C++.

Para permitir la interoperabilidad entre los distintos elementos de la arquitectura, la solución planteada en este trabajo para dar soporte a la diversidad de tecnologías existentes en un sistema de vigilancia inteligente es el uso de un middleware de comunicaciones, tal y como se introdujo en la sección 3.4. En concreto, el prototipo desarrollado para validar la arquitectura se ha desarrollado con el *middleware* ZeroC ICE [Hen04b], por lo que la definición de tipos de datos neutros se ha realizado con *Slice* (el lenguaje de definición de interfaces de ICE).

Inicialmente, el prototipo de la arquitectura multi-agente proporciona tipos de datos abstractos para la definición de los tipos de restricciones más comunes, tal y como sugiere el modelo de normalidad. En el siguiente listado se muestran dichos tipos de datos.

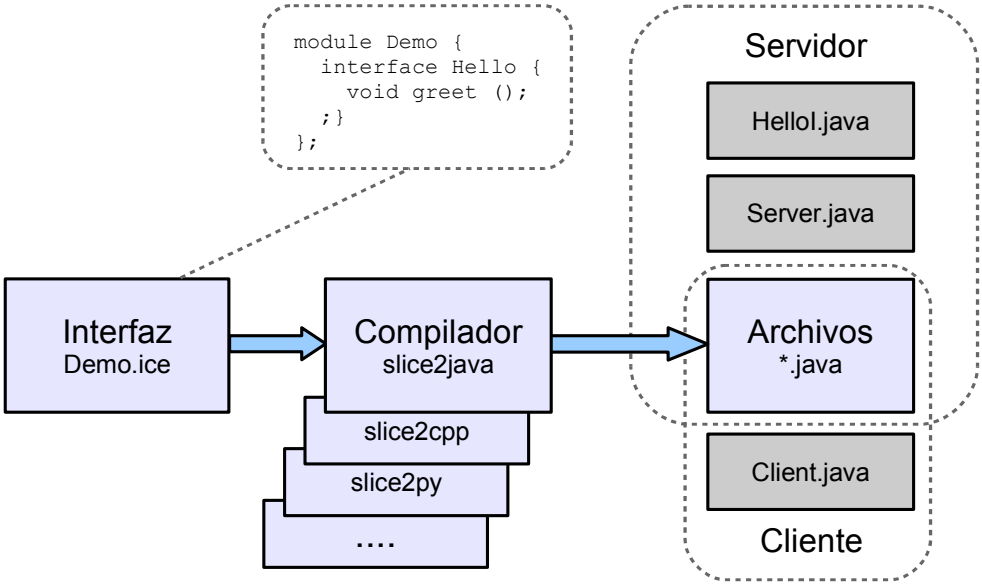


Figura 5.12: Modelo de desarrollo basado en la generación de *proxies* y esqueletos a partir de una descripción abstracta.

Tipos de datos abstractos. Restricciones

```
1  class SpatialRestriction {}; sequence<SpatialRestriction>
2  SpatialRestrictionSeq; class TemporalRestriction {};
3  sequence<TemporalRestriction> TemporalRestrictionSeq;
4  class ActorRestriction {};
5  sequence<ActorRestriction> ActorRestrictionSeq;
6
7  class Restrictions
8  {
9    SpatialRestrictionSeq spatialRestrictions;
10   TemporalRestrictionSeq temporalRestrictions;
11   ActorRestrictionSeq actorRestrictions;
12 };
```

Mediante este diseño basado en clases, el cual permite la herencia, es posible especializar los tipos de datos abstractos cuando se desee integrar un nuevo agente de análisis de normalidad. Sin embargo, lo más importante es que la interfaz del agente de análisis de normalidad no requiere ningún cambio debido a que maneja estos tipos de datos abstractos. En el siguiente listado se muestra la interfaz del componente utilizado para instanciar agentes de normalidad. En dicha interfaz se puede apreciar cómo los agentes de normalidad (*interface NA*) se despliegan mediante otro componente que está diseñado mediante el patrón *abstract factory* [GHJV95] y cómo la operación de creación (*create*) acepta como tercer parámetro las restricciones de tipo abstracto.

Interfaz del agente de análisis de normalidad

```
interface NA;
exception NAExists {};

interface NAFactory
{
  /* @param type The surveillance concept.
   * @param s The monitored scene characteristics.
   * @param rs The instantiated constraints of the concept.
   * @return The agent proxy. */
  NA* create (string type, Scene s, Restrictions rs)
        throws NAExists;
  // ...
};
```

En el caso particular de las trayectorias, los tipos de datos específicos de dicho concepto se muestran en el siguiente listado.

Restricciones específicas del concepto de *trayectorias*

```
class MovementSpatialRestriction extends SpatialRestriction
{
  Movement currentMovement;
  bool order;
  Ice::StringSeq allowedZones;
};

class MovementTemporalRestriction extends TemporalRestriction
{
  Movement currentMovement;
  Movement previousMovement;
  int duration;
  Interval temporalInterval;
};

class MovementActorRestriction extends ActorRestriction
{
  Movement currentMovement;
  Ice::StringSeq allowedActors;
};

class MovementRestrictions extends Restrictions {};
```

Esta definición de datos con un lenguaje de especificación neutro e independiente de la tecnología utilizada habilita la comunicación entre la herramienta de adquisición de conocimiento, por una parte, y el propio agente de razonamiento, por otra. Así mismo, estos tipos de datos específicos también se utilizan en los módulos de consulta de los agentes de normalidad, con el objetivo de posibilitar la obtención de información de las restricciones de los objetos monitorizados desde las herramientas de monitorización.

5.2.3. Desarrollo del agente de normalidad

Una vez definidos los tipos de datos necesarios para que el agente de normalidad específico interactúe con el resto de componentes de la arquitectura, el siguiente paso consiste en proporcionar la implementación física del mismo.

En el caso particular de las trayectorias, uno de los agentes de normalidad desarrollados ha sido implementado en C++ y en CLIPS. La parte relativa a C++ permite la interacción con el *middleware* de comunicaciones y, por lo tanto, con el resto de componentes de la arquitectura. La parte relativa a CLIPS sirve para materializar el conocimiento del concepto de *trayectorias* previamente definido con el modelo (ver sección 5.2.1). La integración entre la parte desarrollada en C++ y la parte desarrollada en CLIPS es posible gracias a la API[3] en C proporcionada por CLIPS, lo cual facilita enormemente la integración entre ambos lenguajes. En el anexo A se detalla la implementación realizada en CLIPS para dicho agente de análisis de trayectorias.

El listado de la página siguiene muestra parte del código desarrollado en C++ para el agente de análisis de normalidad de trayectorias. Como se puede apreciar, las principales operaciones representan la implementación real de la intefaz del agente genérico definida con el lenguaje de descripción de interfaces *Slice* de ZeroC ICE. Este enfoque facilita el modelo de desarrollo porque no es necesario alterar la interfaz abstracta de dicho agente. De hecho, el proceso consiste en implementar al sirviente o agente de normalidad que proporciona la funcionalidad necesaria para monitorizar un determinado concepto.

5.2.4. Despliegue del agente de normalidad

En el prototipo de arquitectura desarrollado en este trabajo, el despliegue de agentes de análisis de normalidad se realiza mediante una fábrica de agentes, la cual está diseñada de acuerdo al patrón *abstract factory* [GHJV95]. De este modo, existe un mecanismo establecido y ampliamente probado para facilitar no sólo el despliegue sino también la gestión de los agentes de normalidad. En el siguiente listado se muestra la interfaz de dicha fábrica.

[3] http://clipsrules.sourceforge.net/documentation/v630/apg.htm

Código en C++ del agente de *trayectorias*

```
#include <IceUtil/Mutex.h>

#include <Deliberative.h>
#include <NAI.h>

using namespace ISurveillance;

class MovementReasoningAgentI: public NAI
{
 public:
  MovementReasoningAgentI(const Scene&,
                          const RestrictionsPtr&,
                          Ice::CommunicatorPtr comm,
                          Ice::ObjectAdapterPtr oa);
  ~MovementReasoningAgentI();

  virtual QueryResultPtr query
    (const QueryInfoPtr&,
     const ::Ice::Current&);
  virtual void notifyEvent
    (const Video&,
     const ::Ice::Current&);
  virtual void destroy(const ::Ice::Current&);

};

typedef IceUtil::Handle<MovementReasoningAgentI>
                        MovementReasoningAgentIPtr;
```

Interfaz de la fábrica de agentes de normalidad

```
interface NA;
sequence<NA*> NASeq;
exception NAExists {};

interface NAFactory
{
  /**
   * Create a normality agent.
   * @param type The surveillance concept.
   * @param s The monitored scene characteristics.
   * @param rs The instantiated constraints of the concept.
   * @return The agent proxy.
   **/
  NA* create (string type, Scene s, Restrictions rs)
        throws NAExists;

        /**
   * Find out a normality agent.
   * @param type The surveillance concept.
   * @return The agent proxy.
   **/
  idempotent NA* find (string type);

  /**
   * Get the sequence of normality agents.
   * @return The sequence of normality agents.
   **/
  idempotent NASeq list ();
};
```

La operación *create* determina los tipos de datos necesarios para que el agente de normalidad pueda comenzar a operar:

1. *Type* representa el concepto a monitorizar, por ejemplo el concepto de *trayectorias*.
2. *s* representa la información del sub-entorno a monitorizar, típicamente la división en zonas físicas del mismo.
3. *rs* representa las restricciones que definen el concepto a monitorizar de acuerdo al modelo de normalidad.

Es importante resaltar que los tipos de datos recogidos en la operación *create* de la fábrica de agentes se ven reflejados en el método constructor del agente de normalidad en cuestión. Este enfoque garantiza la escalabilidad de la arquitectura cuando se han de integrar nuevos agentes de normalidad.

En el caso del agente de análisis de trayectorias, su instanciación en el sistema de vigilancia inteligente la solicita a la herramienta de adquisición de conocimiento desarrollada para definir las trayectorias normales del sub-entorno a monitorizar. Básicamente, este proceso consiste en realizar una invocación remota (operación *create*) desde la herramienta mediante el núcleo de comunicaciones de ICE al servidor que soporta la fábrica de agentes.

A nivel de despliegue en el sistema distribuido, la fábrica de agentes que gestiona la instanciación de agentes de normalidad[4] hace uso de dos servicios avanzados del *middleware* ZeroC ICE. Por una parte, *IceBox* permite la administración y configuración del sirviente que da soporte a la fábrica de agentes. Por otra parte, *IceGrid* gestiona el proceso de despliegue propiamente dicho, facilitando enormemente todos los procesos de administración asociados al servidor.

IceGrid maneja los conceptos de *servicio* y *servidor* para independizar la definición y parametrización de los mismos a la hora de desplegar aplicaciones distribuidas, posibilitando la creación de plantillas que automaticen el proceso de despliegue de ambos elementos (los servicios están contenidos en los servidores). Por este motivo, se han desarrollado sendas plantillas que facilitan el proceso de administración de las fábricas de agentes. Dichas plantillas se crean en los descriptores que, como se introdujo en la sección 2.4.3, son archivos XML gestionados por *IceGrid* que incluyen todos los elementos de la aplicación. A continuación se muestran sendos listados con las plantillas asociadas al servicio y al servidor de la fábrica de agentes.

[4] Cabe la opción de desplegar más de una fábrica.

Plantilla del servicio asociado a la fábrica

```
<service-template id="ReasoningService">
  <parameter name="id" default="Reasoning.${server}"/>
  <parameter name="adapter" default="${adapter}"/>
  <service name="ReasoningServer"
           entry="ReasoningServer:createReasoningServer">
   <properties>
    <property name="adapter-id" value="${id}.Adapter"/>
    <property name="object-id" value="${id}"/>
    <property name="topic_manager_id"
              value="IceStorm/TopicManager"/>
    <property name="topic_video_id" value="Camera"/>
    <property name="topic_abnormal_id" value="Abnormality"/>
   </properties>
   <adapter name="${id}.Adapter" endpoints="default"
            id="${id}.Adapter" register-process="true"
            replica-group="ReplicatedReasoningAdapter">
    <object identity="ISurveillance/ReasoningAgentFactory"
            type="::ISurveillance::ReasoningAgentFactory"/>
   </adapter>
  </service>
</service-template>
```

Plantilla del servidor asociado a la fábrica

```
<server-template id="IceBoxReasoningServer">
  <parameter name="index" default="0"/>
  <parameter name="instance-name"
             default="${application}.IceBoxReasoningServer"/>
  <parameter name="adapter" default="default"/>
  <icebox id="${instance-name}.${index}"
          activation="always" exe="icebox">
   <properties>
    <property name="IceBox.ServiceManager.Endpoints"
              value="tcp -h 127.0.0.1"/>
    <property name="IceBox.ServiceManager.ReasoningProcess"
              value="1"/>
    <property name="IceBox.InstanceName" value="${server}"/>
    <property name="IceBox.ServiceReasoning.Endpoints"
              value="tcp -h 127.0.0.1"/>
   </properties>
   <log path="logs/${instance-name}.${index}.err"
        property="Ice.StdErr"/>
   <log path="logs/${instance-name}.${index}.out"
        property="Ice.StdOut"/>
   <service-instance template="ReasoningService"/>
  </icebox>
</server-template>
```

5.2.5. Registro del agente de normalidad en el SMA

Una vez que el agente de normalidad ya está en ejecución, éste se ha de registrar en el sistema multi-agente de vigilancia desplegado para monitorizar un entorno concreto. En la sección 3.5.1 se estableció que dicho registro ha de ser con los agentes pm (*platform manager*) y *sm* (*service manager*), responsables de proporcionar los servicios de páginas blancas (más ciertas ca-

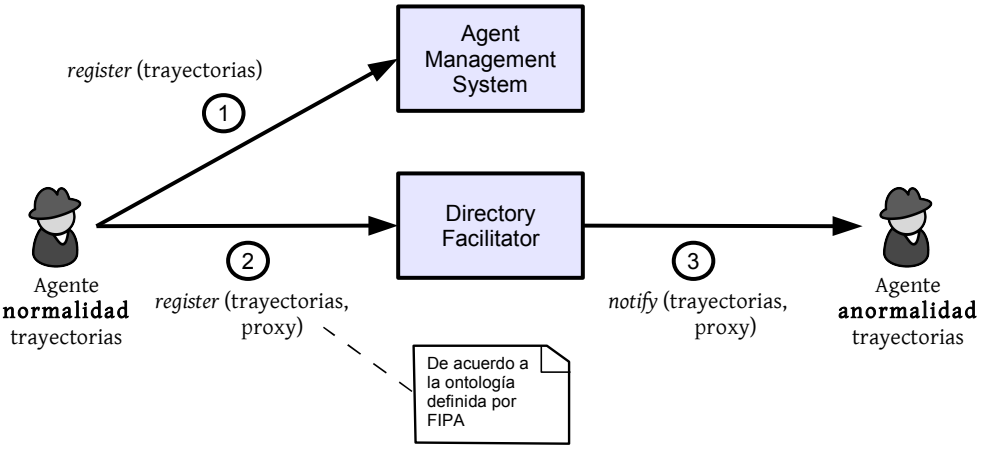


Figura 5.13: Interacciones a nivel de gestión entre un agente de normalidad, el *Agent Management System*, el *Directory Facilitator* y un agente de anormalidad .

pacidades de gestión) y páginas amarillas, respectivamente. En el desarrollo del prototipo de la arquitectura, tal y como se introdujo en la sección previamente referenciada, estos agentes siguen el esquema de los componentes *Agent Management System* (AMS) y *Directory Facilitator* (DF), respectivamente.

En el caso de las trayectorias, el agente de normalidad solicita su registro en la plataforma con el identificador *Trayectorias* a través del AMS. Así mismo, este agente se registra en el DF proporcionando la descripción de su servicio, la cual básicamente consiste en especificar qué servicio proporciona (análisis de normalidad de trayectorias) y cómo otros agentes pueden contactar con él (identificador del objeto remoto). Este proceso se muestra de manera gráfica en la figura 5.13. Estas dos funcionalidades se realizan en tiempo de ejecución una vez que el agente haya sido instanciado en el sistema de vigilancia.

5.3. Evaluación de la plataforma multi-agente de propósito general

Para evaluar la eficiencia y la escalabilidad de la plataforma de propósito general propuesta para el despliegue de sistemas multi-agente se ha realizado un extenso conjunto de pruebas. En esencia, estos tests están orientados al envío de mensajes en aplicaciones con una alta demanda de dicho proceso y al consumo de recursos a la hora de desplegar sistemas con un alto número de agentes.

En la tabla 5.7 se resumen las especificaciones técnicas de las máquinas utilizadas para evaluar el rendimiento de la plataforma de agentes. En este contexto, es importante tener en cuenta el número de núcleos (*cores*) físicos de los procesadores de las máquinas usadas para desplegar el sistema de pruebas. El motivo reside en que uno de los aspectos de la evolución de las estaciones de trabajo es el multi-procesamiento, de manera que los desarrolladores han de tener en cuenta este aspecto para obtener sistemas de alto rendimiento. Aunque la plataforma de agentes propuesta es esencialmente multi-hilo, el rendimiento final de la misma estará condicionado por el número de núcleos físicos. Es decir, cuanto mayor sea el número de elementos de procesamiento, mayor rendimiento tendrá el sistema.

Sistema operativo	Debian GNU/Linux
Procesador	Intel(R) Core(TM)2 CPU
Máxima frecuencia de la CPU	Desde 2.13 GHz hasta 2.2 GHz
Caché	4 MB
RAM	Desde 1 GB hasta 2 GB

Tabla 5.7: Especificaciones técnicas de los computadores de escritorio utilizados para evaluar la plataforma de agentes.

5.3.1. Test de *spam*

Descripción y objetivos

Este primer conjunto de tests para evaluar la eficiencia de la arquitectura multi-agente está motivado por los resultados obtenidos a la hora de evaluar JADE en un escenario con una alta demanda en lo que a intercambio de mensajes entre agentes se refiere [CGK+05]. El objetivo es establecer una comparación entre la plataforma discutida en esta sección con uno de los *frameworks* más maduros y utilizados en la actualidad.

El esquema utilizado para desplegar este escenario está basado en un conjunto de máquinas en el que se instancias pares de agentes *spammer* y *usuarios*. Básicamente, los *spammers* envían mensajes a los *usuarios*, que

son los agentes encargados de procesarlos. El procesado de mensajes consiste simplemente en la correcta recepción de los mismos. El número de pares de agentes instanciados, el número de mensajes enviados por cada *spammer* o el tamaño de los mensajes varía en cada uno de los tests realizados. Los resultados obtenidos son los tiempos empleados en efectuar el envío de todos los mensajes y los tiempos utilizados por los agentes de usuario para procesarlos, respectivamente. Respecto a la evaluación de JADE en este escenario, los autores concluyen que la escalabilidad del sistema es buena aunque la eficiencia se ve afectada por el rendimiento de Java.

Para ejecutar este conjunto de pruebas se ha desarrollado un sencillo sistema multi-agente que responde a la arquitectura planteada de manera gráfica en la figura 5.14. Este sistema se ha concebido de manera que el proceso de ajuste de parámetros sea sencillo e independiente de los procesos de compilación y enlazado del mismo. Dichos parámetros son los siguientes: número de máquinas, número de nodos, número de agentes de *spam*, número de agentes de usuario, número de mensajes enviados por cada *spammer* y tamaño de los mensajes. Así mismo, y con el objetivo de reflejar la flexibilidad de la plataforma de agentes respecto al lenguaje de programación empleado para implementarlos, los agentes de *spam* se han implementado en Python y los agentes de *usuario* en C++.

De manera adicional a este conjunto de tests, inspirados en la evaluación llevada a cabo en [CGK+05], las pruebas diseñadas en esta sección también contemplan el uso del ***Message Transport Service*** como elemento intermedio utilizado por los agentes a la hora de enviar mensajes. De este modo, es posible evaluar la eficiencia del sistema respecto a los mecanismos de comunicación desde dos puntos de vista distintos (así contemplados en las especificaciones de FIPA):

1. comunicación directa entre agentes,
2. comunicación indirecta entre agentes haciendo uso del MTS como intermediario en el proceso.

Obviamente, la inclusión de un intermediario en el proceso de comunicación entre agentes puede suponer, a priori, una reducción en la eficiencia del sistema. Sin embargo, y así se ha llevado a cabo en la evaluación de este segundo conjunto de tests, el MTS se puede replicar para repartir la carga del mismo y distribuir el trabajo asociado al envío y recepción de mensajes entre distintas réplicas, normalmente desplegadas en distintas máquinas físicas.

Resultados obtenidos

La primera serie de tests de *spam* en el escenario previamente descrito se resumen en la tabla 5.8. La configuración para esta serie de pruebas es la siguiente:

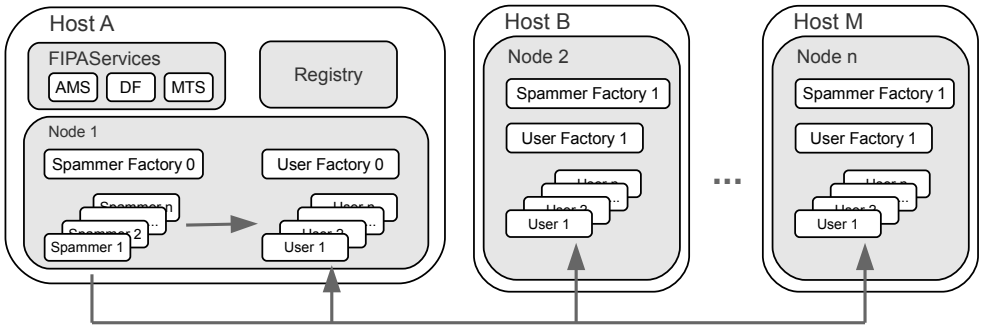


Figura 5.14: Escenario utilizado para evaluar la eficiencia del sistema a la hora de enviar mensajes entre agentes.

- Despliegue de 3 pares de agentes en cada nodo. Por ejemplo, si el número de nodos es 3, entonces el número total de agentes de *spam* es 9, al igual que el número total de agentes de usuario.
- El tamaño de los mensajes es de 300 *bytes*.
- El modelo de comunicación entre agentes es directo, es decir, sin hacer uso del MTS.

