ASISTENTE PARA GESTIÓN DE ORGANIZACIONES DE AGENTES SOFTWARE

Álvaro Rodríguez Pérez
<alvinr6hgt@gmail.com>

Director:
Juan Pavón
<jpavon@fdi.ucm.es>

Proyecto Fin de Máster en Sistemas Inteligentes
Máster en Investigación en Informática, Facultad de Informática,
Universidad Complutense de Madrid
Curso 2008 – 2009

El/la abajo firmante, matriculado/a en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Asistente para gestión de organizaciones de agentes software", realizado durante el curso académico 2008- 2009 bajo la dirección del Dr. Juan Pavón Mestras en el Departamento de Ingeniería del Software e Inteligencia Artificial (ISIA) , y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Álvaro Rodríguez Pérez

Agradecimientos

En primer lugar, me gustaría agradecer a Juan Pavón el tiempo y los conocimientos que ha compartido conmigo y que han hecho posible la realización de este trabajo.

De manera muy especial, quiero también agradecer la ayuda y dedicación que Paco Garijo me ha brindado, no sólo para la consecución de este trabajo sino a lo largo de casi toda mi carrera profesional, que aunque todavía no es muy larga, sí ha sido intensa en muchos momentos y en todos ellos he podido contar con Paco. Así, la relación profesional ha trascendido al plano de lo personal y por ello llevo mi agradecimiento y admiración también a este terreno.

Por último, no puedo olvidarme de agradecer a mi familia. Míriam y Mateo, me he apoyado mucho en vosotros para poder finalizar este trabajo, a pesar de las dificultades.

A todos… Gracias.

Resumen

Los desarrollos de los sistemas multiagente se han centrado en la autonomía de los agentes como entidades individuales y en los mecanismos de comunicación entre ellos. El concepto de organización aplicado a los sistemas multiagente les aporta una nueva dimensión y múltiples ventajas. Por este motivo cada vez son más las plataformas que lo incorporan. El trabajo aquí realizado se enmarca dentro del framework de desarrollo de organizaciones de agentes, ICARO-T. Se ha desarrollado un asistente como un componente funcional del framework para facilitar la construcción de descripciones y estructuras básicas de las organizaciones de agentes.

Palabras clave. Organización, sistemas multiagente, agente, plataforma de agentes, modelo de organización de agentes.

Abstract

Most of Mulitagent system research effort has focused on individual agent's behaviour and interaction. The organization concept raises multiagent systems to another level. This is the reason why many multiagent platforms are now adopting it. This paper describes the development process of a computational component of the organization-oriented agents platform ICARO-T. The main objective of the component is to make the definition of organizations descriptions and the developing of organization basic structures easier.

Keywords Organization, multiagent system, agent, agent framework, agent organization model.

ÍNDICE

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

Los sistemas multiagente son una tecnología de software distribuido donde los principales elementos de proceso son entidades autónomas, denominadas agentes, que interaccionan entre sí para alcanzar un objetivo. La cuestión es que los desarrollos de los sistemas multiagente se han centrado, sobre todo, en la autonomía de los agentes y en los mecanismos de comunicación entre ellos [Zambonelli et al., 2003]. Se ha desarrollado en menor medida el concepto de organización, que permite añadir una nueva dimensión a los sistemas multiagente, aportándoles nuevas capacidades, como la asignación de roles, formación de grupos o el establecimiento de normas.

Los modelos de sistemas multiagente más populares son los sistemas cerrados en los que los agentes son diseñados por un mismo equipo de desarrollo, para ejecutarse en un entorno homogéneo, con objetivos comunes y en los que no se permite la entrada de nuevos agentes. Sin embargo, actualmente están cobrando relevancia los sistemas multiagente *abiertos*, que permiten que los agentes pueden entrar y salir libremente, y cuyo comportamiento no está previamente determinado y por lo tanto esperado. El objetivo es llegar a desarrollar sistemas abiertos en los que se permita la participación de agentes heterogéneos con diferentes arquitecturas y objetivos.

En los sistemas abiertos, como se ha apuntado antes, el comportamiento de los agentes no es ni predecible, ni controlable. Debido a esto, surge la idea de agrupar los agentes en organizaciones, estableciendo medios de control basados en reglas sociales, normas, asignación de roles, etc. De esta manera se puede modelar la coordinación de los agentes en estos sistemas. Las organizaciones permiten definir objetivos a alto nivel, objetivos que los agentes no pueden o no quieren realizar de forma individual. De esta manera se limitan las interacciones de los agentes, la visibilidad entre los mismos, etc.