A partir de estos resultados se obtienen las siguientes **conclusiones** (ver también las figuras 5.15.a y 5.15.d). En primer lugar, los tiempos de *spamming* y de procesamiento crecen de manera proporcional al número de mensajes y, por lo tanto, a la carga del sistema. En segundo lugar, los tiempos de *spamming* empiezan a ser relevantes cuando el número de mensajes sobrepasa los 100.000. Sin embargo, es importante tener en cuenta que los *spammers* han de ejecutar el código necesario para atender los objetos de retrollamada (*callback*), ya que los mensajes se envían de manera asíncrona tal y como se discutió en la sección 4.3. Finalmente, el tiempo de procesamiento es mayor de un minuto cuando el número de *spammers* es de 9 y el número de mensajes es de 648.000.

En el segundo conjunto de tests, el tamaño de los mensajes se incrementa hasta los 600 *bytes* y los resultados se muestran en la tabla 5.9 (el resto de parámetros no cambian). Los datos obtenidos sirven para confirmar las conclusiones obtenidas a partir de los resultados obtenidos en el primer conjunto de pruebas: la escalabilidad respecto a la carga del sistema y a los tiempos de procesamiento. Por otra parte, el hecho de doblar el tamaño de los mensajes no tiene un efecto significativo en los tiempos resultantes.

Finalmente, en el tercer conjunto de pruebas el tamaño de los mensajes se incrementa hasta los 1.200 *bytes*. Los resultados se muestran en la tabla 5.10. En ellos, los tiempos de procesamiento se incrementan ligeramente, pero no de una manera proporcional al incremento del tamaño de los mensajes (como ocurría en el anterior conjunto de tests), es decir, el doble. Este

Test	Nodos	Mensajes *spammer*	Total mensajes	Tráfico (MB)	t_s (s)	t_p (s)
1	1	1000	9000	2.57	0.49	0.63
2	2	1000	36000	10.30	3.73	6.85
3	3	1000	81000	23.17	5.61	11.25
4	1	2000	18000	5.14	1.21	1.22
5	2	2000	72000	20.60	6.38	12.67
6	3	2000	162000	46.34	10.34	21.51
7	1	4000	36000	10.30	2.06	2.45
8	2	4000	144000	41.20	11.97	24.61
9	3	4000	324000	92.68	20.24	43.08
10	1	8000	72000	20.60	3.33	5.04
11	2	8000	288000	82.40	23.16	48.39
12	3	8000	648000	185.36	38.40	83.58

Tabla 5.8: Primer test de *spam*: 3 parejas de agentes por nodo, mensajes de 300 *bytes* .

Test	Nodos	Mensajes *spammer*	Total mensajes	Tráfico (MB)	t_s (s)	t_p (s)
1	1	1000	9000	5.14	0.60	0.62
2	2	1000	36000	20.60	3.58	6.88
3	3	1000	81000	46.34	5.40	11.55
4	1	2000	18000	10.28	0.83	1.39
5	2	2000	72000	41.20	6.44	13.01
6	3	2000	162000	92.68	10.76	22.84
7	1	4000	36000	20.60	2.33	2.05
8	2	4000	144000	82.40	11.99	25.07
9	3	4000	324000	185.36	20.93	44.91
10	1	8000	72000	41.20	4.25	5.20
11	2	8000	288000	164.80	23.86	50.17
12	3	8000	648000	370.72	40.02	88.62

Tabla 5.9: Segundo test de *spam*: 3 parejas de agentes por nodo, mensajes de 600 *bytes* .

hecho corrobora que el rendimiento del sistema no depende directamente del tamaño de los mensajes, sino del número de ellos. Por ejemplo, el test número 12 del primer y tercer experimento cargan al sistema con 185.36 MB y 741.44 MB, respectivamente. Sin embargo, los tiempos de procesamiento son de 83 segundos y 104 segundos. Es decir, el tamaño de los datos enviados se ha cuadruplicado mientras que los tiempos de procesamiento sólo se han incrementado en un factor de 1.25. Por otra parte, y de acuerdo a los tiempos de *spamming*, el impacto del tamaño de mensaje es mínimo en todos los casos (ver figuras 5.15.c y 5.15.f).

Con el objetivo de ilustrar la escalabilidad de la plataforma propuesta, los conjuntos de tests discutidos hasta el momento se han replicado haciendo uso de JADE y de las mismas máquinas físicas. Esto nos permite comparar

Test	Nodos	Mensajes *spammer*	Total mensajes	Tráfico (MB)	t_s (s)	t_p (s)
1	1	1000	9000	10.28	0.67	0.68
2	2	1000	36000	41.20	4.24	7.96
3	3	1000	81000	92.68	5.37	14.34
4	1	2000	18000	20.56	1.42	1.43
5	2	2000	72000	82.40	7.71	15.35
6	3	2000	162000	185.36	9.41	26.78
7	1	4000	36000	41.20	2.55	2.77
8	2	4000	144000	164.80	14.71	30.16
9	3	4000	324000	370.72	18.49	50.01
10	1	8000	72000	82.40	5.22	5.31
11	2	8000	288000	329.60	30.18	60.84
12	3	8000	648000	741.44	38.98	104.36

Tabla 5.10: Tercer test de *spam*: 3 parejas de agentes por nodo, mensajes de 1.200 *bytes* .

ambos resultados y justificar la importancia de adoptar nuevos enfoques que contribuyan a la evolución de los *middlewares* de agentes. La tabla 5.11 muestra los tiempos de *spam* y procesamiento empleados por JADE bajo los siguientes supuestos: tres pares de agentes por nodo, dos nodos y variando el número de mensajes enviados y el tamaño de los mismos. El envío de mensajes con JADE también se ha realizado de manera asíncrona.

Si se comparan, por ejemplo, los tiempos empleados cuando cada *spammer* envía 4.000 mensajes (test 8 de las tablas 5.8, 5.9, 5.10 y tests 7, 8, 9 de la tabla 5.11) difieren considerablemente. JADE escala bien a la hora de ejecutar los tests de *spam*, pero los tiempos resultantes son mayores que los obtenidos con la plataforma propuesta. No obstante, quizá la diferencia más relevante radique a la hora de incrementar el tamaño del mensaje hasta los 1.200 *bytes*. JADE prácticamente dobla los tiempos de procesamiento respecto al tiempo obtenido con mensajes de 600 *bytes* (ver tests 8 y 9 de la tabla 5.11), mientras la plataforma propuesta no depende de dicho parámetro hasta tal extremo. Además, los tiempos de procesamiento con JADE son más elevados, lo cual podría deberse a los lenguajes de programación empleados por cada una de las plataformas y a la manera de gestionar la recepción de mensajes. Se puede apreciar como el resto de pruebas corroboran estas conclusiones.

La segunda serie de tests realizada sigue la mismo filosofía que la primera, es decir, evaluar el rendimiento de la plataforma a la hora de enviar/procesar un alto número de mensajes. Sin embargo, en esta serie se pretende evaluar la eficiencia del sistema cuando el *Message Transport Service* (MTS) se utiliza como gestor de mensajes. Los principales objetivos de este nuevo conjunto de pruebas son los siguientes:

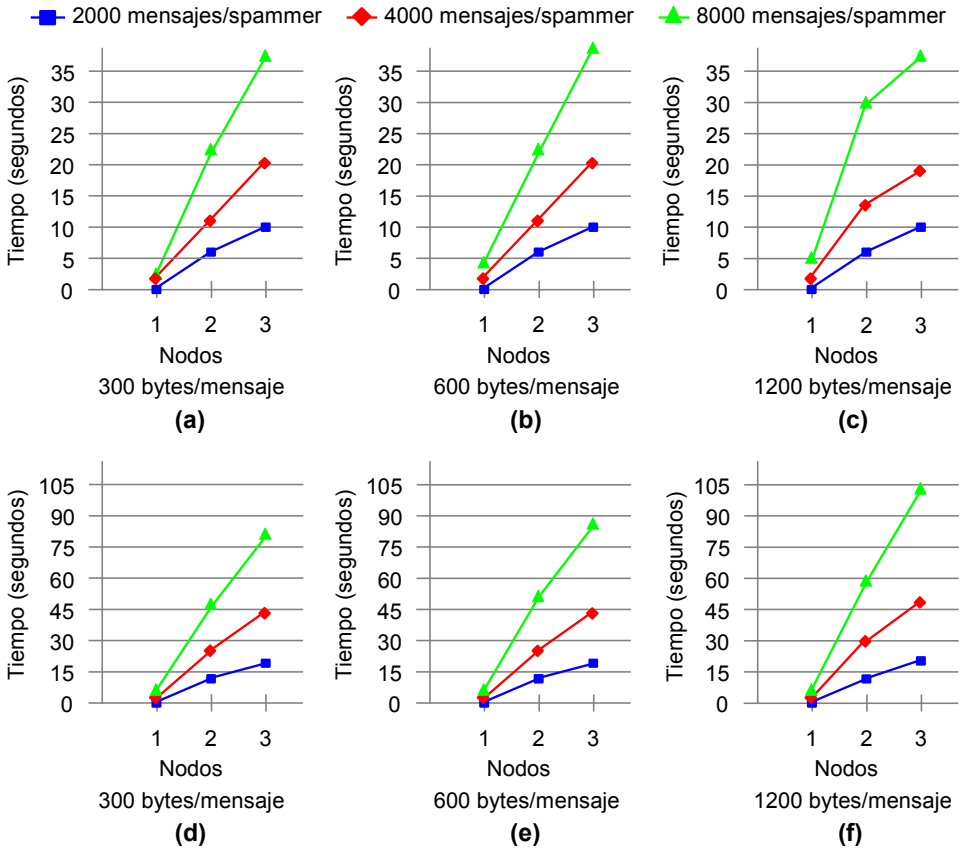


Figura 5.15: a), **b)**, **c)**. Tiempos de *spamming*. **c)**, **d)**, **e)**. Tiempos de procesamiento de mensajes.

Test	Nodos	Mensajes *spammer*	Total mensajes	Tráfico (MB)	t_s (s)	t_p (s)
1	1000	300	36000	10.30	1.89	24.77
2	1000	600	36000	20.60	2.94	31.13
3	1000	1200	36000	41.20	3.39	38.45
4	2000	300	72000	20.60	5.67	27.28
5	2000	600	72000	41.20	7.80	32.50
6	2000	1200	72000	82.40	12.61	59.73
7	4000	300	144000	41.20	20.34	29.90
8	4000	600	144000	82.40	23.34	30.05
9	4000	1200	144000	164.80	25.72	72.39
10	8000	300	288000	82.40	44.47	39.84
11	8000	600	288000	164.80	46.69	91.26
12	8000	1200	288000	329.60	51.41	156.45

Tabla 5.11: Tests de *spam* con JADE: 3 pares de agentes por nodo, 2 nodos.

1. Establecer una comparación con los resultados previamente obtenidos, es decir, sin hacer uso del MTS.
2. Evaluar el impacto en la eficiencia del sistema cuando el MTS se despliega con distintas configuraciones.
3. Proporcionar directrices sobre cómo configurar la plataforma en función de los requisitos de comunicación de la aplicación de dominio.

La tabla 5.12 resume los resultados obtenidos bajo los siguientes supuestos:

- El número de nodos es de 3 y cada uno de ellos se ejecuta en una máquina distinta.
- El número de pares de agentes desplegados en cada nodo es de 3, por lo que el número total de pares de agentes es 9 (9 agentes de *spam* y 9 agentes de usuario).
- El tamaño de los mensajes se establece a 300 *bytes*.
- La política de balanceado de carga entre las réplicas del MTS es adaptativa, es decir, cada petición es procesada por el nodo menos cargado a nivel de CPU en un momento determinado.

Los resultados de la tabla 5.12 ofrecen conclusiones interesantes. En primer lugar, el tiempo de *spam* se reduce significativamente respecto a la comunicación directa entre agentes. Este hecho se debe a que los *spammers* pueden enviar ahora un mensaje con múltiples receptores, de manera que el MTS se encargará de hacerlos llegar a dichos destinatarios. Sin embargo, el tiempo de procesamiento se incrementa notablemente, sobre todo en el caso de utilizar una única réplica del MTS. De hecho, el MTS se convierte en el cuello de botella del sistema cuando el número de mensajes se incrementa. El segundo aspecto importante es que la política de balanceado de carga adaptativa no escala bien en este experimento. Esta conclusión se puede apreciar porque los tiempos de procesamiento no se reducen de manera considerable cuando se hace uso de más de una réplica.

Finalmente, el último conjunto de tests de *spam* se muestra en la tabla 5.13, en el que se ha utilizado una política de balanceado de carga basado en el algoritmo de turno rotatorio (*round-robin*). En otras palabras, las distintas réplicas del MTS atienden las peticiones recibidas siguiendo un orden rotatorio.

En este último experimento, los tiempos de procesamiento mejoran al incrementar el número de réplicas (ver test 9 de las tablas 5.12 y 5.13, respectivamente). La principal conclusión obtenida después de evaluar la plataforma de agentes con distintas configuraciones es que el desarrollador es el que ha de tener en cuenta las características del dominio de aplicación para escoger el balanceado de carga más adecuado.

Test	Mensajes *spammer*	Mensajes	Tráfico (MB)	Réplicas	t_s (s)	t_p (s)
1	1000	81000	23.17	1	0.57	69.07
2	1000	81000	23.17	2	0.95	61.65
3	1000	81000	23.17	3	0.94	47.15
4	2000	162000	46.34	1	1.17	135.95
5	2000	162000	46.34	2	1.63	103.15
6	2000	162000	46.34	3	1.66	95.87
7	4000	324000	92.68	1	1.99	273.91
8	4000	324000	92.68	2	2.41	237.02
9	4000	324000	92.68	3	2.61	197.12
10	8000	648000	185.36	1	2.85	538.76
11	8000	648000	185.36	2	3.11	525.87
12	8000	648000	185.36	3	4.32	246.13

Tabla 5.12: Cuarto test de *spam*: 3 nodos, 3 pares de agentes por nodo, mensajes de 300 *bytes*, comunicación mediante el MTS y política de balanceado de carga adaptativa.

Test	Mensajes *spammer*	Mensajes	Tráfico (MB)	Réplicas	t_s (s)	t_p (s)
1	1000	81000	23.17	1	0.44	64.95
2	1000	81000	23.17	2	0.86	50.87
3	1000	81000	23.17	3	0.85	52.08
4	2000	162000	46.34	1	1.26	129.51
5	2000	162000	46.34	2	1.54	105.38
6	2000	162000	46.34	3	1.29	58.38
7	4000	324000	92.68	1	2.09	259.25
8	4000	324000	92.68	2	2.34	212.29
9	4000	324000	92.68	3	2.17	126.25
10	8000	648000	185.36	1	3.07	549.17
11	8000	648000	185.36	2	4.48	478.38
12	8000	648000	185.36	3	3.89	235.41

Tabla 5.13: Quinto test de *spam*: 3 nodos, 3 pares de agentes por nodo, mensajes de 300 *bytes*, comunicación mediante el MTS y política de balanceado de carga de turno rotatorio.

5.3.2. Test de despliegue de agentes

Descripción y objetivos

Una de los aspectos relevantes a considerar a la hora de desplegar aplicaciones multi-agente, especialmente en dominios con restricciones computacionales, es el **uso eficiente de recursos**. En este contexto, surge la cuestión de cómo gestionar los recursos físicos de los distintos agentes que componen la aplicación, entendiendo dichos recursos principalmente como consumo de memoria y tiempo empleado para cambiar de estado, por ejemplo de *hibernación* a *activo*.

Una posible solución consiste simplemente en incrementar el número de recursos, de modo que si el número de agentes crece, entonces se pueden utilizar máquinas más potentes, incrementar sus prestaciones o utilizar un mayor número. Sin embargo, desde un punto de vista práctico es más interesante adoptar un mecanismo que permita que los agentes hibernen cuando no son necesarios, con el objetivo de optimizar el uso de los recursos del sistema. Por ejemplo, no resulta práctico, generalmente, que mil agentes residan en memoria principal de manera simultánea. De hecho, resulta más coherente que sólo un subconjunto de dichos agentes residan en memoria, mientras que el resto permanece en un estado de hibernación en el almacenamiento persistente. Para afrontar este reto, la plataforma propuesta hace uso de un mecanismo de activación de agentes cuando éstos son necesarios.

El objetivo de este conjunto de experimentos es evaluar el mecanismo implementado por las fábricas de agentes midiendo la penalización que supone la gestión de persistencia y la activación de agentes. Los tests ejecutados están asociados a la instanciación de agentes con un estado de 1.500 *bytes*, permitiendo obtener la memoria utilizada por los servidores en función del número de agentes creados con una determinada fábrica. De este modo, es posible comparar los recursos consumidos por las fábricas de agentes con y sin persistencia. En el caso de hacer uso de dicha persistencia, también es interesante medir el tiempo empleado en activar un agente que se encuentre en estado de hibernación.

Finalmente, otro parámetro importante de las fábricas de agentes con persistencia es el tamaño de la cola de agentes en memoria. El desarrollador de la aplicación multi-agente ha de tener en cuenta el valor de este parámetro para obtener un alto rendimiento, sobre todo cuando el número de agentes de la aplicación crece. Por esta razón, y para proporcionar una evaluación más rigurosa, los experimentos realizados también contemplan variaciones en el valor de dicho parámetro.

Resultados obtenidos

Los tests para evaluar la eficiencia de las fábricas de agentes se han realizado bajo los siguientes supuestos:

- El estado de un agente es de 1.500 *bytes*.
- El número de invocaciones remotas efectuadas sobre los agentes supone la cuarta parte del número de agentes instanciados en cada test. Por ejemplo, si se despliegan 1.000 agentes, entonces se ejecutarán 250 invocaciones sobre los mismos de manera aleatoria. El objetivo perseguido es medir el tiempo empleado en activar agentes que estuvieran en estado de hibernación.
- La plataforma para ejecutar los tests ha sido un ordenador portátil con un procesador Intel(R) Core(TM)2 CPU 1.8 GHz, 2 GB de RAM y con un

Agentes instanciados	Persistencia	RAM	Tiempo creación (s)	Peticiones aleatorias	Time (s)
100	No	976 KB	0.049	25	0.008
100	Yes	1.7 MB	0.108	25	0.009
1000	No	4.2 MB	0.549	250	0.059
1000	Yes	1.8 MB	2.271	250	0.083
10000	No	36.6 MB	4.007	2500	0.546
10000	Yes	1.9 MB	19.494	2500	0.658

Tabla 5.14: Fábrica de agentes implementada en C++. El tamaño de la cola de agentes activos es de 10.

disco duro de 7.200 r.p.m. El sistema operativo empleado fue Debian GNU/Linux AMD64 con el compilador g++-4.3.2 (sin optimizaciones), la máquina virtual Java 1.6 y la versión 2.5 de Python.

- El número de hilos de los servidores se establece a 2 y el modelo de comunicaciones es síncrono.

Las tablas 5.14 y 5.15 muestran los resultados obtenidos con las fábricas de agentes implementadas en C++ y Java, respectivamente, cuando el tamaño de la cola de agentes activos es de 10. La tabla 5.16 muestra los resultados cuando la fábrica está implementada en Python sin soporte de persistencia para los agentes. Todos estos resultados se resumen de manera gráfica en la figura 5.16.

Las tablas 5.17 y 5.18 muestran los resultados obtenidos con las fábricas de agentes implementadas en C++ y Java, respectivamente, cuando el tamaño de la cola de agentes activos se establece a 100.

A continuación, se discuten las conclusiones obtenidas después de haber estudiado los resultados de esta serie de experimentos. Las fábricas de agentes con persistencia son más lentas a la hora de instanciar agentes, pero consumen menos memoria cuando el número de agentes se incrementa. Sin embargo, la cuestión más relevante es que el impacto en el rendimiento del sistema es mínimo cuando se reactivan los agentes, tal y como se puede apreciar en los tiempos asociados a las solicitudes o invocaciones remotas. Por lo tanto, y teniendo en cuenta dichos resultados, se puede concluir que las fábricas de agentes con persistencia mantienen una gran escalabilidad pero requieren una pequeña inversión inicial a la hora de crear la base de datos de los agentes instanciados.

Por otra parte, si el tamaño de la cola de los agentes activos crece, los tiempos se reducen pero el uso de RAM es ligeramente superior. El desarrollador de la aplicación ha de evaluar las distintas posibilidades en función de los requisitos de la misma. Finalmente, merece la pena reseñar que la versión implementada con Java consume mucha más memoria que la de C++, pero escala bien en tiempos de procesamiento y en el uso de memoria.

Agentes instanciados	Persistencia	RAM	Tiempo creación (s)	Peticiones aleatorias	Time (s)
100	No	20.8 MB	0.178	25	0.010
100	Yes	22.4 MB	0.304	25	0.022
1000	No	27.8 MB	1.177	250	0.112
1000	Yes	36.3 MB	2.219	250	0.139
10000	No	90.7 MB	6.581	2500	0.619
10000	Yes	44.9 MB	25.320	2500	1.009

Tabla 5.15: Fábrica de agentes implementada en Java. El tamaño de la cola de agentes activos es de 10.

Agentes instanciados	Persistencia	RAM	Tiempo creación (s)	Peticiones aleatorias	Time (s)
100	No	3.9 MB	0.072	25	0.007
1000	No	7.3 MB	0.805	250	0.082
10000	No	40.6 MB	7.764	2500	0.715

Tabla 5.16: Fábrica de agentes implementada en Python.

Agentes instanciados	Persistencia	RAM	Tiempo creación (s)	Peticiones aleatorias	Time (s)
100	Yes	2.0 MB	0.103	25	0.007
1000	Yes	2.1 MB	1.693	250	0.077
10000	Yes	2.2 MB	17.536	2500	0.670

Tabla 5.17: Fábrica de agentes implementada en C++. El tamaño de la cola de agentes activos es de 100.

Agentes instanciados	Persistencia	RAM	Tiempo creación (s)	Peticiones aleatorias	Time (s)
100	Yes	26.5 MB	0.304	25	0.020
1000	Yes	28.1 MB	2.509	250	0.148
10000	Yes	42.5 MB	22.070	2500	0.718

Tabla 5.18: Fábrica de agentes implementada en Java. El tamaño de la cola de agentes activos es de 100.

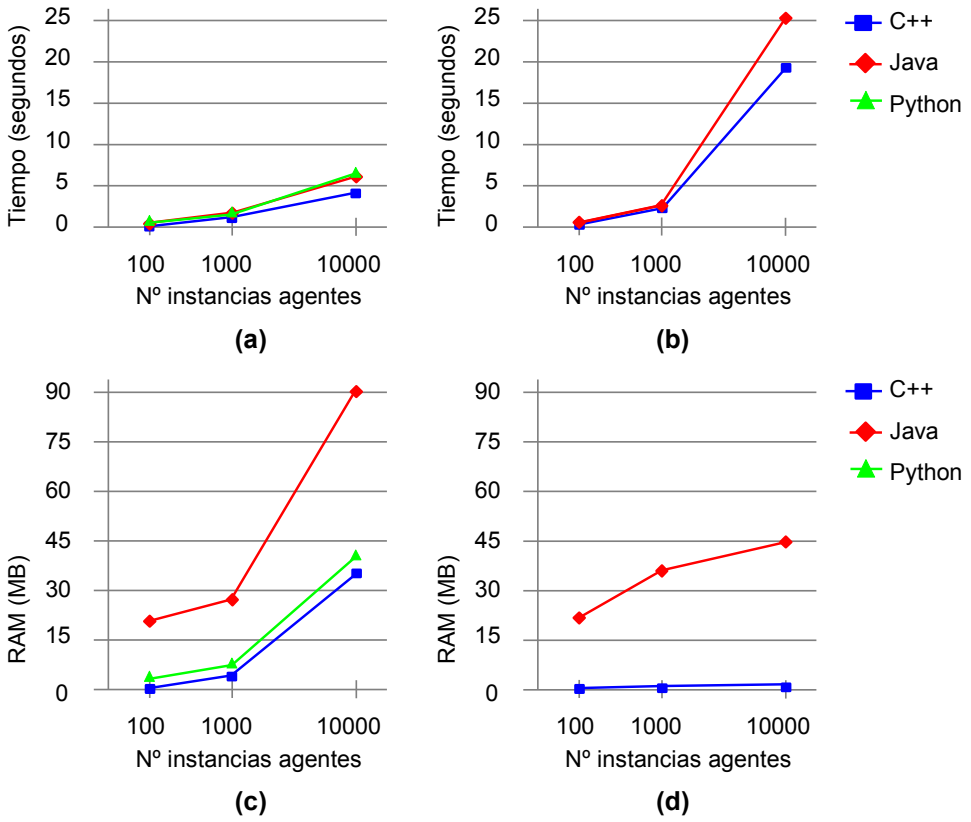


Figura 5.16: Arriba. Tiempo empleado cuando se crean agentes ((a)sin soporte de persistencia, (b) con persistencia). **Abajo**. RAM consumida cuando se crean agentes ((c)sin soporte de persistencia, (d) con persistencia).

5.4. Caso de estudio práctico de la plataforma multi-agente: despliegue de un sistema multi-agente para la coordinación de robots

En esta sección se describe el uso de la plataforma de agentes para el despliegue de un sistema multi-agente aplicado a la coordinación de robots [VRM+09]. Con el estudio de este caso práctico se pretende poner de manifiesto las ventajas que la plataforma general propuesta aporta a la hora de desarrollar soluciones basadas en agentes inteligentes. Así mismo, esta sección tiene como objetivo justificar la aplicación práctica de la plataforma en un problema real, haciendo especial hincapié en las facilidades que proporciona en el problema concreto de la coordinación de un conjunto de robots.

5.4.1. Motivación

La **coordinación de robots** (*Multi-Robot Coordination*) [FIN04] consiste en el uso de métodos y técnicas para que un conjunto de robots, normalmente con distintas habilidades, sean capaces de cooperar para alcanzar un objetivo global. Esta línea de investigación está estrechamente relacionada con la coordinación multi-agente [Woo01], donde los agentes software cooperan teniendo en cuenta al resto de agentes que conforman el sistema. La coordinación en el campo de la robótica ha de tratar con aspectos relevantes como la gestión de la incertidumbre, problemas con las comunicaciones físicas y restricciones asociadas a la propia física de los robots. Por lo tanto, la coordinación entre un grupo de robots que tienen un objetivo común en un entorno dinámica supone un reto dentro de esta línea de investigación.

En este contexto, los **Sistemas Multi-Agente** encajan perfectamente como solución a este problema, el cual involucra a un determinado número de fuentes heterogéneas de conocimiento (los robots) que cooperan para alcanzar una meta común. Por otra parte, el uso de protocolos de coordinación permite que los robots interactúen entre sí para tomar decisiones correctas. Dichos protocolos definen las reglas a seguir por los robots para ir cumpliendo objetivos parciales, como la asignación de roles, la distribución de tareas o incluso la exploración de entornos físicos.

En esta sección se describe un sistema multi-agente diseñado para abordar el problema de la coordinación de un conjunto de robots. Particularmente, el principal objetivo es el de establecer de manera dinámica equipos de agentes, donde cada agente controlará a un robot, para coordinarlos a la hora de desarrollar el proceso de exploración de un entorno. Para cumplir esta meta se ha hecho uso de un mecanismo de comunicación basado en eventos y se han diseñado dos protocolos de coordinación: uno para asignar roles y otro para recoger información del entorno. En este punto, el concepto de

movilidad se define para evaluar la eficacia de un robot para realizar una tarea en un instante de tiempo determinado, el cual se utilizará como medida fundamental a la hora de efectuar la cooperación entre robots.

A nivel de diseño y desarrollo, los agentes específicos creados para controlar a los agentes se han implementado como extensiones a los agentes de gestión proporcionados por la plataforma. Gracias a este enfoque, es posible heredar todos los mecanismos de comunicación incorporados a dichos agentes de gestión, además de la posibilidad de hacer uso de las fábricas de agentes para facilitar su despliegue.

A nivel de configuración y administración, se hace uso de la plataforma para definir la máquina en la que se ejecutarán los agentes, así como los recursos que utilizarán, además de monitorizar el funcionamiento de los mismos.

Finalmente, y a nivel de evaluación del sistema, el despliegue del sistema multi-agente se facilita gracias a las plantillas proporcionadas por la plataforma y al uso de descriptores facilitados por el middleware ZeroC ICE. Este hecho simplifica enormemente la fase de pruebas del sistema, ya que es posible parametrizar el sistema en función de diferentes cuestiones, como por ejemplo el número de robots utilizados en cada test.

5.4.2. Visión general del sistema

En la figura 5.17 se muestra la arquitectura del sistema multi-agente diseñado para abordar el problema de la coordinación multi-robot planteado en la sección 5.4.1. El nivel más alto desde el punto de vista organizacional es el equipo de agentes T, el cual a su vez se compone de una serie de células C_i ($C_i \in T$). Cada célula representa un grupo de agentes software, de manera que cada agente es el responsable de controlar un robot individual. La composición de las células se establece y se readaptan de manera dinámica, en función del estado actual del entorno en el que coexisten los robots.

Inicialmente, el entorno está compuesto por una serie de objetos O con posiciones estáticas y que no cambian a lo largo del tiempo. El principal objetivo de cada célula C_i es obtener la mayor cantidad de información posible del objeto $o_j \in O$. La monitorización de un objeto puede consistir en identificarlo o en realizar tareas más complejas.

Cada **célula** C_i está compuesta de una serie de agentes $A_i = \{l_i, E_i\}$, donde l_i representa al *líder* de la célula responsable de analizar o_i y E_i es el conjunto de *exploradores* asociados a C_i. El principal objetivo del líder l_i es el de coordinar a los agentes que forman la célula a la hora de monitorizar un entorno. Como se comentó anteriormente, existe un agente responsable de controlar a cada uno de los robots, actuando como su *proxy*.

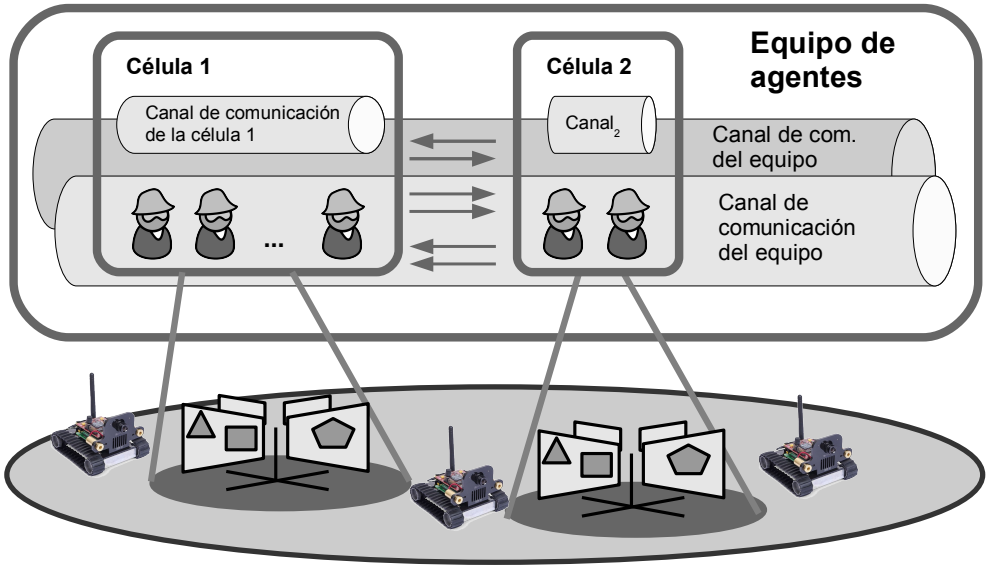


Figura 5.17: Vista abstracta del SMA para la coordinación multi-robot.

El estado de un agente en el instante de tiempo t se define como

$$a_j^i(t) =< p_{a_j^i}, \theta, R >$$

donde a_j^i denota al agente j en la célula C_i, $p_{a_i^j}$ es la posición del robot representado por el agente a_i^j, θ es el ángulo que define la orientación del robot en el sentido de rotación de las agujas del reloj, y R denota el conjunto de roles $r_i \in R$ del agente a_j^i.

5.4.3. Mecanismos de comunicación

La comunicación entre agentes se lleva a cabo mediante canales de eventos. La elección de este mecanismo de comunicación se debe a la flexibilidad proporcionada para tratar con el dinamismo del sistema multi-agente planteado: gestión de subastas, reconfiguración dinámica de las células y comunicación entre los agentes a nivel de célula. A partir de este enfoque se distinguen dos canales de comunicación:

- Canal de comunicación de **equipo**, el cual permite que los agentes de un mismo equipo publiquen y reciban mensajes. El principal objetivo de este canal es el de notificar información que concierna al equipo entero, como por ejemplo la elección de los líderes de las distintas celdas o la necesidad de explorar un determinado objeto o_j.

- Canal de comunicación de **célula**, creados por los líderes de cada célula con el objetivo de establecer una comunicación interna entre los agentes que componen la misma. Este tipo de canal se utiliza para coordinar el proceso de análisis de un objeto o_j por parte de una célula C_i.

Asignación de roles

Como se comentó anteriormente, los agentes asociados a una célula C_i pueden jugar el rol de explorador o líder (el cual es también de manera inherente un explorador). Para seleccionar el mejor candidato l_i responsable de analizar un objeto o_j se hace uso de un mecanismo simple de subasta con el objetivo de establecer los distintos líderes que llevarán a cabo el estudio del entorno. En este contexto, las pujas representan el valor de *movilidad* del robot representado por cada uno de los agentes. Así, valores altos de movilidad indican que un agente está preparado para liderar una célula y, por lo tanto, la exploración de un determinado objeto. A continuación se resumen los principales pasos de este protocolo de subasta.

1. **Anunciamiento de objetivos.** El primer paso consiste en obtener la posición de los objetos a estudiar, diferenciando dos soluciones posibles: i) usar una cámara externa que tenga una visión global del escenario o ii) hacer uso de las cámaras de los robots para construir la vista global del entorno. Una vez obtenida dicha información, uno de los agentes se proclama de manera aleatoria como el *gestor de subastas* y mediante un mensaje *Announcement* notifica a todos los agentes las posiciones de todos los objetos.

2. **Evaluación de la movilidad.** Cuando un agente recibe el mensaje generado en el paso anterior, evalúa la movilidad de su robot asociado respecto a todos los objetos con el objetivo de pujar por el objeto con el que mantiene una mayor movilidad. La movilidad se evalúa mediante una métrica $M =< V, DDV, R >$, donde $V = \{v_1, v_2, \ldots, v_n\}$ es el conjunto de variables de M, $DDV = \{ddv_1, ddv_2, \ldots, ddv_n\}$ son los rangos de definición para dichas variables, y $R = \{r_1, r_2, \ldots, r_n\}$ es el conjunto de reglas que determinan el valor de movilidad.

   Esencialmente, este conjunto de reglas tiene en cuenta la distancia del robot al objeto o_j y la autonomía actual del mismo para evaluar la movilidad. Para modelar tanto las variables como las reglas que determinan la movilidad se ha optado por usar un sistema difuso [Zad96]. Esta adopción se justifica por la incertidumbre presente en el entorno a explorar y por la interpretabilidad del conjunto de reglas R usado para definir el sistema. En la figura 5.18 se muestra la definición de las variables utilizadas y de algunas de las reglas que componen el sistema.

3. **Envío de pujas.** Una vez que la métrica ha sido aplicada, cada agente a_i envía su mejor puja b_i al gestor de subastas mediante un mensaje

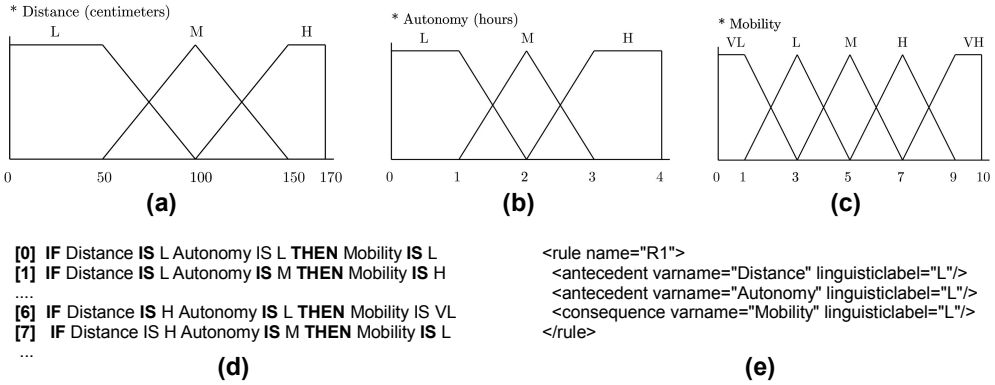


Figura 5.18: Métrica para evaluar la movilidad: **a)** definición de la variable *distance*, **b)** definición de la variable *autonomy*, **c)** definición de la variable *mobility*, **d)** algunas de las reglas del sistema difuso, **e)** definición XML de la regla *R1* (VL = very low, L = low, M = medium, H = high, VH = very high).

Bid. Una puja $b_i =< a_i, o_j, v >$ permite que dicho gestor obtenga la información necesaria para evaluar todas las pujas, donde v representa el valor de movilidad del agente a_i respecto al objeto o_j.

4. **Evaluación de pujas y asignación de objetivos.** El gestor de subastas ordena por movilidad las pujas recibidas y notifica qué agentes se han convertido en líder de algún objeto mediante un mensaje *Close*. Por ejemplo, si dos agentes a_i y a_j envían dos pujas $b_i =< a_i, o_1, v_i >$ y $b_j =< a_j, o_1, v_j >$ para el mismo objeto o_1 y $v_i > v_j$, entonces el agente a_i será el líder del objeto o_1. En este caso, a_j continuará pujando para ser el líder de otro objeto.

5. **Nueva iteración.** En cada iteración de este proceso se determinarán uno o más líderes. Sin embargo, en este paso puede haber objetos que no tengan líderes asignados, por lo que será necesaria otra iteración en el proceso de subasta. Esto puede ser debido a que varios agentes pujaran por el mismo objeto, el cual sólo puede tener asociado un único líder. Si el número de objetos n_o es menor que el número de robots n_r, entonces $n_r - n_o$ tendrán que unirse como exploradores a alguna de las células existentes, creadas por los líderes existentes. Actualmente, un explorador tratará de unirse a la célula cuyo objeto asociado mantiene la mejor relación de movilidad posible con dicho explorador.

Negociación para obtener información del entorno

Una vez que se han establecido las distintas células, el siguiente paso consiste en distribuir el estudio de un objeto o_j. En este punto, cada célula C_i conforma una entidad autónoma que usa su propio canal de comunicación para coordinar dicho proceso. Actualmente, los exploradores de una célula hacen uso de un **protocolo** simple para rodear el objeto que han de estudiar.

Este método consiste en alcanzar la posición más cercana al objeto en función del número de exploradores y la movilidad de cada explorador respecto a las posiciones que permiten rodear dicho objeto. La unión de estas posiciones forma una circunferencia en la cual el radio es la posición del objeto a analizar.

Por otra parte, es posible que un explorador cambie de célula para seguir estudiando el entorno. Actualmente, el valor que condiciona este cambio es el número de componentes de una célula. Si este valor es mayor que un umbral, el líder envía un mensaje de difusión mediante el canal del equipo para encontrar una célula que tenga pocos exploradores. Si dos o más líderes responden, el explorador se unirá a la célula con menor número de exploradores.

5.4.4. Evaluación experimental

Para llevar a cabo la evaluación del sistema multi-agente se utilizaron cuatro robots *Surveyor SRV-1 Blackfin*, como se puede apreciar en la figura 5.19.d. El entorno elegido para probar el sistema fue un escenario construido de $1,2$ x $1,3$ metros de superficie, tal y como se puede ver desde la cámara de un robot en las figuras 5.19.e y 5.19.f. En dicho escenario se desplegaron una serie de objetos O donde cada objeto $o_i \in O$ estaba compuesto por una serie de figuras geométricas pegadas sobre una de sus caras.

Con el objetivo de que fuesen capaces de detectar estas figuras y, por lo tanto, llevar a cabo el análisis de los distintos objetos que componen el entorno, los robots fueron entrenados previamente a su despliegue utilizando una **red neuronal** mediante el método de retro-propagación (*back propagation*) [HN89]. La entrada de dicha red fue una representación gráfica de cada objeto a través de una matriz de píxeles y la salida fue la distancia del robot al objeto, el ángulo de visión y el identificador del objeto a reconocer. Con esta información se pretendía que los robots fueran capaces de identificar los distintos objetos y estimar a qué distancia y con qué orientación estaban de los mismos a partir de las imágenes tomadas en el propio escenario. Este proceso de aprendizaje consistió en tomar 36 imágenes desde la cámara de un robot variando la orientación y la posición del mismo, como se puede apreciar en la figura 5.19.d.

El software utilizado por una cámara cenital colocada a 1,5 metros del escenario para obtener las posiciones de los robots y los objetos en cada momento del análisis se implementó en C++ haciendo uso de Roborealm[5], mientras que los agentes se implementaron en Python, al igual que el software de gestión de la red neuronal utilizada.

Finalmente, el funcionamiento del sistema multi-agente se simuló me-

[5]http://www.roborealm.com

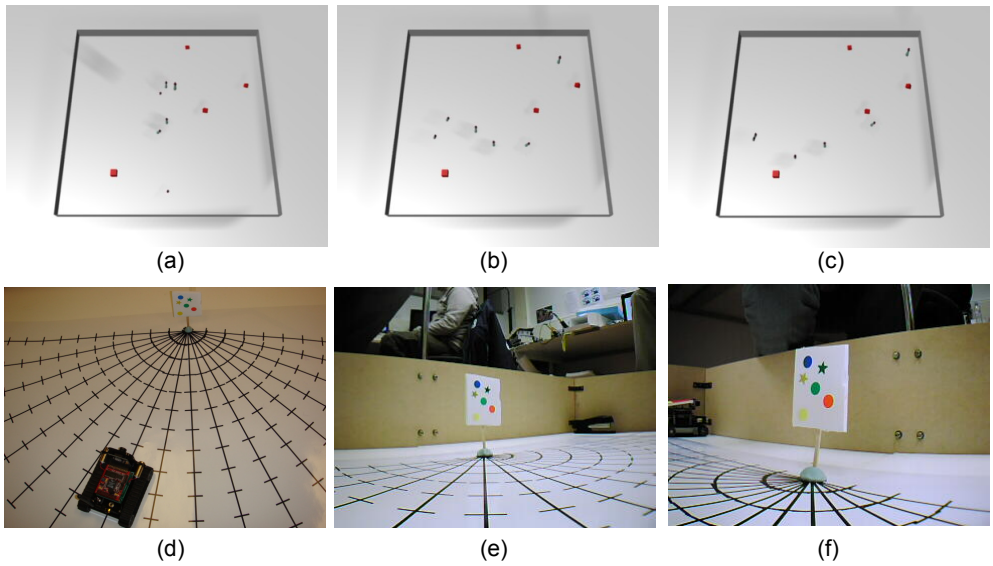


Figura 5.19: (a)(b)(c). Simulación del sistema multi-robot mediante una herramienta gráfica. **(d)**. Aprendizaje de patrones. **(e)(f)**. Imágenes obtenidas desde la cámara del robot.

diante una herramienta gráfica, como se puede observar en las figuras 5.19.a, 5.19.b y 5.19.c. Además, también se realizaron pruebas sobre el escenario real variando el número y la posición de tanto los robots como los objetos que componían el entorno a estudiar.

5.4.5. Despliegue del sistema multi-agente

En esta sección se describirá cómo se ha utilizado la plataforma de agentes para desplegar el sistema multi-agente concebido para coordinar el conjunto de robots. El objetivo es poner de manifiesto las acciones específicas que han de realizarse para extender la funcionalidad de la plataforma en un dominio específico. En la figura 5.20 se muestra el diagrama de clases que refleja los distintos componentes que forman parte del sistema multi-agente. En la parte superior se pueden ver los componentes integrados en la plataforma, mientras que en la parte inferior se denotan las clases específicas del sistema de coordinación multi-robot.