Existen múltiples plataformas de desarrollo de sistemas de agentes, como JADE [JADE Home], Jack [JACK] o MadKit [MADKIT], pero lo que ocurre en la mayoría de los casos es que el concepto de organización no está presente, al menos de forma explícita. Es el diseñador el que debe plantear y desarrollar las características de la organización, como en el caso de JADE. O bien, si el concepto de organización está

contemplado, como en Jack (Jack Teams) o MadKit, aparecen otros problemas más relacionados con la ingeniería de Software, como es la falta de pruebas en entornos industriales o la carencia de mecanismos de integración con componentes software convencionales.

El framework de desarrollo de sistemas multiagente sobre el que se basa este trabajo, ICARO-T [ICARO], sí contempla el concepto de organización, permitiendo aplicar las ideas asociadas a la misma. El concepto de organización que maneja ICARO-T es relativamente sencillo comparado al que utilizan otros sistemas que se basan en organizaciones que pretenden ser iguales a las humanas, pero ha superado la barrera de quedarse en una mera especificación, cosa no que ocurre en el caso de las otras.

1.1. Motivación

El trabajo que se describe en este documento está enmarcado dentro del framework de desarrollo de aplicaciones distribuidas ICARO-T, que plantea éstas como organizaciones de agentes. ICARO-T proporciona un conjunto de patrones arquitectónicos y componentes genéricos para agentes, implementados en Java, para el desarrollo de aplicaciones distribuidas. Partiendo de los patrones se generan ejemplares, en los que se especifican ciertos parámetros, como el comportamiento. Es una manera de uso de patrones similar a los patrones clase-objeto cuando se programa en Java o en cualquier lenguaje basado en el paradigma objeto.

Como se ha comentado anteriormente las organizaciones de agentes aportan una nueva dimensión a los sistemas multiagente. Gracias a poder aplicar las organizaciones a los sistemas multiagente se facilita la gestión de sistemas de gran escala (sistemas reales), se optimiza y regula la coordinación de servicios así como la distribución de tareas. Al poder coordinar de una manera más eficiente las tareas de los agentes que componen el sistema se facilita la consecución del objetivo del sistema.

Entre las plataformas actuales de desarrollo con agentes se pueden encontrar tanto las que no contemplan el concepto de organización como las que sí lo hacen, pero en ambos casos suelen tener problemas. En las que el concepto de organización no se contempla de manera explícita es el propio diseñador el que tiene que implementar los

mecanismos necesarios manualmente, si quiere obtener algunos de los beneficios que ofrece el uso de organizaciones. Por otro lado las plataformas en las que sí se contempla el concepto de organización suelen carecer de la estabilidad necesaria para utilizarlas en entornos reales, es decir, sistemas de gran escala. Además en este último caso el ingeniero debe adaptarse al modelado no estándar que proponen.

ICARO-T aúna el uso de organizaciones y la capacidad de integración con componentes de Ingeniería de Software reales para la construcción de sistemas de gran escala. Aunque permite construir y coordinar organizaciones, por el momento la definición de estas requiere la definición manual de un fichero XML que se valida durante la ejecución. Es necesario poder gestionar las infraestructuras a un mayor nivel, de tal manera que se facilite al desarrollador la tarea de describir una organización. Gracias a la gestión de las organizaciones se puede coordinar el despliegue de los componentes, sus interacciones y su comportamiento. Se deduce entonces que la definición de la organización es una fase clave en el desarrollo de una aplicación basada en ICARO-T. En la actualidad la descripción de la organización debe realizarse de manera manual, procedimiento que, en el momento que la organización alcanza un tamaño considerable, resulta un poco complicado. Teniendo en cuenta que la descripción de la organización es fundamental para el correcto funcionamiento y despliegue del sistema, es lógico pensar que facilitar lo máximo posible este proceso supone un gran avance.

Para lograr este propósito se ha considerado útil realizar primero un estudio del estado del arte de organizaciones en sistemas multiagente. De esta manera se podrán considerar mejoras a las facilidades que proporciona ICARO-T. Se trata por tanto de plantear también que la plataforma pueda ser extensible para incluir nuevas capacidades para dominios de aplicación concretos.