En primer lugar es necesario identificar los agentes que compondrán el sistema multi-agente, los cuales heredan la funcionalidad de gestión del agente proporcionado por la plataforma. En este caso en particular, dichos agentes son el **explorador** y el **agente de cámara**. Las instancias derivadas de este primer tipo de agente serán las encargadas de controlar a los distintos robots físicos desplegados en el entorno. Así mismo, se denota el agente **líder** como caso particular del explorador para cubrir las necesidades

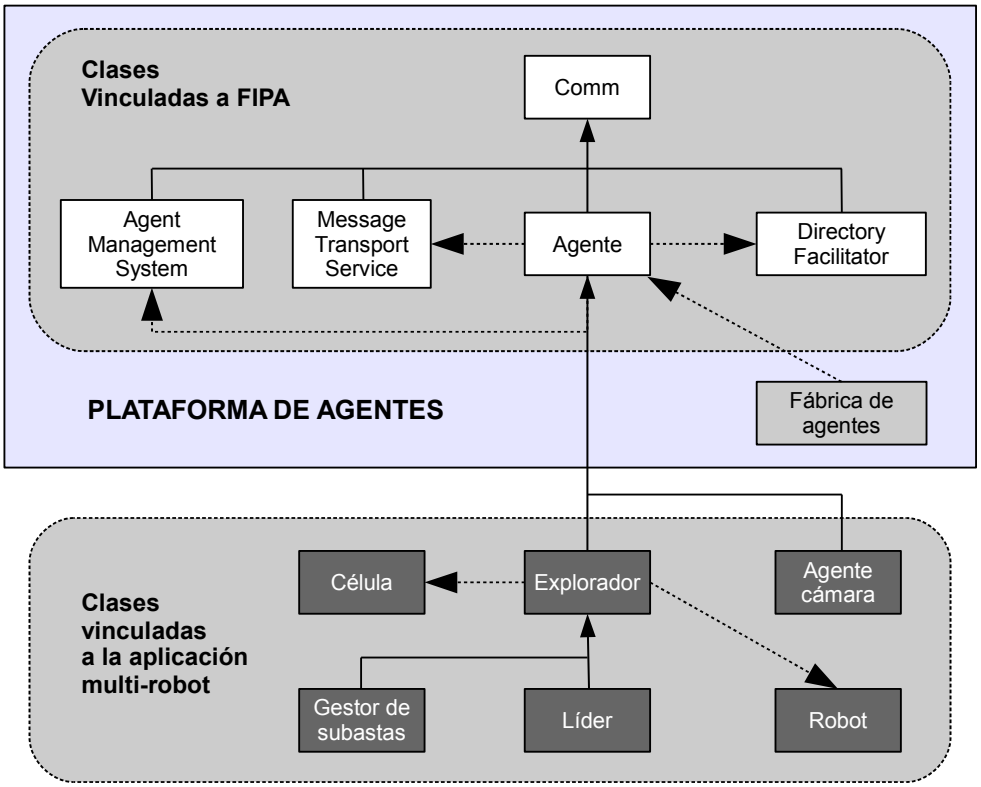


Figura 5.20: Diagrama de clases del sistema multi-agente para la coordinación multi-robot desplegado mediante la plataforma de agentes propuesta.

previamente discutidas en la sección 5.4.2 y el agente **gestor de subastas** como responsable de coordinar el proceso de coordinación en tareas como la elección los líderes de cada célula.

El otro tipo de agente que se observa en la figura 5.20 es el agente de cámara, que básicamente es un agente reactivo encargado de proporcionar las posiciones de los robots y de los objetos cuando algún otro agente así lo solicite. Parte del software asociado a este agente es el encargado de analizar las imágenes tomadas desde la cámara cenital situada sobre el entorno. Finalmente, el diagrama muestra otras clases (*célula* y *robot*) vinculadas a otros elementos del sistema pero que no representan a agentes propiamente dichos.

Después de identificar a los agentes del sistema, es necesario definir las interfaces que ofrecerán al sistema, es decir, la funcionalidad que *a priori* pueden proporcionar al resto de elementos de la plataforma. En principio, tanto los exploradores como los líderes son capaces de llevar a cabo sus tareas sin la necesidad de definir nueva funcionalidad, ya que sólo necesitan enviar y recibir mensajes para participar en los protocolos de coordinación discutidos en la sección 5.4.3. Por lo tanto, no es necesario extender la fun-

cionalidad de estos dos tipos de agentes a nivel de interfaz, ya que la recepción de mensajes está gestionada por el agente FIPA, aunque sí incluir el código asociado al procesamiento del contenido de los mensajes.

Sin embargo, tanto el agente gestor de subastas (*Auctioneer*) como el agente de cámara (*Camera*) ofrecen nueva funcionalidad respecto al agente de gestión FIPA facilitando la elección de líderes y proporcionando información del entorno al resto de agentes, respectivamente. Todas estas cuestiones quedan reflejadas en el listado que se muestra a continuación.

Interfaces del sistema de coordinación multi-robot

```
#include <Ice/Identity.ice>

#include <FIPA.ice>

module Robots {

  interface Explorer extends FIPA::Agent;
  interface Leader extends Explorer;

  interface Auctioneer extends Explorer {
    ["ami"] void begin ();
    ["ami"] void receiveBid (Bid b);
    Agent* getLeader (SignPost sp);
  };

  exception AlreadySubscribed {};

  interface Camera extends FIPA::Agent{
    idempotent RobotPos getRobotPos (Ice::Identity robot);
    idempotent SignPostSeq getSignPostPositions ();

    void subscribe (Agent* a) throws AlreadySubscribed;
    void unsubscribe (Agent* a);
  };

  interface Robot {
    ["ami"] void move (RobotPos to,
    Speed sp);

    Ice::ByteSeq capture (Resolution res);

    void setLaser (bool on);
    void destroy ();
  };

};
```

Después de identificar y diseñar las interfaces de los agentes que forman parte del sistema, el siguiente paso consiste en implementar la funcionalidad específica de los mismos. En este punto, es importante recordar que la funcionalidad del agente básico de gestión y del resto de elementos de acuerdo a la propuesta de FIPA está soportada por la plataforma de agentes, por lo que se pueden ver como componentes listos para utilizarse. En el siguiente listado y como ejemplo particular se muestra parte de la implementación del agente **explorador** (*ExplorerI*) con el lenguaje de programación Python. En

este listado se puede apreciar cómo dicha implementación (sirviente en la tecnología *middleware*) proporciona la funcionalidad mostrada en la interfaz de dicho agente en el listado anterior.

Parte de la implementación del agente explorador

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

class ExplorerI (Robots.Explorer):

  def __init__ (self, name, adapter, team, robot, ...):
    # Agent's identity.
    self.__name = name
    self.__team = team
    # ...

  def processMessage (self, message):

    # Announcement --> look for a signpost to lead.
    if isinstance(message, Robots.Announcement) and
      self.__signpost == None:

      mob = -1
      key = -1

      # Look for the best signpost to move.
      for k, s in self.__signposts.iteritems():
        for x in message.signposts:
          if s == x and mob < self.__mobilities[k]:
            mob = self.__mobilities[k]
            key = k

      # Bid for the signpost with the best mobility.
      if key != -1:
        self.__auctioneer.receiveBid_async
          (AMI_Model_receiveBidI(),
           Robots.Bid(self.__proxies['self'],
           self.__signposts[key],
           mob))

    # Close --> it becomes a leader. Go to the signpost.
    # ...
```

Finalmente, es necesario desplegar tanto los distintos componentes de la plataforma como los agentes específicos del dominio de la aplicación. El mecanismo más aconsejable, debido a la facilidad de uso y de administración, consiste en utilizar el servicio IceGrid proporcionado directamente por el middleware ZeroC ICE, el cual da soporte a la plataforma de agentes. Como se discutió en la sección 4.6, esta operación se puede llevar a cabo mediante la creación de un descriptor en el que se especifiquen qué componentes se despliegan y sobre qué nodos y máquinas se ejecutarán.

5.5. Uso de la plataforma multi-agente en otros dominios de aplicación

5.5.1. Renderizado distribuido

La generación de una imagen bidimensional a partir de la definición abstracta de una escena tridimensional es un proceso que está compuesto de ciertas etapas como el modelado, la aplicación de materiales y texturas, el efecto de fuentes de luz virtual y, finalmente, el renderizado (*rendering*) [Ker04]. Los **algoritmos de renderizado** toman como entrada la descripción de la geometría, materiales, texturas, fuentes de luz y la cámara virtual para producir una imagen (o secuencia de imágenes en el caso de una animación) como salida. Existen distintos algoritmos de renderizado, desde los más simples y menos costosos en términos computacionales hasta aquellos más complejos y reales que simulan la naturaleza de la luz con una gran precisión.

En este contexto, el renderizado fotorrealista de escenas complejas supone uno de los retos actuales de los Gráficos por Computador. Sin embargo, este proceso es extremadamente costoso en tiempo de CPU requiriendo, especialmente, una gran cantidad de tiempo cuando el algoritmo de renderizado hace uso de técnicas de iluminación global [Jen04]. En función del método empleado y de las características de la escena a renderizar, la generación de una única imagen puede tardar horas o días incluso en máquinas potentes. De manera adicional, la correcta configuración de los distintos parámetros involucrados en el renderizado es una tarea compleja. De hecho, el usuario del motor de renderizado tiende a utilizar parámetros con valores elevados que no mejoran la calidad final de la imagen generada pero que elevan considerablemente el tiempo utilizado para obtener el resultado final, es decir, la imagen bidimensional.

En la literatura existen distintas **alternativas** para reducir el tiempo empleado en el proceso de renderizado, las cuales se pueden clasificar en tres grandes grupos:

- **Optimización mediante hardware especializado**, representado principalmente por el uso de unidades de procesamiento gráfico (*graphics processing units* o *GPUs*). Dichas unidades se encuentran integradas en la mayoría de ordenadores de escritorio y que se pueden utilizar para ejecutar el código de un *trazador de rayos* de manera paralela [CPC84]. El uso de *GPUs* puede mejorar el rendimiento de las actuales *CPUs* en un factor de siete, aproximadamente [BFH+04].
- **Optimización mediante computación paralela**, representado principalmente por los *clusters* de renderizado. En este contexto, se suele hacer uso de un *cluster* de computadores [Pfi98], normalmente con las

mismas características hardware, para distribuir el renderizado de las distintas imágenes o *frames* que componen una animación en distintas máquinas físicas.

- **Sistemas multi-agente**, enmarcados dentro del área de la Inteligencia Artificial Distribuida y utilizados no sólo para distribuir el proceso de renderizado sino también para optimizar y ajustar parámetros relevantes del mismo.

En esta sección se describe brevemente *MAgArRO*[6], el sistema multi-agente concebido para optimizar el proceso de renderizado reduciendo el tiempo final empleado en dicha tarea [GMWJ+07, GMWJV07]. Dicho sistema hace uso de la plataforma de agentes discutida en el capítulo 4 para el desarrollo y despliegue de aplicaciones multi-agente, facilitando las tareas de gestión y comunicación mediante los agentes *Agent Management System*, *Directory Facilitator* y *Message Transport System* (ver sección 4.5), diseñados de acuerdo a las especificaciones del comité FIPA (ver sección 2.1.4).

Como se puede apreciar en la figura 5.21, *MAgArRO* está compuesto por los siguientes agentes y servicios:

- ***Master***, agente encargado de gestionar las tareas que llegan al sistema y de coordinar a los agentes de renderizado para obtener resultados.
- ***Analyst***, agente especializado en estudiar la complejidad de la escena a renderizar y establecer la división en unidades de trabajo.
- ***Rendering Agent***, agente encargado de renderizar unidades de trabajo en base a la definición previa de conocimiento experto, con el objetivo de optimizar el ajuste de parámetros del motor de renderizado utilizado.
- ***Model Repository***, servicio que da soporte al almacenamiento de proyectos de renderizado para que los distintos agentes involucrados puedan acceder a los mismos.
- ***Blackboard***, servicio que permite compartir conocimiento entre los *rendering agents* que realizan el renderizado de un proyecto.

A continuación se describen brevemente los pasos que definen el flujo de trabajo de *MAgArRO* (representados por círculos numerados en la figura 5.21):

1. El sistema recibe un nuevo trabajo mediante la interfaz pública del agente *analyst*. No obstante, la interacción con el usuario se realiza mediante un sistema web. Previamente, los *rendering agents* se han registrado con alguno de los *masters* desplegados en el sistema, conformando dinámicamente distintos grupos o equipos de renderizado distribuido.

[6] http://www.inf-cr.uclm.es/www/cglez/magarro/

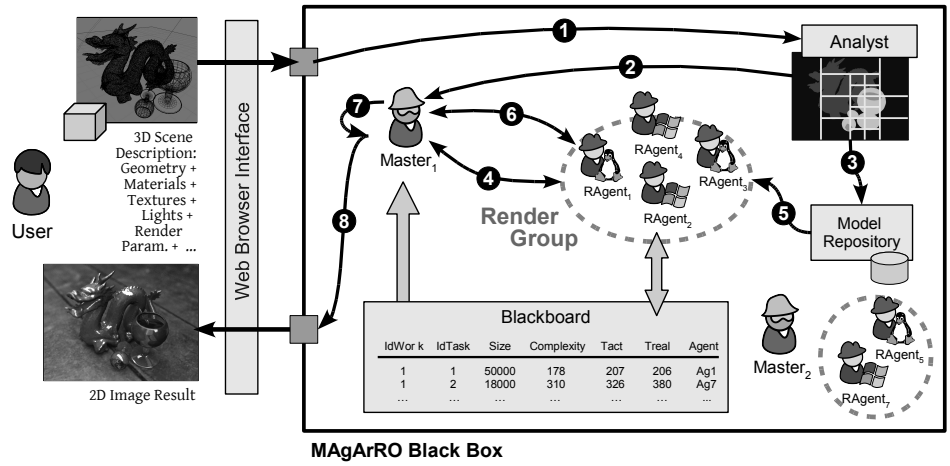


Figura 5.21: Descripción visual del flujo de información entre los distintos componentes de *MAgArRO*.

2. El agente *analyst* realiza un análisis de la escena a renderizar con el objetivo de dividirla en unidades de trabajo que mantengan una complejidad similar (en términos de tiempo empleado para renderizarlas).
3. El *analyst* envía el proyecto al *model repository* y notifica la existencia de dicho proyecto a un *master*.
4. El *master* notifica el proyecto al grupo de *rendering agents* que tiene asociado y que, posiblemente, están ejecutándose en distintas máquinas físicas.
5. Cada uno de estos *rendering agents* obtiene el modelo tridimensional del *model repository*. A continuación, comienza un proceso de subasta coordinado por el *master* en el que los *rendering agents* pujan por las unidades de trabajo.
6. Cuando un *rendering agent* ha terminado de renderizar una unidad de trabajo, vuelve a pujar por otra disponible hasta que todas las unidades se hayan renderizado.
7. El *master* compone el resultado final, es decir, la imagen bidimensional, a partir de los resultados parciales (unidades de trabajo) generados por los *rendering agents*.
8. El *master* envía la imagen final al usuario que solicitó el renderizado de la escena tridimensional.

En la figura 5.22 se muestran visualmente los resultados obtenidos por *MAgArRO* a la hora de renderizar una escena compleja, mientras que en la figura 5.22 se resumen los tiempos empleados en renderizar dicha escena variando el número de recursos (*rendering agents*) utilizados para dicha tarea.

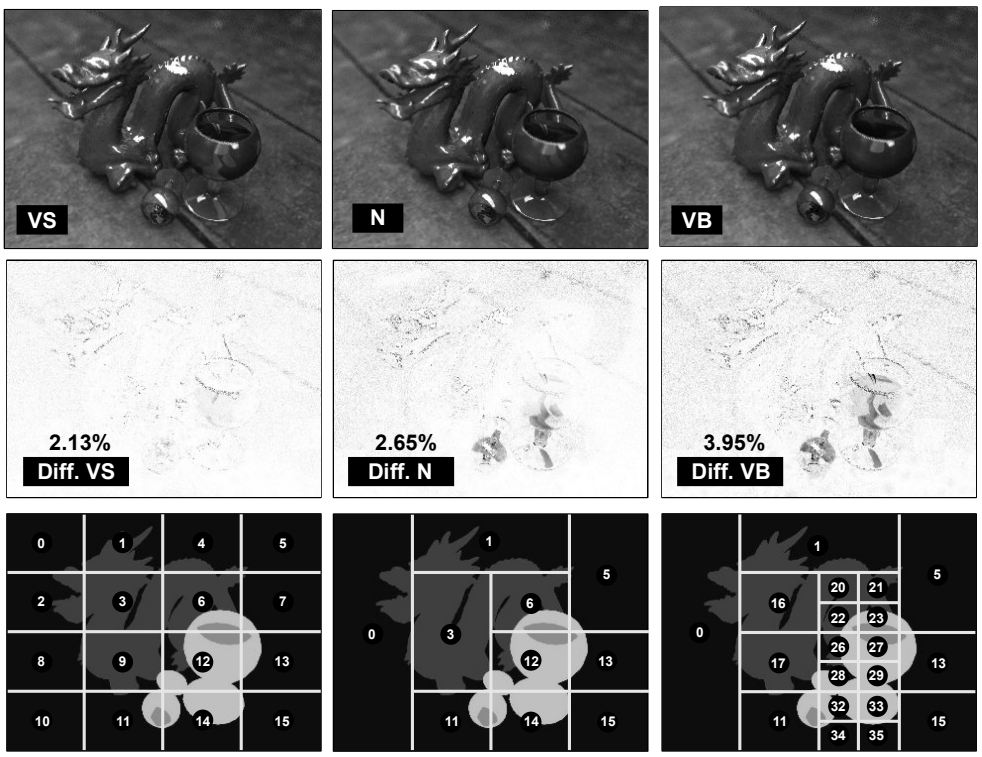


Figura 5.22: Resultados obtenidos tras utilizar *MAgArRO* para renderizar una escena compleja. **Primera fila**: imágenes generadas con distintos niveles de optimización (VS, *very small*; N, *normal*; VB, *very big*). **Segunda fila**: diferencia de calidad entre la imagen renderizada sin emplear el sistema y haciendo uso del mismo con distintos niveles de optimización. **Tercera fila**: distintos niveles de particionado manejados por el agente *analyst* para dividir la escena en unidades de trabajo.

5.5.2. Comercio electrónico

La consolidación de la web como tecnología del día a día ha generado un entorno altamente competitivo en el que las compañías desarrollan y extienden sus procesos de negocio [LL00] para tratar con clientes de cualquier lugar del mundo [TLKC00]. En los últimos años, esta competitividad ha fomentado la investigación en el desarrollo de la infraestructura que da soporte a este nuevo paradigma de negocio, comúnmente conocido como Comercio Electrónico (*Electronic Commerce* o *E-Commerce*). Dicho paradigma se puede definir como cualquier tipo de negocio que se efectúa utilizando medios electrónicos, como por ejemplo Internet u otra red informática.

A partir de estas perspectivas de negocio han surgido distintas ramas, como el *C2C (Consumer-to-Consumer) e-commerce*, en el que las transacciones se realizan entre consumidores que negocian directamente para alcanzar algún tipo de acuerdo. El ejemplo más popular es la subasta electrónica, la

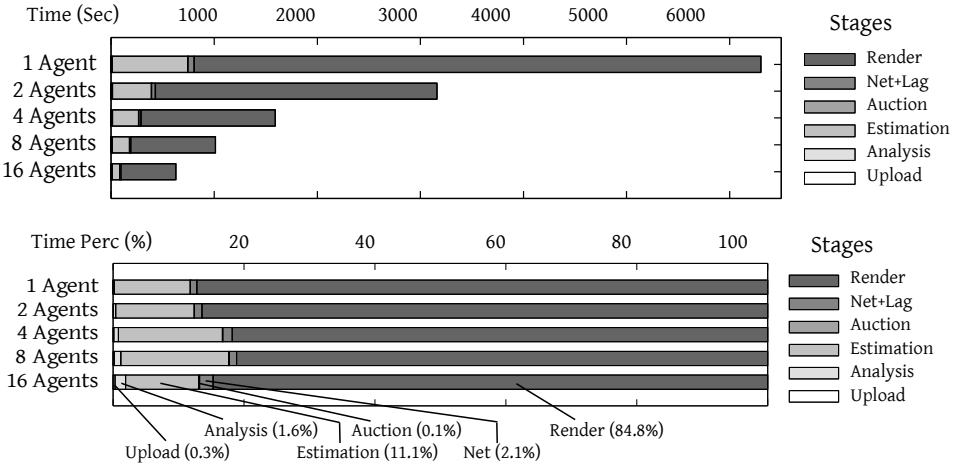


Figura 5.23: Resumen de tiempos empleados para renderizar la escena del dragón de la figura 5.22 variando el número de *rendering agents*. **Arriba**: tiempo empleado en cada una de las etapas del proceso de renderizado distribuido. **Abajo**: porcentaje de tiempo dedicado a cada una de dichas etapas.

cual solventa las restricciones geográficas y temporales de las subastas tradicionales. Tras el éxito del modelo C2C, las empresas comenzaron a introducir la venta directa en este tipo de infraestructura, utilizando los portales C2C como portales *B2C (Business-to-Consumer)*.

Debido a que el origen de muchos portales de comercio electrónico fueron las subastas, la mayoría de ellos hacen uso de descripciones lexicográficas y listas de objetos para gestionar el catálogo de productos. Este hecho implica que el usuario ha de proporcionar una lista de palabras clave para definir un criterio de búsqueda, lo cual puede no ser apropiado dentro del paradigma de compra directa debido a problemas como la polisemia y la complejidad asociada a automatizar la comparación y recomendación de productos. En otras palabras, los vendedores conocen perfectamente los productos que ponen a la venta, pero los compradores no suelen tener un conocimiento preciso de lo que quieren adquirir. Este problema se traduce en el auge de productos populares que no tienen por qué ser la mejor opción de compra desde el punto de vista de la relación calidad/precio.

Una posible solución a este problema se planteó en [LLCS+09], caracterizado por un sistema de catálogo jerárquico para ordenar productos de acuerdo a una serie de características, un sistema de selección de productos basado en la definición de criterios de búsqueda vagos e imprecisos y un sistema de recomendación de productos basado en el conocimiento del mercado.

Desde un punto de vista general, el comercio electrónico supone un problema complejo y abierto en el que distintas entidades cooperan con el objeti-

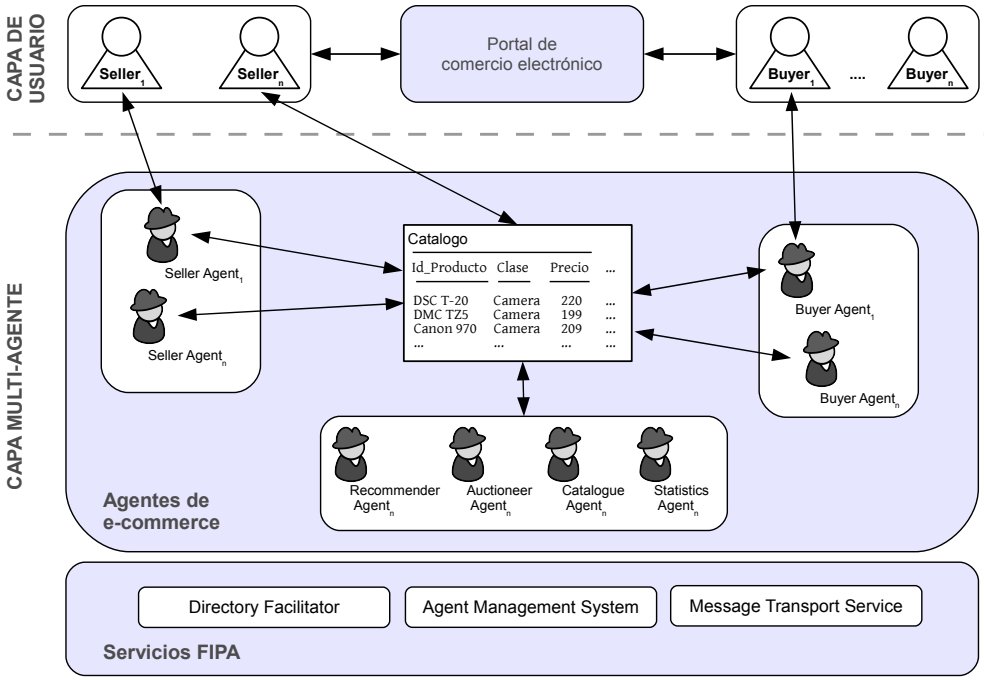


Figura 5.24: Vista general de la arquitectura del sistema multi-agente que da soporte al portal de comercio electrónico [LLCS+09].

vo de negociar entre ellas y obtener un beneficio mutuo. Este hecho conduce a las soluciones basadas en sistemas multi-agente como soporte natural a las actividades de negocio llevadas a cabo por los distintos usuarios del sistema de comercio electrónico [HJL03].

El sistema multi-agente diseñado para dar soporte al proceso de comercio electrónico de acuerdo con las necesidades planteadas anteriormente se muestra en la figura 5.24. Dicho sistema está soportado por la plataforma de agentes descrita en el capítulo 4. A continuación se describe brevemente la funcionalidad de los distintos agentes que proporcionan un determinado beneficio a los usuarios (tanto compradores como vendedores) del portal de comercio electrónico:

- ***Catalogue Agent***, responsable del mantenimiento del catálogo de productos del portal y de informar sobre su contenido. Sus principales funciones son las de garantizar la integridad del catálogo y de interactuar con el administrador del portal para facilitar su expansión.
- ***Product Selection and Recommender Agent***, encargado de gestionar la selección de productos y los mecanismos de aprendizaje asociados a la recomendación de productos. Su principal función es la de analizar descripciones de los usuarios y buscar los productos existentes en el

que catálogo que más se asemejen a dichas descripciones para ofrecer recomendaciones de calidad.

- ***Auctioneer Agent***, responsable de iniciar y controlar las subastas en el portal, garantizando la correcta finalización de las mismas.

- ***Seller Agent***, agente que representa al usuario vendedor dentro del portal. Su principal responsabilidad es la de interactuar con el resto de agentes del sistema para beneficiar al usuario vendedor al que representa.

- ***Buyer Agent***, agente que representa al usuario comprador a la hora de adquirir productos en los que esté interesado en base a las preferencias de dicho usuario.

5.6. Relevant contributions to scientific journals and international conferences

As a result of part of the work carried out during the development of this dissertation, the more relevant scientific contributions that have been done are summarised here:

- **D. Vallejo**, J. Albusac, L. Jimenez, C. Gonzalez, J. Moreno, 2009, *A Cognitive Surveillance System for Detecting Incorrect Traffic Behaviors*, Expert System with Applications, Elsevier, 36(7), p. 10503-10511, [**Impact Factor: 2.596 in 2008**].

  This paper discusses the traffic control system used to establish a first approximation of the multi-agent architecture discussed in section 3.1. This work allowed us to introduce the need for a flexible and scalable architecture for intelligent surveillance systems.

- **D. Vallejo**, J. Albusac, J.A. Mateos, L. Jimenez, C. Gonzalez, 2009, *A Modern Approach to MultiAgent Development*, Journal of Systems and Software, Elsevier. IN PRESS [**Impact Factor: 1.241 in 2008**].

  This paper discusses the design and development of the agent platform studied in Chapter 4. This platform has been used in different application domains, such as Distributed Rendering [GMWJV07, GMWJ+07], Electronic Commerce [VCSAJGM08], Multi-Robot Coordination [VRM+09] or Virtual Reality [Mat09].

- **D. Vallejo**, J. Albusac, J.J. Castro-Schez, C. Glez-Morcillo, L. Jiménez, 2009, *A Multi-Agent Architecture for Supporting Distributed Normality-based Intelligent Surveillance* , Engineering Applications of Artificial Intelligence, Elsevier. UNDER REVIEW.

This paper summarises the multi-agent architecture for intelligent surveillance systems studied in Chapter 3. Some initial ideas were submitted to the Twelfth International Workshop CIA 2008 on Cooperative Information Agents [VAGMJ08].

- J. Albusac, **D. Vallejo**, L. Jimenez-Linares, J.J Castro-Schez, L. Rodriguez-Benitez, 2009, *Intelligent Surveillance based on Normality Analysis to Detect Abnormal Behaviors*, International Journal of Pattern Recognition and Artificial Intelligence, Special issue on Visual Analysis and Understanding for Surveillance Applications, World Scientific, 23(7), p. 1223-1244. [**Impact Factor: 0.66 in 2008**].

  This paper describes the normality-based surveillance model that inspired the multi-agent architecture for intelligent surveillance proposed in this dissertation. This paper also covers how to design a particular surveillance concept by using such model.

- J. Albusac, J.J. Castro-Schez, L.M. López-López, **D. Vallejo**, L. Jimenez, 2009, *A Supervised Learning Approach to Automate the Acquisition of Knowledge in Surveillance Systems*, Signal Processing, Special issue on Visual Information Analysis for Security, Elsevier. 89(12), p. 2400-2414, [**Impact Factor: 1.256 en 2008**].

  This paper introduces the application of a supervised learning approach to automate the acquisition of knowledge in surveillance systems.

- J. Albusac, **D. Vallejo**, J.J. Castro-Schez, P. Remagnino, C. Glez-Morcillo, L. Jiménez, 2009, *A knowledge-driven approach based on intelligent agents to monitor complex environments*, IEEE Intelligent Systems, Special Issue on Intelligent Monitoring of Complex Environments. UNDER REVIEW.

  This paper describes a multi-layered component-based model devised to design and develop intelligent surveillance systems.

- C. González-Morcillo, G. Weiss, **D. Vallejo**, L. Jiménez-Linares, J.J. Castro-Schez, 2009, *A Multi-Agent Architecture for 3D Rendering Optimisation*, Applied Artificial Intelligence, Tailor & Francis. UNDER REVIEW. Some initial ideas were submitted to the Nineteenth Conference on Innovative Applications of Artificial Intelligence [GMWJV07].

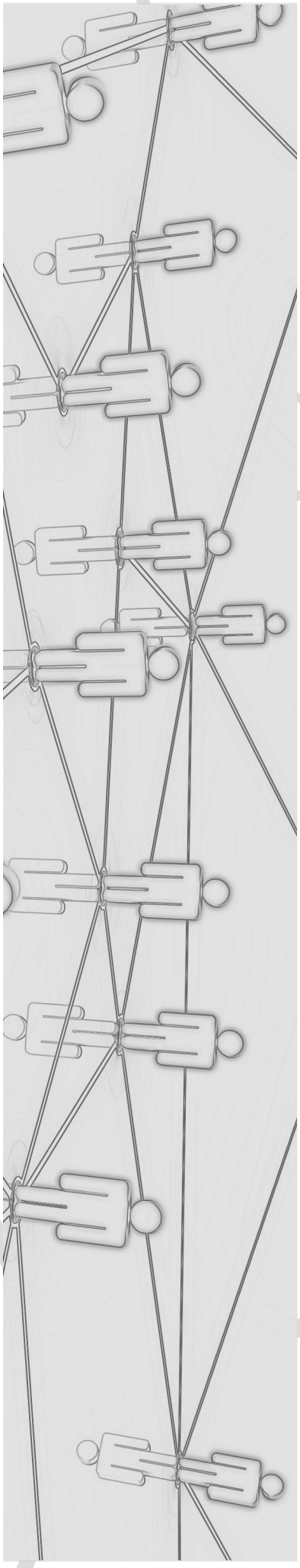
  This paper discusses a modern approach for distributed rendering based on agent technology and expert knowledge to reduce the time spent in photorealistic 3D rendering.

- L.M. López-López, J.J. Castro-Schez, **D. Vallejo**, 2009, *A Highly Adaptive Recommender System based on Fuzzy Logic for B2C e-Commerce Portals*, Expert Systems with Applications, Elsevier. UNDER REVIEW.

  This paper describes a highly adaptive recommender system based on fuzzy logic for B2C e-commerce portals. The infrastructure that gives support to this system is a multi-agent architecture.

Chapter 6

Conclusions and Future Work



Chapter 6
Conclusions & Future Work

This chapter discusses the conclusions obtained from the development of this dissertation, establishing the main future research lines devised from the elaboration of this dissertation.

"Nothing important happened today."
George III of the United Kingdom (4 July 1776)

6

Conclusions and Future Work

6.1. Summary of achievements

This thesis has discussed the multi-agent architecture proposed to give support to the design and development of intelligent surveillance systems, which has been inspired by a scalable and general normality analysis model [AVJL+09]. The agents that compose the architecture have been conceived to cover the needs posed by the model in which it is inspired and the communication mechanisms needed for the agents to carry out a global surveillance of the environment have been defined.

Initially, an in-depth review of the state of the art was performed in order to analyse existing proposals and identify the needs devised by our concept of surveillance model. This task was accomplished by studying the evolution of surveillance systems and the current approaches to deal with the design and development of surveillance architectures. Furthermore, the main concepts of Distributed Artificial Intelligence, Intelligent Surveillance, Service-Oriented Architectures, and Communication Middlewares were studied.

The proposed architecture has a multi-layer design that abstracts the services provided in its different levels:

- **Reactive level**, which represents the sensory inputs of the surveillance system and hosts the agents that maintain a reactive model, such as the *preprocessing agents*.
- **Deliberative level**, which is composed of the agents that maintain a knowledge-based model and give support to the tasks of analysis, understanding and monitoring of environments.
- **User level**, which represents the interface with the security personnel via monitoring and knowledge acquisition tools.

This architecture has been designed to achieve the generality and scalability needed to deploy particular multi-agent surveillance systems in a wide variety of environments. These desirable properties have been partly acquired because the architecture is inspired by a formal model based on the normality analysis that is general, domain-independent, and allows to model surveillance concepts when they are needed. However, different surveillance models can be supported by the architecture.

The main advantages of the proposed architecture are as follows:

- **Generality**, since it allows the deployment of agents that manage the knowledge derived from the design of surveillance components in any environment.
- **Scalability**, since new agents can be added to the surveillance system in order to deal with a surveillance task that was not initially considered, such as the preprocessing of the information gathered by new sensors or the need of including new analysis components into the environment. The architecture also maintains a multi-layer design, which allows to structure the functionality of the system into independent layers.
- **Agent-based** approach, which facilitates the design of complex distributed systems where different autonomous entities cooperate to reach a common goal. In our case, this goal is the surveillance of complex environments.
- **Distributed control**, which makes possible the development of robust and adaptable solutions instead of using centralised approaches.

The implemented prototype of the surveillance architecture has been used to deploy a surveillance system in a common urban traffic environment in order to analyse the trajectories and speed of moving objects. This environment has been structured into three different sub-environments to simplify the monitoring process and to justify how the architecture gives support to the agents and communication mechanisms needed to carry out the global surveillance. Both the processing and communication times spent by the different agents prove the feasibility of the proposed architecture.

The process of integrating a particular *normality agent* has been discussed to illustrate the steps needed to deploy a surveillance agent responsible for analysing a certain kind of concept or event of interest.

Some of the design solutions devised for the abstract surveillance architecture have been generalised and extended in order to design and develop an agent framework that can be used to deploy and manage multi-agent systems. Within this context, a modern agent platform [VAM+09] to develop MAS according to FIPA specifications [Fou04] has been presented. The main contribution of this work is to provide a new approach extensible to a wide

range of platforms and domains when developing MAS. To do that, a software architecture based on design patterns and templates on top of a modern middleware, which covers a high number of operating systems, programming languages, hardware devices, and communication networks, has been designed and developed. The main characteristics of this agent platform are as follows:

- Based on the set of **FIPA standards**.
- **Automatic deployment** of FIPA services, which are highly configurable depending on the application constraints.
- Use of a **modern middleware** interoperable with a high number of programming languages, operating systems and hardware platforms.
- **Ready-to-use agents** developed in different programming languages.
- Integration with an administrative web system to remotely manage multi-agent applications.
- Scalability, robustness, authentication, encryption, and message integrity.

To evaluate the agent platform, an extensive set of experiments has been carried out to mainly test the efficiency and the scalability of the agent platform. The results show that the system offers good response-times when the load of the multi-agent application becomes relevant (as discussed in Section 5.3.1) and suggest that this kind of tools must be as configurable as possible to adequate the system deployment to the application requirements. Besides, a multi-agent application example has been described in order to expose the agent platform features and how the development of new tools can help to deploy and manage MAS.

6.2. Discussion

In the last years, Intelligent Surveillance has become very important due to the need of deploying systems that monitor and analyse both public and private environments where the information sources are distributed around. Within this context, reducing the number of criminal actions in both indoor and outdoor scenarios makes essential the design and development of sophisticated surveillance systems.

Intelligent Surveillance involves the use of **AI techniques** to monitor environments whose analysis is becoming more and more complex because of the large number of sensors used and the need of dealing with different issues and threats in a same environment. Most of current surveillance systems

provide solutions for particular problems but still suffer from lack of flexibility and scalability when they are used on different or related surveillance problems. To overcome this limitation, the adopted solution should ideally address two aspects: a knowledge-based surveillance model flexible enough to deal with different issues, and an architecture that gives support to this model when deploying the surveillance system within a particular scenario. In fact, relevant authors in the field state that incorporating expert-domain knowledge into the automatic monitoring task is very important to advance surveillance systems [VBL+05].

The solution discussed in this work to face the design of such architecture is inspired by Multi-Agent Systems (MAS) [Wei99]. The main **qualities of intelligent agents**, autonomy, reactivity, proactiveness, and social ability, lay the foundations for an architecture where these entities provide the support needed to deal with the information distributed around the monitored environment. Together with learning capabilities, this approach involves a good framework for addressing the design of complex surveillance systems.

The adoption of the **theoretical model** for intelligent surveillance devised in [AVJL+09] has served as a guide when designing the multi-agent architecture. This is because it allowed to identify the functional requirements (services) of some of the agents that compose the architecture and to establish the inference model of some of them. Since the surveillance model is based on normality analysis, the *normality agents* emerge in a natural way as the entities that manages the knowledge needed to carry out normality surveillance. The adoption of this approach instead of an approach based on abnormality definition was discussed in Section 3.3.

To complete the model of **normality**, the architecture makes possible the deployment of **abnormality** agents that are responsible for identifying the more common anomalous situations that can happen in the monitored environment according to a surveillance concept. This simple and scalable solution allows to devise a surveillance model composed of the following steps:

- Definition of normal situations.
- Definition of more common anomalous situations.
- Deployment of normality and abnormality agents.
- Identification of a situation that does not match with a normal situation.
- Identification of the anomalous situation that took place in the monitored environment.

This model, linked with a particular surveillance concept, has been extrapolated to an indefinite number of concepts or threats to be monitored. For this reason, the **communication infrastructure** provided by the architecture is flexible enough to deal with this issue. On the one hand, the

event-based communication mechanism allows to integrate new surveillance devices, analysis agents and monitoring tools into the surveillance system. The direct communication mechanism facilitates the interactions between the agents that study a same concept or threat from the points of view of normality and abnormality. Finally, the blackboard architecture makes possible the sharing of knowledge between agents.

On top of this sub-layer of normality and abnormality agents, a global normality analysis module has been designed in order to give support to the process of **information fusion**. This design solution is easy to address because the knowledge flow follows a bottom-up approach. However, it complicates the *fusion information agents* because models that concurrently take into account the different information providers must be managed by these agents. Besides, it establishes an agent hierarchy that requires mechanisms of replication in order to not compromise the surveillance system if one of the fusion agents does not reason properly. A possible alternative, considered in the next section, may involve the use of negotiation mechanisms between agents in order to share knowledge.

Most of the design solutions adopted in this work has the **scalability** and the **generality** as the last goals, taking into account not only the information consumers and providers, but also covering the management of the knowledge distributed between the agents that compose the architecture. In fact, the design solutions for the multi-agent architecture proposed in this dissertation have been generalised, when possible.

This though became evident in Chapter 4, where the **agent platform** that reuses some of the solutions of the surveillance architecture was described in depth. Using another agent middleware, such as JADE [BCG07], could have been another alternative instead of developing a new framework. However, as discussed in Section 4.1, most existing agent development frameworks are focused on particular technologies that may not be appropriate for the surveillance domain, where certain accuracy and response-time are needed. The evaluation of the agent framework presented in this work offers very good results (see Section 5.3). These new approaches can contribute to the evolution of agent technology from the development point of view.

6.3. Future research lines

There exist different research lines that can be addressed to enrich both the architecture for intelligent surveillance systems and the agent platform devised from the extended solutions of such architecture. First, the proposals related to the surveillance architecture will be studied. Then, future research lines for the agent platform will be described.

Currently, incorporating **negotiation and argumentation** abilities to the *normality agents* as an alternative to the information aggregation provided by the fusion agents is an interesting research line. This would involve a more consistent approach where the distributed control has more weight instead of the bottom-up approach supported by the fusion agents. This research line would imply the design of new communication protocols other than the blackboard architecture to support the interactions between the *normality agents*.

The hypothesis is that the *normality agents* try to convince each other to establish who of them is reasoning in a more precise way when monitoring the environment. For instance, two *normality agents* could be studying the same normality concept in different sub-environments, but the first of them could have more accurate information than the other one. In this point, the first agent could use such information to prove that he is more capable of performing the analysis. Within this context, one of the issues to take into account is the concurrency control between the agents that monitor a common object or concept.

Machine learning algorithms could also be integrated into *abnormality agents* in order to try to predict future anomalies in the monitored environment. This quality would be very useful for the security personnel to anticipate potentially dangerous situations. For instance, an *abnormality agent* may use the knowledge generated by the *normality agent* who shares the same surveillance concept to build behaviour patterns. Afterwards, these patterns could be used for directly notifying the monitoring tools about a possible future anomalous situation.

The architecture could also be extended with a **KDD** (Knowledge Discovery in Databases [FPSS96]) agent for extracting useful knowledge from the surveillance database. This agent would be responsible for carrying out a forensic analysis of the information gathered by the surveillance system in order to extract patterns for different purposes.

Another important research line is to deploy more surveillance systems that analyse different surveillance concepts in **more complex environments**, which requires the monitoring of a high number of sub-environments. This would directly affect the rate of surveillance events and the capacity of the architecture prototype to respond in environments of higher demand.

The surveillance model and the architecture can be used in domains that deal with behaviour analysis others than surveillance. Some examples are communication networks, in order to detect anomalies or avoid malfunction, or smart grids, in order to study the behaviour of the nodes and help the automatic system readjust the configuration of the grid.

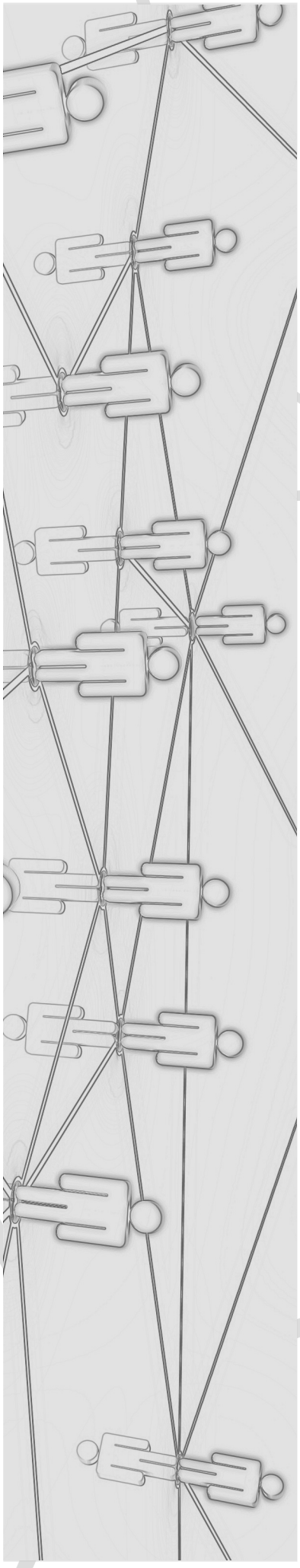
On the other hand, the **development of the agent platform** is being continued. One of the main goals is to interoperate with JADE [BCG07] for the agents to interact one another without worrying about the underlying

technology. To do that, a gateway agent between our platform and JADE is being developed due to the use of different transport protocols. In the context of FIPA standards, more interaction protocols have to be integrated in order to support a wider range of communicative acts.

Moreover, how to include **mobility support** for the agents to move out between hosts is also being evaluated. For example, a mechanism that allows a concrete agent to migrate to a certain host if more resources are needed to attend a request may be very interesting. To do that, a mobility protocol which establishes the rules to be followed in order for the agents to clone or migrate should be developed.

Finally, a **lightweight agent** can also be developed in order to run it on low-resource devices, such as smartphones or PDAs, and to integrate it within the agent platform. From a technical point of view, this could be done by using ZeroC Ice-E, the embedded version of the middleware Ice that gives support to the agent platform. This approach would imply a higher number of hardware devices that can take part in the agent platform.

Anexos



Anexo A
Diseño de un Agente de Normalidad

Anexo B
Detalles de Implementación

A

Diseño de un Agente de Normalidad

En este anexo se detalla y discute una posible implementación de un *normality agent* na_i basado en el diseño planteado en la sección 3.4.1, el cual se recuerda a continuación:

$$na_i =< IDna_i, c, IA_i, \widetilde{KB}_i, IE_i, QM_i, O_i >$$

Las partes más relevantes de cara a la implementación de este tipo de agentes, en las que se profundizará en esta sección, son la base de conocimiento $\widetilde{KB}_i$ y el motor de inferencia IE_i. La primera de ellas especifica el conocimiento experto que se utiliza para llevar a cabo el análisis de normalidad de un concepto c en un sub-entorno particular E_j. Por otra parte, el motor de inferencia define cómo generar nuevo conocimiento como consecuencia de procesar el nuevo estado del mundo o sub-entorno en el caso particular estudiado en este trabajo. En este contexto, la información necesaria para generar nuevo conocimiento la proporciona IA_i, es decir, el conjunto de sensores o *preprocessing agents* responsables de notificar los eventos de vigilancia al agente de normalidad.

El agente de normalidad que se describirá a continuación ha sido desarrollado con el lenguaje de sistemas expertos CLIPS para llevar a cabo el análisis de normalidad del concepto *trayectorias*. La elección de CLIPS para representar el conocimiento de este agente de normalidad se basa en los siguientes motivos:

- **Flexibilidad**, debido a que con CLIPS es posible representar tanto el conocimiento mediante reglas, funciones y objetos.
- **Eficiencia**, gracias a que el motor de inferencia de CLIPS se basa en el algoritmo RETE [For91] y a que está desarrollado con el lenguaje C, proporcionando un entorno de trabajo eficiente. Esta característica es especialmente relevante en los sistemas de vigilancia inteligente.

- **Extensibilidad**, debido a que es posible integrar código de otros lenguajes de programación con código escrito en CLIPS. Así mismo, cabe la posibilidad de utilizar *Fuzzy CLIPS* (versión que incorpora lógica difusa) para tratar de manera más directa con la incertidumbre del entorno.

Antes de detallar los aspectos de implementación de dicho agente, se resumirá en qué consiste CLIPS y cuáles son sus principales características.

A.1. CLIPS: herramienta para la construcción de sistemas expertos

CLIPS (*C Language Integrated Production System*) es una herramienta para el diseño y desarrollo de sistemas expertos. CLIPS fue concebido para facilitar el desarrollo de software que modelara conocimiento humano o experto. Existen tres maneras distintas de representar el conocimiento con CLIPS:

1. **Reglas**, utilizadas para modelar conocimiento heurístico basado en la experiencia.
2. **Funciones**, utilizadas para modelar conocimiento procedural.
3. **Programación orientada a objetos**, cuyo propósito es el de modelar conocimiento procedural mediante las principales características de la orientación a objetos: clases, manejadores de mensajes, abstracción, encapsulación, herencia y polimorfismo. Las reglas se pueden aplicar tanto a objetos como a hechos.

CLIPS proporciona un entorno completo para el desarrollo de sistemas expertos e incluye tanto un editor como un depurador. El núcleo de razonamiento de CLIPS está basado en el algoritmo RETE [For91], un algoritmo eficiente para la comparación de patrones usados para implementar sistemas de reglas. RETE fue diseñado por Charles L. Forgy en la Universidad de Carnegie Mellon como su proyecto de tesis [For79].

Los elementos básicos de CLIPS son la *lista de hechos*, la *base de conocimiento* o conjunto de reglas y el *motor de inferencia*. La ejecución de un sistema desarrollado con CLIPS está guiado por los datos, los cuales definen qué reglas se disparan o activan, las cuales a su vez pueden generar nuevo conocimiento mediante la inserción de nuevos datos o hechos. A continuación se muestra un programa sencillo que muestra la inserción de un hecho en la base de conocimiento.

Inserción de un hecho con CLIPS