1.2. Objetivos

El objetivo principal de este trabajo consiste en simplificar la definición de organizaciones para ICARO-T, la generación de nuevos componentes y la validación de las definiciones. Para llevar esto a cabo se ha desarrollado un asistente que permite

definir organizaciones para ICARO-T de una manera más rápida, intuitiva y robusta.

Para realizar el asistente, primero se han analizado los tipos de organizaciones que permite definir ICARO-T y la forma de describirlas. El asistente no deberá sólo permitir al diseñador crear descripciones de organizaciones ya existentes, también podrá generar la estructura básica para poder crear nuevos componentes de la infraestructura. Además el asistente deberá poder interpretar también descripciones de organizaciones ya existentes y presentarlas para su posterior modificación.

El objetivo principal se desglosa en los siguientes apartados:

- *Estudio de los diferentes tipos de organizaciones.* Identificación de las características de los diferentes tipos de organizaciones, desde las humanas hasta las computacionales. Marcando las semejanzas y las diferencias.
- *Análisis del tratamiento del concepto de organización por las plataformas de agentes actuales.* Se comprueba si el concepto de organización está presente, y en qué medida, en diferentes plataformas.
- *Diseño del asistente aplicado al tratamiento de organizaciones que maneja la plataforma ICARO-T.* El asistente se diseña como un agente ICARO-T más de la infraestructura.
- *Valoración de los resultados obtenidos por el asistente*. Comprobar con ejemplos el correcto funcionamiento del asistente.

Un objetivo que surge del desarrollo del asistente es facilitar la creación de nuevos asistentes para mejorar la gestión de las organizaciones, tanto en tiempo de diseño como en tiempo de ejecución. Para ello se podrían desarrollar otros agentes que proporcionen nuevos servicios de organización, construidos a partir de los existentes.

1.3. Estructura del documento

En este apartado se describe la estructura general de este documento. Aparte de este primer capítulo de introducción, la organización del resto de capítulos es la siguiente:

Capítulo 2. Se desarrolla de forma resumida el estado del arte relacionado con el concepto de organización, las plataformas de agentes y la relación entre ambos. En el apartado de organizaciones se comentan las características de las organizaciones humanas y las de agentes y se realiza una comparación entre ambos tipos. En la parte de las plataformas de agentes se enumeran algunos ejemplos de plataformas y se comenta su aproximación al concepto de organización.

Capítulo 3. Se describe la plataforma de agentes sobre la que se basa este trabajo. Como parte de la descripción se comentan los componentes básicos de la plataforma y la manera de crear una aplicación en la misma.

Capítulo 4. Se analiza el desarrollo del asistente presentado en el trabajo para abstraer la construcción de descripciones de organizaciones en la plataforma ICARO-T. Se describe la arquitectura del asistente así como los componentes que lo forman y su funcionamiento.

Capítulo 5. Se muestran las conclusiones del trabajo realizado y los trabajos futuros que surgen del desarrollo de éste.

2. ESTADO DEL ARTE

Este capítulo se divide fundamentalmente en dos partes. En la primera parte se analiza el concepto de organización, desde el punto de vista humano y desde el punto de vista computacional. Como resultado de este análisis se realiza una comparación de los puntos más importantes que tienen en común. La segunda parte de este capítulo analiza el grado de ayuda para trabajar con organizaciones de agentes que proporcionan las plataformas de agentes software más utilizadas actualmente.

2.1. Organizaciones

En el capítulo anterior se ha mostrado los beneficios de la inclusión del concepto de organización en los sistemas multiagente para gestionar las entidades que los componen y sus interacciones. Para poder entender bien estos beneficios es necesario analizar las características de las organizaciones. En esta sección se comparan los conceptos de organización humana y organización de agentes (computacional) y se detallan las características principales de cada uno.

2.1.1. Organizaciones humanas

A continuación se presentan unas cuantas definiciones de organización según diversas fuentes:

- Según el diccionario de la Real Academia Española de la Lengua [RAE], una organización es una *asociación de personas regulada por un conjunto de normas en función de determinados fines*.
- Según la teoría sociológica [Peiro, 1991] una organización es *una formación o entidad social con un número de miembros que puede ser precisado y una diferenciación interna de las funciones que son desempeñadas por dichos miembros*.