```
CLIPS (V6.24 06/15/06)
CLIPS> (assert (duck))
<Fact-0>
CLIPS> (facts)
f-0     (duck)
For a total of 1 fact.
CLIPS>
```

Los comandos básicos de CLIPS incluyen la manipulación de hechos mediante la inserción (*assert*) y eliminación (*retract*), la consulta de los hechos (*facts*) y reglas (*rules*) que pueblan la base de conocimiento de un sistema desarrollado con CLIPS, y la limpieza de la base de hechos (*reset*) o de toda la base de conocimiento (*clear*), incluyendo las propias reglas.

CLIPS ofrece un entorno de desarrollo muy potente, debido a que permite la combinación de los objetos, las reglas o una combinación de ambos a la hora de desarrollar un sistema experto. Así mismo, es bastante eficiente debido a que está implementado en C y su motor de inferencia está basado en el algoritmo RETE, como se comentó anteriormente. Por otra parte, CLIPS se puede integrar con otros lenguajes de programación con el objetivo de delegar determinadas funcionalidades de manera externa al programa principal.

Uno de los elementos más importantes de CLIPS son las **reglas**, las cuales definen qué acciones se han de ejecutar en caso de que se cumplan unas determinadas condiciones. A continuación se expone la estructura general de una regla, la cual mantiene un esquema *if-then-else*.

Estructura general de una regla con CLIPS

```
(defrule nombre-regla ``Comentarios''
    (condición 1)
    (condición 2)
    ...
=>
    (acción 1)
    (acción 2)
    ...
    (acción n))
```

Como se puede apreciar en el listado anterior, los antecedentes de la regla representan las condiciones que se han de cumplir para que se ejecute el bloque de consecuentes. Dichos consecuentes permiten establecer las acciones que se ejecutarán en el caso de que se dispare o active la regla, es decir, siempre que se cumplan todas las condiciones del bloque *if*.

Ejemplo simple de regla

```
(defrule duck
    (animal-is duck)
=>
    (assert(sound-is quack));
```

CLIPS también posibilita el uso de variables que, como en otros lenguajes de programación, permiten almacenar valores que pueden variar a lo largo del tiempo. En otras palabras, es posible mantener contenedores dinámicos de información. A continuación se muestra un ejemplo del uso de variables y en la inserción de un hecho que dispara una regla que actualiza la base de conocimiento.

Uso de variables

```
CLIPS> (reset)
CLIPS> (defrule make-quack
    (duck-sound ?sound)
=>
    (assert (sound-is ?sound)))
CLIPS> (assert (duck-sound quack))
<Fact-1>
CLIPS> (run)
CLIPS> (facts)
f-0      (initial-fact)
f-1      (duck-sound quack)
f-2      (sound-is quack)
For a total of 3 facts.
```

Además de la programación heurística mediante reglas y de la programación funcional mediante funciones, CLIPS ofrece desde la versión 6.0 la posibilidad de definir y manejar clases mediante COOL (*CLIPS Object-Oriented Language*), lenguaje integrado en CLIPS para la programación procedural orientada a objetos. Esta característica dota a CLIPS de una gran potencia ya que facilita el modelado de problemas mediante las ventajas que proporciona la programación orientada a objetos: herencia, encapsulación y polimorfismo.

A continuación se muestra un ejemplo sencillo del uso de clases y de objetos para modelar dos clases, instanciar una de ellas, y definir el manejador de una de las clases.

Ejemplo simple con clases e instancias

```
(defclass DUCKLING (is-a USER)
  (slot sound (default quack))
  (slot age (visibility public)))

(defclass DUCK (is-a DUCKLING)
  (slot sound (default quack)))

(definstances DUCKY_OBJECTS
  (Dorky_Duck of DUCK (age 2))
  (Dinky_Duck of DUCKLING (age .1)))

(defmessage-handler USER print-slots ()
  (ppinstance))

(defmessage-handler DUCK lie-about-age (?change)
  (bind ?new-age (- ?self:age ?change))
  (dynamic-put age ?new-age)
  (printout t "I am only " ?new-age crlf))
```

A.2. Detalles de la base de conocimiento

En esta sección se detallan los aspectos principales de la base de conocimiento de un prototipo de *normality agent* encargado de realizar el análisis de normalidad del concepto *trayectorias*.

En la base de conocimiento de los *normality agents* se diferencian dos tipos de definiciones. Por una parte, existe el conocimiento **general**, el cual está asociado a la definición de ciertos conceptos que son comunes a cualquier entorno de vigilancia. Algunos ejemplos de estos conceptos son *zona*, *instante*, *actor* o *rol*. Todos estos elementos son independientes del concepto o amenaza a analizar por el propio agente. A continuación se muestra parte de la base de conocimiento definida en CLIPS y asociada a este tipo de conceptos.

Clases *POSITION* y *ZONE*

```
(defclass POSITION (is-a USER)
  (slot x (type INTEGER))
  (slot y (type INTEGER))

  (message-handler equals))

(defclass ZONE (is-a USER)
  (multislot points
        (access initialize-only))

  (message-handler get-i-x)
  (message-handler get-i-y)
  (message-handler in))
```

Clases *MOMENT* e *INTERVAL*

```
(defclass MOMENT (is-a USER)
  ; Si year, month o day son 0,
  ; todos los valores son posibles.
  ; Ej. day = 5, month = 0, year = 0;
  ; los días 5 de cada mes y cada año.
  (slot year (type INTEGER) (default 0))
  (slot month (type INTEGER) (default 0) (range 0 12))
  (slot day (type INTEGER) (default 0) (range 0 31))
  (slot hour (type INTEGER) (default -1) (range -1 23))
  (slot minute (type INTEGER) (default 0) (range 0 59))
  (slot second (type INTEGER) (default 0) (range 0 59))

  (message-handler subtract)
  (message-handler before)
  (message-handler after))

(defclass INTERVAL (is-a USER)
  (slot beginning
        (access initialize-only))
  (slot end
        (access initialize-only))

  (message-handler during))
```

Clases *ROLE* y *ACTOR*

```
(defclass ROLE (is-a USER)
  (slot role (type STRING)
        (access initialize-only))
  (slot belief (type FLOAT)))

(defclass ACTOR (is-a USER)
  ; Lista de posibles roles de un actor
  ; con sus grados de creencia.
  (multislot roles)
  ; Posición 2D del actor.
  (slot pos)
  ; El tiempo se refleja con una instancia del tipo MOMENT.
  (slot t)
  (slot r1 (type INTEGER))
  (slot r2 (type INTEGER))
  ; Velocidad vectorial.
  (slot speed)
  ; Últimas zonas recorrida por el actor.
  (multislot last_zones_covered)
  (multislot normality)

  (message-handler put-x-pos)
  (message-handler get-x-pos)
  (message-handler put-y-pos)
  (message-handler get-y-pos)
  (message-handler add-role)
  (message-handler get-normality-component)
  (message-handler belongs-ellipse)
  (message-handler update-last-zones-covered)
  (message-handler update-last-zones-beliefs)
  (message-handler new-covered-zone)
  (message-handler increment-points-zone))
```

El segundo tipo de conocimiento está directamente ligado con el concepto analizado por el *normality agent* y es específico de la especialización del mismo a la hora de monitorizar un sub-entorno E_i. A continuación se muestra el conocimiento más relevante de este ejemplo de agente de análisis de trayectorias.

Concepto *trayectorias* y restricciones associadas

```
(defclass MOVEMENT (is-a USER)
  ; Zona de inicio.
  (slot beginning
        (access initialize-only))
  ; Zona de fin.
  (slot end
        (access initialize-only)))

        (defclass MOVEMENT_SPATIAL_RESTRICTION (is-a USER)
  ; Circulación asociada a la restricción espacial.
  (slot movement
        (access initialize-only))
  (slot order (default FALSE)
        (access initialize-only))
  (multislot allowed_zones
        (access initialize-only))

  (message-handler get-belief-without-order)
  (message-handler get-belief-with-order))

(defclass MOVEMENT_TEMPORAL_RESTRICTION (is-a USER)
  ; Circulación asociada a la restricción temporal.
  (slot movement
        (access initialize-only))
  ; Circulación previa, en caso de que así fuera.
  (slot previous_movement
        (access initialize-only))
  ; Instancia de momento en el tiempo.
  (slot duration (type INTEGER) (default 0)
        (access initialize-only))
  (slot interval
        (access initialize-only))

  (message-handler get-belief))

(defclass MOVEMENT_ACTOR_RESTRICTION (is-a USER)
  (slot movement
        (access initialize-only))
  ; Lista de actores que pueden realizar la circulación.
  (multislot allowed_actors
        (access initialize-only))

  (message-handler get-belief))
```

Manejadores para comprobar las restricciones

```
; Calcula el grado de creencia de la restricción espacial
; a partir de una secuencia de zonas visitadas.
(defmessage-handler MOVEMENT_SPATIAL_RESTRICTION
                    get-belief-without-order (?zones_covered)
  (bind ?belief 0.0)
  (progn$ (?allow_zone (send ?self get-allowed_zones))
    (progn$ (?zone ?zones_covered)
      ; Si coincide la zona visitada...
      (if (= (str-compare ?allow_zone
                          (send ?zone get-zone)) 0)
          then
        ; Actualizamos la creencia.
        (bind ?belief (max ?belief
                           (send ?zone get-belief))))))

  (return ?belief))

; Calcula el grado de creencia de la restricción temporal
; a partir de las zonas reconocidas.
(defmessage-handler MOVEMENT_TEMPORAL_RESTRICTION
                    get-belief (?rec_movs)
  ; Si alguna de las circulaciones reconocidas
  ; coincide con la de la restricción...
  ; Se devuelve 1.
  (if (= (str-compare ?self:previous_movement "[]") 0)
      then
    (return 1))
  (progn$ (?mov ?rec_movs)
    (if (= (str-compare ?self:previous_movement
                        (send ?mov get-movement)) 0)
        then
      (return 1)))

  (return 0))

; Calcula el grado de creencia de la restricción de actor
; a partir de una secuencia de roles (objetos de tipo ROLE).
(defmessage-handler MOVEMENT_ACTOR_RESTRICTION
                    get-belief (?roles)
  (bind ?belief 0)
  (progn$ (?allow_actor (send ?self get-allowed_actors))
    (progn$ (?role ?roles)
      ; Si coincide el actor de la restricción con el del rol.
      (if (= (str-compare ?allow_actor
                          (send ?role get-role)) 0)
          then
        ; Actualizamos la creencia.
        (bind ?belief (max ?belief
                           (send ?role get-belief))))))
  (return ?belief))
```

A.3. Detalles del motor de inferencia

El motor de inferencia del *normality agent* para el análisis de trayectorias está representado por un conjunto de reglas que activan los procesos de monitorización del entorno conforme el agente va recibiendo eventos de

vigilancia. En este contexto, la tarea de comprobación de antecedentes de las mismas y de su activación está delegado en CLIPS. A continuación se muestran las principales reglas que permiten llevar a cabo el proceso de vigilancia de trayectorias, haciendo uso para ello del conocimiento previamente descrito.

Principales reglas del agente de trayectorias

```
1  ; Disparo: cuando un actor entra en la escena.
2  ; Funcionalidad: asignar al actor un objeto del tipo
3  ; normalidad de circulaciones.
4  (defrule new-actor
5    ?actor<-(object (is-a ACTOR))
6    =>
7    (bind ?mov_norm (make-instance
8       (sym-cat (instance-name ?actor) -movement)
9                of MOVEMENT_NORMALITY))
10   (slot-insert$ ?actor normality 1 ?mov_norm))
11
12 ; Disparo: un actor aparece en escena o cambia su posición.
13 ; Funcionalidad: actualizar las últimas zonas cubiertas y
14 ; analizar el componente de normalidad.
15 (defrule new-event-position
16   ; Si cambia la posición del actor...
17   ?actor<-(object (is-a ACTOR)(pos ?pos))
18   (object (is-a POSITION) (name ?pos) (x ?x) (y ?y))
19   =>
20   ; Actualizamos la lista de últimas zonas visitadas.
21   (send ?actor update-last-zones-covered)
22   ; Estudio de la normalidad de las circulaciones.
23   (send (send ?actor
24               get-normality-component MOVEMENT_NORMALITY)
25               analysis ?actor))
26
```

A continuación se detallan los tiempos de ejecución empleados por este agente de normalidad de trayectorias a la hora de monitorizar el entorno estudiado en la sección 5.1[1]. En la tabla A.1 se muestran las características técnicas del equipo utilizado para desplegar dicho agente.

Como se discutió anteriormente, la base de conocimiento de los agentes de normalidad se ha modelado con CLIPS y el motor de inferencia está basado en el algoritmo RETE utilizado por el mismo. Resulta interesante recalcar este hecho debido a que la mayor carga de trabajo en el proceso de monitorización recae sobre los agentes de normalidad. En particular, el agente de trayectorias se ha implementado en C++ para facilitar la integración con CLIPS y proporcionar una implementación eficiente. En la tabla A.2 se expone el tiempo empleado en cada uno de los instantes de tiempo analizados por parte del agente de normalidad en el proceso de vigilancia del entorno de tráfico urbano (ver figura 5.1). Como se puede apreciar, y debido a que la mayor carga de trabajo recae en los agentes de normalidad, existen dos

[1]Los agentes de normalidad de la versión final del prototipo de arquitectura, desarrollados en Java, son una evolución del agente de normalidad discutido en el presente anexo.

Sistema operativo	Debian GNU/Linux
Procesador	Intel(R) Core(TM)2 Duo CPU P9300
Frecuencia CPU	2.26 GHz
Memoria caché	6 MB
RAM	4 GB

Tabla A.1: Especificaciones técnicas de la máquina usada para evaluar la eficiencia del agente de normalidad.

Instante	Objectos		Trayectorias	Tiempo análisis (seg.)
	Peatones	Vehículos		
t_1	1	2	8	0.129 s.
t_2	1	2	7	0.173 s.
t_3	1	1	6	0.162 s.
t_4	1	1	5	0.174 s.
t_5	1	1	4	0.122 s.
t_6	1	2	8	0.204 s.

Tabla A.2: Tiempo empleado por el agente de trayectorias para la vigilancia del escenario de la figura 5.1 en cada instante de tiempo.

factores que influyen de manera directa en el tiempo de análisis por parte del sistema:

1. El número de objetos monitorizados.
2. El número de instancias de trayectorias normales asociadas a cada objeto en cada instante de tiempo.

Como se puede apreciar en la tabla A.2, los tiempos empleados en el análisis de cada instante de tiempo son bajos e incluyen los tiempos de comunicación entre agentes y la actualización de las estructuras de datos asociadas a cada agente. Dichos instantes de tiempo tienen una separación que varía entre uno y dos segundos para comprobar cómo evoluciona el análisis del sistema multi-agente. No obstante, a la hora de explotar este tipo de sistemas es preciso estudiar y parametrizar correctamente la frecuencia con la que el nivel reactivo envía eventos al nivel deliberativo para que sean analizados y posteriormente comunicados al personal de seguridad a través de las herramientas de monitorización. Este valor ha de ser estudiado en función de las características del entorno y de la precisión con la que se quiere efectuar la vigilancia del mismo.

B

Detalles de Implementación

B.1. Arquitectura del sistema de vigilancia

En esta sección se detallan los aspectos de implementación más relevantes de la arquitectura para el despliegue de sistemas de vigilancia diseñada en este trabajo. El código fuente se puede descargar de Internet[1].

B.1.1. Definición de interfaces

```
1  // -*- mode: c++; coding: utf-8 -*-
2
3  #ifndef _INTERFACES
4  #define _INTERFACES
5
6  #include <Ice/BuiltinSequences.ice>
7
8  #include <Types.ice>
9
10 module ISurveillance
11 {
12
13   interface NA
14   {
15     /**
16      * Query the normality agent's state.
17      * @param qi The query information.
18      * @return The query result.
19      **/
20     QueryResult query (QueryInfo qi);
21
```

[1] http://code.google.com/p/oreto-surveillance/

```
22      /**
23       * Notify a surveillance event.
24       * @param qe The event to analyse.
25       **/
26      void notifyEvent (Event ev);
27
28      /**
29       * Terminate the agent and free the resources.
30       **/
31      void destroy ();
32    };
33
34    sequence<NA*> NASeq;
35
36    exception NAExists {};
37
38    interface NAFactory
39    {
40      /**
41       * Create a normality agent.
42       * @param type The surveillance concept.
43       * @param s The monitored scene characteristics.
44       * @param rs The instantiated constraints of the concept.
45       * @return The agent proxy.
46       **/
47      NA* create (string type, Scene s, Restrictions rs) throws NAExists;
48
49      /**
50       * Find out a normality agent.
51       * @param type The surveillance concept.
52       * @return The agent proxy.
53       **/
54      idempotent NA* find (string type);
55
56      /**
57       * Get the sequence of normality agents.
58       * @return The sequence of normality agents.
59       **/
60      idempotent NASeq list ();
61    };
62
63    exception OntologyNotExists {};
64
65    interface LoaderAgent
66    {
67      /**
68       * Get the ontology associated to a surveillance concept.
69       * @param ontology The kind of surveillance concept.
70       * @return The requested ontology.
71       **/
72      idempotent Ice::ByteSeq request (string ontology)
73         throws OntologyNotExists;
```

```
74    };
75
76    interface AA
77    {
78      /**
79       * Notify the violation of the normality.
80       * @param ai The abnormality information.
81       **/
82      void notify (AbnormalityInfo ai);
83    };
84
85    exception AlreadySubscribed {};
86
87    interface Blackboard
88    {
89      /**
90       * Read a slot of the blackboard.
91       * @param id The identifier of the slot.
92       * Return The blackboard information.
93       **/
94      idempotent T read (TID id);
95
96      /**
97       * Read n slots of the blackboard.
98       * @param seqid The identifiers of the slots.
99       * Return The blackboard information.
100      **/
101     idempotent TSeq readSeq (TIDSeq seqid);
102
103     /**
104      * Read all the slots of the blackboard.
105      * Return The blackboard information.
106      **/
107     idempotent TSeq readAll ();
108
109     /**
110      * Write onto the blackboard.
111      * @param data The data to be written.
112      * Return If the writing succeeded.
113      **/
114     bool write (T data);
115
116     /**
117      * Subscribe an object to the blackboard.
118      * @param prx The object to be subscribed.
119      **/
120     void subscribe (Object* prx) throws AlreadySubscribed;
121
122     /**
123      * Unsubscribe an object to the blackboard.
124      * @param prx The object to be unsubscribed.
125      **/
```

```
126     idempotent void unsubscribe (Object* prx);
127   };
128
129   interface ENA
130   {
131     /**
132      * Notify that a blackboard was updated.
133      * @param prx The updated blackboard.
134      **/
135     void notify (Blackboard* prx);
136   };
137
138 };
139
140 #endif
```

B.1.2. Concepto de trayectorias: estructuras de datos específicas

```
1  // -*- mode: c++; coding: utf-8 -*-
2
3  #ifndef _RESTRICTIONS
4  #define _RESTRICTIONS
5
6  #include <Ice/BuiltinSequences.ice>
7  #include <Types.ice>
8
9  module ISurveillance
10 {
11
12   class MovementSpatialRestriction extends SpatialRestriction
13   {
14     Movement currentMovement;
15     bool order;
16     Ice::StringSeq allowedZones;
17   };
18
19   class MovementTemporalRestriction extends TemporalRestriction
20   {
21     Movement currentMovement;
22     Movement previousMovement;
23     int duration;
24     Interval temporalInterval;
25   };
26
27   class MovementActorRestriction extends ActorRestriction
28   {
29     Movement currentMovement;
30     Ice::StringSeq allowedActors;
31   };
32
```

```
33  class MovementRestrictions extends Restrictions {};
34
35  class MovementQueryInfo extends QueryInfo
36  {
37    string actor;
38    bool location;
39    bool restriction;
40    bool direction;
41  };
42
43  struct Location
44  {
45    string zone;
46    float belief;
47  };
48
49  sequence<Location> LocationSeq;
50
51  struct MovementBelief
52  {
53    string movement;
54    float MAR;
55    float MSR;
56    float MTR;
57    float destination;
58    float belief;
59  };
60
61  sequence<MovementBelief> MovementBeliefSeq;
62
63  struct MovementDistance
64  {
65    string movement;
66    string end;
67    float belief;
68  };
69
70  struct RecognizedMovement
71  {
72    string movement;
73    float belief;
74  };
75
76  sequence<RecognizedMovement> RecognizedMovementSeq;
77
78  sequence<MovementDistance> MovementDistanceSeq;
79
80  class MovementQueryResult extends QueryResult
81  {
82    LocationSeq locations;
83    MovementBeliefSeq movementBeliefs;
84    MovementDistanceSeq movementDistances;
```

```
85      RecognizedMovementSeq recognizedMovements;
86    };
87
88    class MovementAbnormalityInfo extends AbnormalityInfo
89    {
90      string id;
91      RoleSeq obj;
92      Moment t;
93      MovementQueryResult res;
94    };
95
96  };
97
98  #endif
```

B.2. Plataforma multi-agente

En esta sección se detallan los aspectos de implementación más relevantes de la plataforma para el despliegue de sistemas de multi-agente diseñada en este trabajo. En primer lugar se muestran las definiciones más relevantes de los tipos de datos e interfaces de la plataforma multi-agente definidas con el lenguaje Slice (IDL de ZeroC ICE). A continuación se discuten los aspectos más relevantes de la versión actual de la implementación de dicha plataforma. Todo el código fuente está liberado[2] bajo licencia GPL[3].

B.2.1. Definición de interfaces

```
1  // -*- mode: c++; coding: utf-8 -*-
2
3  /**
4   *
5   * Implementation of the FIPA Agent Management Specification.
6   * See http://fipa.org/specs/fipa00023/SC00023K.html.
7   **/
8
9  #include <ASDF.ice>
10 #include <PropertyService.ice>
11 #include <FIPATypes.ice>
12
13 module FIPA {
14
15   interface Agent;
16
17   exception AgentExists {};
18   exception AlreadySubscribed {};
19
20   interface AgentFactory {
21     /**
22      * Create an agent.
23      *
24      * @param name The agent's name.
25      *
26      * @return The agent's proxy.
27      **/
28     Agent* create (string name) throws AgentExists;
29
30     /**
31      * Create an agent.
32      *
33      * @param names The names of the agents.
34      *
```

[2] http://code.google.com/p/basic-fipa-multiagentsystem/
[3] http://www.gnu.org/copyleft/gpl.html

```
35        * @return The sequence of created agents proxies.
36        *
37        * @return The names of the agents already registered.
38        **/
39      AgentSeq createSeq (Ice::StringSeq names,
40                  out Ice::StringSeq namesAlreadyRegistered);
41    };
42
43    interface AMS {
44      /**
45       * Register an agent with the agent platform.
46       *
47       * @param aid The proxy to the agent.
48       **/
49      void register (Agent* aid) throws AgentExists;
50
51      /**
52       * Deregister an agent with the agent platform.
53       *
54       * @param id The agent's identity.
55       **/
56      idempotent void deregister (Ice::Identity id);
57
58      /**
59       * Search for an agent in the agent platform.
60       *
61       * @param id The agent's identity.
62       *
63       * @return The proxy to the agent.
64       **/
65      idempotent Agent* search (Ice::Identity id);
66
67      /**
68       * Get the description of the agent platform.
69       *
70       * @return The description of the agent platform.
71       **/
72      idempotent ApDescription getDescription ();
73    };
74
75    // TODO: DF federation.
76    interface DF extends ASD::Listener, ASD::Search {
77      /**
78       * Register an agent's description with the DF.
79       *
80       * @param desc The agent's DF description.
81       **/
82      void register (DfAgentDescription desc)
83        throws AgentExists;
84
85      /**
86       * Deregister an agent with the DF.
```

```
87        *
88        * @param id The agent's identity.
89        **/
90      idempotent void deregister (Ice::Identity id);
91
92      /**
93        * Modify an agent's description with the DF.
94        *
95        * @param desc The agent's DF description.
96        **/
97      void modify (DfAgentDescription desc);
98
99      /**
100       * Search for an agent description in the DF.
101       *
102       * @param desc The agent's DF description.
103       *
104       * @return The agents' proxies that match the description.
105       **/
106     idempotent AgentSeq search (DfAgentDescription desc);
107
108     /**
109       * Subscribe an agent with the DF.
110       *
111       * @param aid The proxy to the agent.
112       **/
113     void subscribe (Agent* aid) throws AlreadySubscribed;
114
115     /**
116       * Unsubscribe an agent with the DF.
117       *
118       * @param aid The Proxy to the agent.
119       **/
120     idempotent void unsubscribe (Agent* aid);
121   };
122
123   interface MTS {
124     /**
125       * Receive an ACL message.
126       *
127       * @param aclMessage The ACL message.
128       **/
129     ["ami"] void receiveMessage (Message m);
130   };
131
132   interface Agent {
133     /**
134       * Set the agent's state.
135       *
136       * @param st The new agent's state.
137       **/
138     void setState (AgentState st);
```

```
139
140     /**
141      * Get the agent's state.
142      *
143      * @return The new agent's state.
144      **/
145     idempotent AgentState getState ();
146
147     /**
148      * Receive an ACL message.
149      *
150      * @param aclMessage The ACL message.
151      **/
152     ["ami"] void receiveMessage (Message m);
153
154     /**
155      * Notification about a registration, deregistration
156      * or modification of an agent with the DF.
157      *
158      * @ param act The act of the notification.
159      *
160      * @param desc The agent's description.
161      **/
162     void dfAdv (DFAction act, DfAgentDescription desc);
163
164     /**
165      * Terminate the agent execution.
166      **/
167     void destroy ();
168   };
169
170 };
```

```
1  // -*- mode: c++; coding: utf-8 -*-
2
3  /**
4   *
5   * Implementation of the FIPA Agent Management Specification.
6   * See http://fipa.org/specs/fipa00023/SC00023K.html.
7   **/
8
9  #include <PropertyService.ice>
10 #include <Ice/Identity.ice>
11 #include <Ice/BuiltinSequences.ice>
12
13 module FIPA {
14
15   /**
16    * Possibles agent's states: Initiated, Active, Suspended,
17                                Waiting, and Transit.
18    **/
19   enum AgentState {Initiated, Active, Suspended,
```

```
20                    Waiting, Transit};
21
22    /**
23     * Possibles performatives of a ACL message.
24     **/
25    enum ACLPerformative {Accept, Agree, Cancel, Cfp, Confirm,
26          Disconfirm, Failure, Inform, InformIf, InformRef,
27          NotUnderstood, Propagate, Propose, Proxy, QueryIf,
28          QueryRef, Refuse, RejectProposal, Request, RequestWhen,
29          RequestWhenever, Subscribe};
30
31    /**
32     * Possibles actions with the DF.
33     **/
34    enum DFAction {Registration, Deregistration, Modification};
35
36    interface Agent;
37
38    sequence<Agent*> AgentSeq;
39
40    struct Envelope {
41      AgentSeq to;
42      Agent* frm;
43      string aclRepresentation;
44      long date;
45    };
46
47    struct ACLPayload {
48      ACLPerformative performative;
49      string content;
50      // See FIPA00007.
51      string language;
52      // See FIPA00007.
53      string encoding;
54      string ontology;
55      // See FIPA00025.
56      string protocol;
57      // To be unique.
58      Ice::Identity conversationId;
59    };
60
61    struct Message {
62      Envelope env;
63      ACLPayload payload;
64    };
65
66    struct ApService {
67      string name;
68      string type;
69      Ice::ObjectSeq addresses;
70    };
71
```

```
72    sequence<ApService> ApServiceSeq;
73
74    struct ApDescription {
75      string name;
76      ApServiceSeq apServices;
77    };
78
79    struct ServiceDescription {
80      string name;
81      string type;
82      Ice::StringSeq protocols;
83      Ice::StringSeq ontologies;
84      Ice::StringSeq languages;
85      PropertyService::Properties properties;
86    };
87
88    sequence<ServiceDescription> ServiceDescriptionSeq;
89
90    struct DfAgentDescription {
91      Agent* name;
92      ServiceDescriptionSeq services;
93      Ice::StringSeq protocols;
94      Ice::StringSeq ontologies;
95    };
96
97  };
```

B.2.2. Aspectos relevantes de la implementación

Como se discutió en la sección 4.1, la propuesta planteada en este trabajo permite el uso directo de agentes implementados en tres lenguajes de programación distintos: C++, Java y Python. Este enfoque requiere, por lo tanto, la implementación de la fábrica de agentes en los tres lenguajes de programación mencionados. No obstante, es posible extender esta implementación al resto de lenguajes de implementación soportados por el *middleware* ZeroC ICE en la parte del servidor (C#, Visual Basic y Ruby).

La fábrica de agentes hace uso de *Freeze*, el servicio de persistencia de ZeroC ICE. Particularmente, su implementación está basada en *Freeze Evictor*[4], el cual se utiliza para activar, gestionar y desactivar agentes. Para ello, el desarrollador ha de especificar de manera explícita qué datos son persistentes y forman parte del estado del agente, haciendo uso de metadatos. Actualmente, el único dato persistente del agente de gestión básico es el estado del mismo de acuerdo a la definición de FIPA. En la versión actual de la plataforma, la fábrica de agentes con soporte para persistencia está implementada en C++ y Java (ZeroC ICE no da soporte para persistencia con Python).

[4] http://zeroc.com/doc/Ice-3.3.1/manual/Freeze.41.5.html

Persistencia en C++

```
// -*- mode: C++; coding: utf-8 -*-

#include <PersistentAgentFactoryI.h>
#include <Freeze/Freeze.h>

using namespace std;
using namespace FIPA;

#define MAX_SERVS 10

class PersistentAgentServer :
   virtual public Ice::Application {

public:
  PersistentAgentServer(const string& dbName) :
    _dbName(dbName) {
  }

  virtual int run(int argc, char* argv[]){

   _ic = communicator();

   // The object factories
   Ice::ObjectFactoryPtr factory =
     new PersistentAgentFactoryI(_ic);
   _ic->addObjectFactory(factory,
                         PersistentAgent::ice_staticId());

   // ...

   // Create the Freeze evictor
   Freeze::ServantInitializerPtr init =
     new PersistentAgentInitializer;
   _evictor = Freeze::createBackgroundSaveEvictor
              (_adapter, _dbName, "dbfile", init);
   _adapter->addServantLocator(_evictor, "");

   // Create MAX_SERVS servants
   // ...

   for (int i=0; i< MAX_SERVS; i++){

    Ice::Identity id = //...;

    if (!_evictor->hasObject(id)){
      serv = new PersistentAgentI();
      prx = _evictor->add(serv, id);
    }
   }

   // ...
 }

private:

  string _dbName;
  Ice::ObjectAdapterPtr _adapter;
  Freeze::EvictorPtr _evictor;
  Ice::CommunicatorPtr _ic;
};
```

El AMS está vinculado a un servidor que implementa el patrón *configurador de servicios* (*service configurator*) [JS97], que consiste en desacoplar la implementación de servicios del momento en el que se configuran. Este patrón incrementa la flexibilidad y la extensibilidad de aplicaciones permitiendo configurar servicios en cualquier instante de tiempo. Esta solución de diseño se ha utilizado ampliamente en entornos de aplicación (configuración de *applets* Java en navegadores web), en sistemas operativos (configuración de *drivers* de dispositivos) y en sistemas distribuidos (configuración de servicios de comunicación en Internet).

FIPAServer.cpp

```
1  #include <FIPAServer.h>
2
3  extern "C"
4  {
5    IceBox::Service*
6    createFIPAServer(Ice::CommunicatorPtr communicator)
7    {
8      return new FIPAServer;
9    }
10 }
11
12 void
13 FIPAServer::start
14 (const std::string& name,
15  const Ice::CommunicatorPtr& communicator,
16  const Ice::StringSeq& args)
17 {
18   Ice::PropertiesPtr properties;
19   string adapter;
20
21   properties = communicator->getProperties();
22   adapter = properties->getProperty("adapter");
23   _adapter = communicator->createObjectAdapter(adapter);
24
25   _adapter->add(new AMSI("MyPlatform"),
26                 communicator->stringToIdentity("AMS"));
27   _adapter->add(new DFI(_adapter),
28                 communicator->stringToIdentity("DF"));
29   _adapter->activate();
30 }
31
32 void
33 FIPAServer::stop()
34 {
35   _adapter->deactivate();
36 }
```

Este enfoque permite desarrollar los servicios como componentes que se pueden cargar de manera dinámica en un servidor de propósito general que es altamente configurable. La solución propuesta se basa en el despliegue de un servidor que albergar a los principales servicios FIPA, es decir, al AMS, DF y MTS. Las principales ventajas de esta aproximación son las siguientes:

- Es posible optimizar las interacciones entre los servicios que forman parte de un mismo servidor.
- La configuración del servidor es independiente de los procesos de compilación y enlazado.
- La administración es sencilla debido a que este enfoque proporciona un marco de desarrollo común.

La materialización de este esquema se realiza mediante el uso de IceBox, el cual representa el servicio ICE que permite la carga dinámica de servidores. En la versión actual, el AMS está implementado en C++ por cuestiones de eficiencia a la hora de tratar con las funcionalidades de administración del sistema. De este modo, el desarrollador de la aplicación multi-agente puede hacer un uso directo de este componente sin necesidad de preocuparse de cómo funciona internamente. Finalmente, entre las operaciones *register*, *deregister* y *search* existe un interbloqueo para garantizar el correcto acceso concurrente al directorio de agentes administrado por el AMS.

Código de la operación *register* del AMS

```
1  void
2  AMSI::_cpp_register
3  (const FIPA::AgentPrx& aid,
4   const Ice::Current& curr)
5  {
6    IceUtil::Mutex::Lock lock(_m);
7
8    if (_agents.find(aid->ice_getIdentity()) == _agents.end())
9      _agents[aid->ice_getIdentity()] = aid;
10   else
11     throw FIPA::AgentExists();
12 }
```

Al igual que el AMS, el DF también está gestionado mediante el servicio IceBox de ZeroC ICE. Un aspecto fundamental del DF es el mecanismo de suscripción, que permite notificar qué agentes se registran, entre otras operaciones, en este servicio de páginas amarillas definido por FIPA. Para llevar a cabo la implementación de dicho mecanismo se ha optado por plantear un modelo de comunicación basado en el contenido de los mensajes, en lugar de estar basado en emisores y receptores.

La solución se basa en hacer uso de los denominados **canales de eventos**, entendiendo un canal de eventos como un mecanismo de abstracción que proporciona la independencia necesaria entre los publicadores y suscriptores de información. Básicamente, los publicadores envían eventos al canal, y dichos eventos serán reenviados a los suscriptores del mismo. En el contexto del DF, un evento consistirá en la información de los servicios que proporciona un agente cuando se registra con el mismo, en la solicitud de eliminación de dicha suscripción o en la modificación de la información que describe a un agente desde un punto de vista funcional.

Por otra parte, la federación de las posibles instancias del DF en una misma plataforma se ha tratado mediante enlaces entre canales de eventos. De este modo, si existe un enlace entre dos canales y el primero de ellos recibe un evento, entonces es posible efectuar su reenvío al segundo canal en función de un determinado criterio. Actualmente, este criterio se basa en asignar costes a los eventos y a los propios enlaces entre canales con el objetivo de filtrar mensajes en función de la diferencia de costes. Así, un DF reenviará un evento siempre y cuando el coste el mensaje sea menor o igual que el del enlace o enlaces por los que es posible realizar el reenvío. Para modelar la difusión de eventos o *broadcast*, un DF reenviará eventos por los enlaces con coste cero y los eventos con coste cero serán reenviados por cualquier enlace, independientemente del coste de este último.

Para la implementación de este esquema se ha hecho uso de *IceStorm*, el mecanismo de publicación y suscripción proporcionado por ICE. Básicamente, la principal responsabilidad de este servicio es ofrecer una capa de abstracción entre los generadores de información (publicadores) y los consumidores (suscriptores). Para ello, el *middleware* plantea los conceptos de *topics* (canales de eventos), *links* (enlaces), *message* (evento) y *cost* (coste de los mensajes y de los enlaces).

Finalmente, la implementación del MTS sigue la misma filosofía que los dos servicios previamente comentados, haciendo uso de *IceBox* y estando implementado en C++ para obtener una mayor eficiencia.

Bibliografía

[ABC+00] B. Abreu, L. Botelho, A. Cavallaro, D. Douxchamps, T. Ebrahimi, P. Figueiredo, B. Macq, B. Mory, L.Ñunes, J. Orri, et al. Video-based multi-agent traffic surveillance system. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 457–462, 2000.

[ACSLL+09] J. Albusac, J.J. Castro-Schez, L.M. López-López, D. Vallejo, and L. Jiménez. A Supervised Learning Approach to Automate the Acquisition of Knowledge in Surveillance Systems. *Signal Processing, Special issue on Visual Information Analysis for Security*, 89(12):2400–2414, December 2009.

[AVJL+09] J. Albusac, D. Vallejo, L. Jiménez-Linares, J.J. Castro-Schez, and L. Rodríguez-Benítez. Intelligent Surveillance based on Normality Analysis to Detect Abnormal Behaviors. *International Journal of Pattern Recognition and Artificial Intelligence, Special issue on Visual Analysis and Understanding for Surveillance Applications*, 23(7):1223–1244, 2009.

[BBD+06] R.H. Bordini, L. Braubach, M. Dastani, A.E.F. Seghrouchni, J.J. Gómez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica*, 30(1):33–44, 2006.

[BBNP03] J. Beatty, S. Brodsky, M.Ñally, and R. Patel. Next-generation data programming: Service data objects. A Joint Whitepaper with IBM and BEA, 2003.

[BCG07] F. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*. Wiley, 2007.

[BCK03] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2003.

[BCPR08] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa. JADE: A software framework for developing multi-agent applications. Lessons learned. *Information and Software Technology*, 50(1-2):10–21, 2008.

[BDBR04] E.G.P. Bovenkamp, J. Dijkstra, J.G. Bosch, and J.H.C. Reiber. Multi-agent segmentation of IVUS images. *Pattern Recognition*, 37(4):647–663, 2004.

[BDM+06] M. Bramberger, A. Doblander, A. Maier, B. Rinner, and H. Schwabach. Distributed embedded smart cameras for surveillance applications. *Computer*, 39(2):68–75, 2006.

[BFH+04] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for GPUs: stream computing on graphics hardware. In *International Conference on Computer Graphics and Interactive Techniques ACM SIGGRAPH*, pages 777–786, 2004.

[BHK+98] J.E. Boyd, E. Hunter, P.H. Kelly, L.C. Tai, C.B. Phillips, and R.C. Jain. MPI-Video infrastructure for dynamic environments. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, pages 249–254, 1998.

[BME+98] T. Boult, R. Micheals, A. Erkan, P. Lewis, C. Powers, C. Qian, and W. Yin. Frame-rate multi-body tracking for surveillance. In *Proc. DARPA Image Understanding Workshop*, pages 305–308, 1998.

[BME+07] G. Booch, R. Maksimchuk, M. Engle, B. Young, J. Conallen, and K. Houston. *Object-oriented analysis and design with applications*. Addison-Wesley Professional, 2007.

[BQX08] P.K. Biswas, H. Qi, and Y. Xu. Mobile-agent-based collaborative sensor fusion. *Information Fusion*, 9(3):399–411, 2008.

[Bro92] R.A. Brooks. Intelligence without representation. *Foundations of Artificial Intelligence, MIT Press, Cambridge, MA*, pages 139–159, 1992.

[Bui08] C. Buiu. Hybrid Educational Strategy for a Laboratory Course on Cognitive Robotics. *IEEE Transactions on Education*, 51(1):100–107, 2008.

[CBdPT08] J.M. Corchado, J. Bajo, Y. de Paz, and D.I. Tapia. Intelligent environment for monitoring Alzheimer patients, agent technology for health care. *Decision Support Systems*, 44(2):382–396, 2008.

[CCP09] B. Chen, H.H. Cheng, and J. Palen. Integrating mobile agent technology with multi-agent systems for distributed traffic detection and management systems. *Transportation Research Part C*, 17(1):1–10, 2009.

[CDK+02] F. Curbera, M. Duftler, R. Khalaf, W.Ñagy, N. Mukhi, and S. Weerawarana. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2):86–93, 2002.

[CDK05] G.F. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts And Design*. Addison Wesley Longman, 4 edition, 2005.

[CGK+05] K. Chmiel, M. Gawinecki, P. Kaczmarek, M. Szymczak, and M. Paprzycki. Efficiency of JADE agent platform. *Scientific Programming*, 13(2):159–172, 2005.

[CGPM08] F. Castanedo, J. García, M.A. Patricio, and J.M. Molina. A multi-agent architecture to support active fusion in a visual sensor network. In *Second ACM/IEEE International Conference on Distributed Smart Cameras, 2008. ICDSC 2008.*, pages 1–8, 2008.

[CL02] L. Chunlin and L. Layuan. An agent-oriented and service-oriented environment for deploying dynamic distributed systems. *Computer Standards & Interfaces, Elsevier*, 24(4):323–336, 2002.

[CLK+00] R.T. Collins, A. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, P. Burt, et al. *A system for video surveillance and monitoring*. Carnegie Mellon University, the Robotics Institute, 2000.

[CPC84] R.L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. *Computer Graphics*, 18(3):137, 1984.

[CSMGMG04] J.J. Castro-Schez, J. Moreno-García, R. Manjavacas, and C. González. Assisting negotiations and supporting decisions in small e-business. *IEEE International Conference on Fuzzy Systems*, 2:855–860, 2004.

[CT03] M. Cannataro and D. Talia. Knowledge grid: An architecture for distributed knowledge discovery. *Communications of the ACM*, 46(1):89–93, 2003.

[CT04] M. Cannataro and D. Talia. Semantics and knowledge grids: building the next-generation grid. *IEEE Intelligent Systems*, 19(1):56–63, 2004.

[DGRMPP05] M. Delgado, J. Gómez-Romero, PJ Magańa, and R. Pérez-Pérez. A flexible architecture for distributed knowledge based systems with nomadic access through handheld devices. *Expert Systems With Applications*, 29(4):965–975, 2005.

[DLC87] EH Durfee, VR Lesser, and DD Corkill. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, 36(11):1275–1291, 1987.

[DMS+07] M. Daniels, K. Muldawer, J. Schlessman, B. Ozer, and W. Wolf. Real-time human motion detection with distributed smart cameras. In *First ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC'07)*, pages 187–194, 2007.

[Dow98] T.B. Downing. *Java RMI: remote method invocation*. IDG Books Worldwide, Inc. Foster City, CA, USA, 1998.

[dT07] Dirección General de Tráfico. *Cifras de siniestralidad vial recogidas en el año 2007*, 2007. http://www.dgt.es/portal/es/seguridad_vial/estadistica.

[Dur88] E.H. Durfee. *Coordination of distributed problem solvers*. Kluwer Academic Publishers, 1988.

[DV08] H.M. Dee and S.A. Velastin. How close are we to solving the problem of automated visual surveillance? *Machine Vision and Applications*, 9(5-6):329–343, 2008.

[DvdHD+08] H. Detmold, A. van den Hengel, A. Dick, K. Falkner, D.S. Munro, and R. Morrison. Middleware for Distributed Video Surveillance. *IEEE Distributed Systems Online*, 9(2):1–11, 2008.

[EM88] R. Engelmore and A. Morgan. *Blackboard systems*. Addison-Wesley, 1988.

[ERLA06] R. Enficiaud, R., B. Lienard, and N. Allezard. Clovis-a generic framework for general purpose visual surveillance applications. In *IEEE Workshop on Visual Surveillance*, pages 177–184, 2006.

[Fer92] I.A. Ferguson. Touring machines: autonomous agents with attitudes. *Computer*, 25(5):51–55, 1992.

[Fer98] J. Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley, 1998.

[Fie00] R.T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[FIN04] A. Farinelli, L. Iocchi, and D.Ñardi. Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(5):2015–2028, 2004.

[FKNT02] I. Foster, C. Kesselman, J.Ñick, and S. Tuecke. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, 2002. www.globus.org/research/papers/ogsa.pdf.

[For79] C.L. Forgy. *On the efficient implementation of production systems.* PhD thesis, Carnegie Mellon University Pittsburgh, PA, USA, 1979.

[For91] C.L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *IEEE Computer Society Reprint Collection*, pages 324–341, 1991.

[Fos06] I. Foster. Globus toolkit version 4: Software for service-oriented systems. *Journal of Computer Science and Technology*, 21(4):513–520, 2006.

[Fou00a] Foundation for Intelligent Physical Agents. *FIPA Agent Message Transport Service Specification*. Foundation for Intelligent Physical Agents FIPA, Geneva, Switzerland, June 2000. http://fipa.org/specs/fipa00067/SC00067F.html.

[Fou00b] Foundation for Intelligent Physical Agents FIPA. *FIPA Request Interaction Protocol Specification*. Foundation for Intelligent Physical Agents FIPA, Geneva, Switzerland, June 2000. http://fipa.org/specs/fipa00026/SC00026H.html.

[Fou02a] Foundation for Intelligent Physical Agents FIPA. *FIPA Abstract Arquitecture Specification*. Foundation for Intelligent Physical Agents FIPA, Geneva, Switzerland, June 2002. http://fipa.org/specs/fipa00001/SC00001L.html.

[Fou02b] Foundation for Intelligent Physical Agents FIPA. *FIPA ACL Message Structure Specification*. Foundation for Intelligent Physical Agents FIPA, Geneva, Switzerland, March 2002. http://fipa.org/specs/fipa00061/SC00061G.html.

[Fou04] Foundation for Intelligent Physical Agents FIPA. *FIPA Agent Management Specification*. Foundation for Intelligent Physical Agents FIPA, Geneva, Switzerland, March 2004. http://fipa.org/specs/fipa00023/SC00023K.html.

[FP02] D.A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[FPSS96] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, 1996.

[GCM05] J. García, J. Carbo, and J.M. Molina. Agent-based coordination of cameras. *International Journal of Computer Science and Applications*, 2(1):33–37, 2005.

[GHF+07] C. García, Y. Hernández, D. Fernández, M.A. Ferrer, C.M. Travieso, J.B. Alonso, and P. Henríquez. HESPERIA: homeland security technologies for the security in public spaces and infrastructures. In *41st Annual IEEE International Carnahan Conference on Security Technology*, pages 221–226, 2007.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995.

[Gil06] Y. Gil. On agents and grids: Creating the fabric for a new generation of distributed intelligent systems. *Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier*, 4(2):116–123, 2006.

[GLS99] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message passing interface*. MIT press, 1999.

[GMWJ+07] C. González-Morcillo, G. Weiss, L. Jiménez, D. Vallejo, and J. Albusac. *A MultiAgent System for Physically Based Rendering Optimization*, volume 4676/2007 of *Lecture Notes in Computer Science*, pages 149–163. Springer Berlin/Heidelberg, 2007.

[GMWJV07] C. González-Morcillo, G. Weiss, L. Jiménez, and D. Vallejo. A multiagent approach to distributed rendering optimization. In *Innovative Applications of Artificial Intelligence Conference IAAI 2007*, pages 149–163, 2007.

[Hen04a] M. Henning. Massively Multiplayer Middleware. *ACM Queue*, 1(10):38–45, 2004.

[Hen04b] M. Henning. A new approach to object-oriented middleware. *IEEE Internet Computing*, 8(1):66–75, 2004.

[Hen06] M. Henning. The rise and fall of CORBA. *ACM Queue*, 4(5):28–34, 2006.

[Hen09] M. Henning. Choosing Middleware: Why Performance and Scalability do (and do not) Matter. Technical report, ZeroC Inc., February 2009. http://zeroc.com/articles/IcePerformanceWhitePaper.pdf.

[HHD00] I. Haritaoglu, D. Harwood, and L.S. Davis. W 4: real-time surveillance of people and theiractivities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):809–830, 2000.

[HJL03] M. He, N.R. Jennings, and H.F. Leung. On agent-mediated electronic commerce. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):985–1003, 2003.

[HN89] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *International Joint Conference on Neural Networks (IJCNN)*, pages 593–605, 1989.

[HS05] M.N. Huhns and M.P. Singh. Service-oriented computing: key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.

[HTWM04] W. Hu, T. Tan, L. Wang, and S. Maybank. A survey on visual surveillance of object motion and behaviors. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 34(3):334–352, 2004.

[HV99] M. Henning and S. Vinoski. *Advanced CORBA programming with C++*. Addison-Wesley, 1999.

[HW05] A. Helsinger and T. Wright. Cougaar: A robust configurable multi agent platform. In *Proceedings of IEEE Aerospace Conference*, pages 1–10, 2005.

[HXT04] W. Hu, D. Xie, and T. Tan. A hierarchical self-organizing approach for learning the patterns of motion trajectories. *IEEE Transactions on Neural Networks*, 15(1):135–144, 2004.

[Jac90] P. Jackson. *Introduction to expert systems*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1990.

[JCL95] N.R. Jennings, J.M. Corera, and I. Laresgoiti. Developing Industrial Multi-Agent Systems. In *Proceedings of the First International Conference on Multi-agent Systems (ICMAS-95)*, pages 423–430, 1995.

[Jen00] N.R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.

[Jen04] H.W. Jensen. A practical guide to global illumination using ray tracing and photon mapping. In *International Conference on Computer Graphics and Interactive Techniques ACM SIGGRAPH 2004 Course Notes*. ACM, 2004.

[Jon05] S. Jones. Toward an acceptable definition of service. *IEEE Software*, 22(3):87–93, 2005.

[JS97] P. Jain and D.C. Schmidt. Dynamically Configuring Communication Services with the Service Configurator Pattern. *C++ Report*, 1997.

[JW98] Nicholas R. Jennings and Michael J. Wooldridge. Applications of intelligent agents. In *Agent Technology: Foundations, Applications, and Markets*, pages 3–28. Springer-Verlag: Heidelberg, Germany, 1998.

[KBKA05] B. Karlsson, O. Bckstrm, W. Kulesza, and L. Axelsson. Intelligent sensor networks-an agent-oriented approach. In *Workshop on Real-World Wireless Sensor Networks (REALWSN'05), Stockholm, Sweden*. Citeseer, 2005.

[Ker04] I.V. Kerlow. *The art of 3D computer animation and effects*. Wiley, 2004.

[KPP+97] S.E. Kemeny, R. Panicacci, B. Pain, L. Matthies, E.R. Fossum, and L.C. Photobit. Multiresolution image sensor. *IEEE Transactions on circuits and systems for video technology*, 7(4):575–583, 1997.

[Lee03] R.S.T. Lee. iJADE surveillant: an intelligent multi-resolution composite neuro-oscillatory agent-based surveillance system. *Pattern Recognition*, 36(6):1425–1444, 2003.

[LL00] K.C. Laudon and J.P. Laudon. *Management information systems*. Prentice Hall Englewood Cliffs, 2000.

[LLACSJ09] L.M. López-López, J. Albusac, J.J. Castro-Schez, and L. Jiménez. Semantics-Provided Environment Views for Normality Analysis-Based Intelligent Surveillance . In *International Conference on Agents and Artificial Intelligence*, pages 1–10, 2009.

[LLCS+09] L.M. López-López, J.J. Castro-Schez, , D. Vallejo, and J. Albusac. A Recommender System based on a Machine Learning Algorithm for B2C Portals. In *The 2009 IEEE / WIC / ACM International Conferences on Web Intelligence (WI'09)*, 2009.

[LT99] J. Liu and YY Tang. Adaptive image segmentation with distributed behavior-based agents. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(6):544–551, 1999.

[LWJ03] A.R. Lomuscio, M. Wooldridge, and N.R. Jennings. A Classification Scheme for Negotiation in Electronic Commerce. *Group Decision and Negotiation*, 12(1):31–56, 2003.

[Mae95] P. Maes. Artificial life meets entertainment: lifelike autonomous agents. 38(11):108–114, 1995.

[Mas05] A. Mas. *Agentes software y sistemas multiagente: conceptos, arquitecturas y aplicaciones*. Pearson Educación-Prentice Hall, 2005.

[Mat09] Mateos, J.A. and Vallejo, D. and Arriaga, I. and Glez-Morcillo, C. A multi-agent architecture for augmented reality applications. In *IADIS Multi Conference on Computer Science and Information Systems 2009*, June 2009.

[Mee04] P. Meer. *Robust techniques for computer vision*, volume Emerging topics in computer vision, pages 107–190. Prentice Hall, 2004.

[Mey92] Bertrand Meyer. Applying "design by contract". *Computer*, 25(10):40–51, 1992.

[Mit07] H.B. Mitchell. *Multi-sensor Data Fusion: An Introduction*. Springer, 2007.

[Moh06] M. Mohammadian. Multi-Agents Systems for Intelligent Control of Traffic Signals. In *Computational Intelligence for Modelling, Control and Automation, 2006 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, pages 270–270, 2006.

[MP93] J.P. Muller and M. Pischel. The Agent Architecture InteRRaP: Concept and Application. Technical report, German Research Center for Artificial Intelligence, 1993.

[MWN02] S. McLean, K. Williams, and J.Ñaftel. *Microsoft. Net Remoting*. Microsoft Press Redmond, WA, USA, 2002.

[NMW04] C.Ñorris, M. McCahill, and D. Wood. Editorial. The growth of CCTV: A global perspective on the international diffusion of video surveillance in publicly accessible space. *Surveillance & Society*, 2(2/3):110–135, 2004.

[NSG+06] I.A. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, and I.H. Shu. CLARAty: Challenges and Steps Toward Reusable Robotic Software. *International Journal on Advanced Robotics Systems*, 3(1):23–30, 2006.

[Obj08a] Object Management Group (OMG). *Agent Metamodel and Profile, Event Metamodel and Profile*, September 2008. http://agent.omg.org/Agent_docs.html.

[Obj08b] Object Management Group (OMG). *CORBA Specifications (Document Access Page)*, April 2008. http://www.omg.org/technology/documents/index.htm.

[Obj08c] Object Management Group (OMG). *CORBA/IIOP Specification (Document Access Page)*, September 2008. http://www.omg.org/technology/corba_iiop.htm.

[Ope08] Open Source Software Project. *Universally Unique Identifier (UUID)*, July 2008. http://www.ossp.org/pkg/lib/uuid/.

[PBL05] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI reasoning engine. *Multiagent Systems, Artificial Societies and Simulated Organizations*, 15:149, 2005.

[PCP+07] M.A. Patricio, J. Carbó, O. Pérez, J. García, and J.M. Molina. Multi-agent framework in visual sensor networks. *EURASIP Journal on Advances in Signal Processing*, 2007(1):1–21, 2007.

[Pfi98] G.F. Pfister. *In search of clusters*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1998.

[PG03] M.P. Papazoglou and D. Georgakopoulos. Service-Oriented Computing. *Communications of the ACM*, 46(10):25–28, 2003.

[PZR+06] T.O. Paulussen, A. Zoller, F. Rothlauf, A. Heinzl, L. Braubach, A. Pokahr, and W. Lamersdorf. 4 Agent-Based Patient Scheduling in Hospitals. *Multiagent Engineering: Theory And Applications in Enterprises*, page 255, 2006.

[RBG+04] V. Rodin, A. Benzinou, A. Guillaud, P. Ballet, F. Harrouet, J. Tisseau, and J. Le Bihan. An immune oriented multi-agent system for biological image processing. *Pattern Recognition*, 37(4):631–645, 2004.

[RCO04] R. Ross, R. Collier, and G.M.P O'Hare. AF-APL-bridging principles & practice in agent oriented languages. In *Second International Workshop on Programming Multi-Agent Systems (ProMAS'04)*, volume 3346, pages 66–88. Springer, 2004.

[RG95] A.S. Rao and M.P. Georgeff. BDI Agents: From Theory to Practice. *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, 1995.

[RN04] S. Russell and P.Ñorvig. *Inteligencia Artificial. Un enfoque moderno*. 2004.

[RS05] W. Renz and J. Sudeikat. *Modeling minority games with bdi agents-a case study*, volume 3550/2005 of *Lecture Notes in Computer Science*, pages 71–81. Springer Berlin/Heidelberg, November 2005.

[RSJ04] P. Remagnino, A.I. Shihab, and G.A. Jones. Distributed intelligence for multi-camera visual surveillance. *Pattern Recognition, Elsevier*, 37(4):675–689, 2004.

[RVFT07] P. Remagnino, S.A. Velastin, G.L. Foresti, and M. Trivedi. Novel concepts and challenges for the next generation of video surveillance systems. *Machine Vision and Applications*, 18(3):135–137, 2007.

[RZ94] J.S. Rosenschein and G. Zlotkin. *Rules of encounter: designing conventions for automated negotiation among computers*. MIT press, 1994.

[Set05] M. Settings. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications, Elsevier*, 28(1):1–18, 2005.

[SH05] M.P. Singh and M.N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley & Sons. New York, NY, USA, 2005.

[Sho76] E.H. Shortliffe. *Computer Based Medical Consultations: MYCIN*. American Elsevier, 1976.

[Smi04] G.J.D. Smith. Behind the screens: Examining constructions of deviance and informal practices among cctv control room operators in the UK. *Surveillance and Society*, 2(2/3):376–95, 2004.

[SR00] E. Stringa and C.S. Regazzoni. Real-time video-shot detection for scene surveillance applications. *IEEE Transactions on Image Processing*, 9(1):69–79, 2000.

[Sun06] Sun Microsystems. *Enterprise Java Beans Specification*, May 2006. http://java.sun.com/products/ejb/.

[TBH+08] Y. Tian, L. Brown, A. Hampapur, M. Lu, A. Senior, and C. Shu. IBM smart surveillance system (S3): event based video surveillance system with an open and extensible framework. *Machine Vision and Applications*, 19(5):315–327, 2008.

[TG05] V.R. Tomás and L.A. García. A cooperative multiagent system for traffic management and control. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 52–59. ACM New York, NY, USA, 2005.

[TLKC00] E. Turban, J. Lee, D. King, and H.M. Chung. *Electronic commerce: a managerial perspective*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 2000.

[UM05] N. Ukita and T. Matsuyama. Real-time cooperative multitarget tracking by communicating active vision agents. *computer vision and image understanding*, 97(2):137–179, 2005.

[UPn08] UPnP Forum. *UPnP Standards*, December 2008.

[US05] M.S. Uddin and T. Shioyama. Bipolarity and projective invariant-based zebra-crossing detection for the visually impaired. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 22–30, 2005.

[VAGMJ08] D. Vallejo, J. Albusac, C. Glez-Morcillo, and L. Jiménez. *A Service-Oriented MultiAgent Architecture for Cognitive Surveillance*, volume 5180/2008 of *Lecture Notes in Computer Science*, pages 101–115. Springer Berlin/Heidelberg, 2008.

[VAJ+09] D. Vallejo, J. Albusac, L. Jiménez, C. González, and J. Moreno. A cognitive surveillance system for detecting incorrect traffic behaviors. *Expert Systems With Applications*, 36(7):10503–10511, 2009.

[VAM+09] D. Vallejo, J. Albusac, J.A. Mateos, C. González, and L. Jiménez. A modern approach to multiagent development. *Journal of Systems and Software*, 2009. IN PRESS.

[VBL+05] S.A. Velastin, B.A. Boghossian, B.P.L. Lo, J. Sun, and M.A. Vicencio-Silva. Prismatica: Toward ambient intelligence in public transport environments. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 35(1):164–182, 2005.

[VCSAJGM08] D. Vallejo, J.J. Castro-Schez, J. Albusac-Jiménez, and C. Glez-Morcillo. Integrating a standard communication protocol into an e-commerce environment based on intelligent agents. In *9th International Conference On Enterprise Information Systems (ICEIS)*, pages 55–60, 2008.

[VLS04] S.A. Velastin, B. Lo, and J. Sun. A flexible communications protocol for a distributed surveillance system. *Journal of Network and Computer Applications*, 27(4):221–253, 2004.

[VR05] S.A. Velastin and P. Remagnino. *Intelligent distributed video surveillance systems*. The Institution of Electrical Engineers (IEEE), 2005.

[VRM+09] D. Vallejo, P. Remagnino, D.N. Monekossso, L. Jiménez, and C. Glez-Morcillo. A multi-agent architecture for multi-robot surveillance. In *1st International Conference on Computational Collective, Intelligence Semantic Web, Social Networks and Multiagent Systems (ICCCI'09)*, 2009.

[VV05] M. Valera and S.A. Velastin. Intelligent distributed surveillance systems: a review. In *IEE Proceedings Vision, Image and Signal Processing*, volume 152, pages 192–204, 2005.

[VVM+07] D. Villa, F.J. Villanueva, F. Moya, F. Rincón, J. Barba, and J.C. López. *Minimalist Object Oriented Service Discovery Protocol for Wireless Sensor Networks*, volume 4459/2007 of *Lecture Notes in Computer Science*, pages 472–483. Springer Berlin/Heidelberg, June 2007.

[WC01] M. Wooldridge and P. Ciancarini. *Agent-oriented software engineering: The state of the art*, volume 1957/2001 of *Lecture Notes in Computer Science*, pages 55–82. Springer Berlin/Heidelberg, January 2001.

[WD88] E. Wallace and C. Diffley. CCTV control room ergonomics. Technical report, UK Home Office, 1988. Police Scientific Development Branch (PSDB).

[Wei99] G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[WHRB+88] R. Wesson, F. Hayes-Roth, J.W. Burge, C. Stasz, and C.A. Sunshine. Network structures for distributed situation assessment. *Distributed Artificial Intelligence*, pages 71–89, 1988.

[Win06] M. Winikoff. JACK TM intelligent agents: An industrial strength platform. *Multiagent Systems, Artificial Societies, and Simulated Organizations*, 15:175–193, 2006.

[WJ95] M. Wooldridge and N.R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[Woo97] M. Wooldridge. Agent-based software engineering. *IEE Proceedings Software Engineering*, 144(1):26–37, 1997.

[Woo01] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, Inc. New York, NY, USA, 2001.

[Wor01] World Wide Web Consortium. *Web Services Description Language (WSDL) 1.1*, 2001. http://www.w3.org/TR/wsdl.

[Wor07] World Wide Web Consortium. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*, April 2007.

[Wor09] World Wide Web Consortium. *Web Services Specifications*, 2009. http://www.w3.org/2002/ws/#documents.

[Yag88] R. Yager. On ordered weighted averaging aggregation operators inmulticriteria decisionmaking. *IEEE transactions on Systems, Man and Cybernetics*, 18(1):183–190, 1988.

[Zad65] L.A. Zadeh. Fuzzy sets. *Information Control*, 8(3):338–353, 1965.

[Zad96] L.A. Zadeh. Computing with Words. *IEEE Transactions on Fuzzy Systems*, 4(2):103–111, 1996.

[ZS06] R. Zlot and A. Stentz. *Market-Based Multirobot Coordination Using Task Abstraction*, volume 24/2006 of *Springer Tracts in Advanced Robotics*, pages 167–177. Springer Berlin/Heidelberg, 2006.

Este documento fue editado con *Emacs*
y tipografiado con LaTeX en *Debian GNU/Linux*

Ciudad Real, 30 de Noviembre de 2009