- Según la teoría de organizaciones [Guzmán, 1983], *la organización es la coordinación de las actividades de todos los individuos que integran una empresa con el propósito de obtener el máximo aprovechamiento posible de elementos materiales, técnicos y humanos, en la realización de los fines que la propia empresa persigue*.

Podría deducirse entonces que una organización está formada por un conjunto de individuos que realizan unas tareas concretas y claramente diferenciadas. Además, estos individuos están estructurados siguiendo unas normas y reglas determinadas para poder alcanzar los objetivos de la organización y su conducta y sus acciones deben estar adecuados al contexto de la misma, cumpliendo sus estándares.

En [Fuentes et al. 2004] consideran que la mayoría de las metodologías de sistemas multiagente no representan todas las características sociales que el paradigma orientado a agente requiere. La mejor fuente de adquisición de estas propiedades se puede obtener de las ciencias sociales como la filosofía, la sociología o la psicología. Exportar el conocimiento de estas disciplinas a la ingeniería orientada al agente consiste en encontrar las características que comparten los seres humanos y los agentes.

Las características generales de una organización humana podrían ser [Hodge et al. 2003]:

- Está compuesta por personas.
- Persigue unos fines determinados, que guían las actividades de sus miembros, a través de mecanismos de coordinación y control de la acción.
- Existe una subdivisión del trabajo entre los individuos, mediante la especialización y la división de las tareas.
- Precisa de una estructura formal, con unos roles definidos (independientes de la persona que realice dicho rol); unas responsabilidades asociadas a esos roles; y ciertas relaciones preestablecidas entre los miembros de la organización.

- Todas las actividades establecidas deben relacionarse con los objetivos globales de la organización, pues la existencia de un rol determinado sólo es justificable si sirve para alcanzar realmente esos objetivos.
- Posee unos límites definidos, que establecen quiénes son los miembros de la organización (bien por la especificación directa de cada miembro; o bien por el lugar donde se realiza la actividad).

2.1.2. Organizaciones de agentes

El término organización de agentes, se suele emplear en el campo de los sistemas multiagente para definir un conjunto de agentes que persiguen unos mismos objetivos globales, mediante la coordinación de sus actividades y unos roles y unos patrones de actividad establecidos. Aunque al igual que en el caso de las organizaciones humanas, existen diferentes acepciones del concepto de organización de agentes. Seguidamente se muestran algunas de ellas.

Para L. Gasser las organizaciones son sistemas estructurados con patrones de actividad, conocimiento, cultura, historia y habilidades distintas de cualquier agente particular. Las organizaciones existen a un nivel completamente independiente de los agentes individuales que los constituyan, los cuales pueden ser reemplazables. Además, ocupan alguna región del espacio, bien sea geográfico, temporal, simbólico, etc. [Gasser, 2001]. Por tanto, una organización de agentes proporciona un marco de trabajo para la actividad e interacción de los agentes a través de la definición de roles, expectativas de comportamiento y relaciones de autoridad, como el control.

Por su parte, J. Ferber indica que la organización proporciona una forma para dividir el sistema, separándolo en grupos o unidades que constituyen el contexto de interacción de los agentes [Ferber et al., 2003]. Además, la organización se basa en dos aspectos: estructural y dinámico.

La estructura de la organización representa la parte persistente de la misma cuando los componentes entran y salen de ella, es decir, las relaciones que permiten ver un agregado de elementos como uno único. La estructura define cómo se agrupan los agente, cómo se relacionan los unos con los otros, restricciones, así como los roles

necesarios para realizar las tareas de la organización.

La parte dinámica de la organización define los patrones de interacción que se definen para los roles de la organización, estos patrones definen la manera de entrar y salir de la organización, las obligaciones y permisos de cada uno de los roles y el modo de asignación de los roles.

Para V. Dignum, las organizaciones de agentes asumen la existencia de objetivos globales, aparte de los objetivos individuales de los agentes; y además existen de manera independiente a los agentes, sin asumir ninguna característica interna específica de los mismos [Dignum y Dignum, 2006]. Los roles representan posiciones organizativas responsables de alcanzar parte de esos objetivos globales, de acuerdo a ciertas reglas de interacción predeterminadas. Los agentes pueden tener sus propios objetivos y decidir si adoptan o no unos roles específicos del sistema, así como determinar cuáles de los protocolos disponibles resultan más apropiados para alcanzar los objetivos marcados por esas posiciones o roles que han adoptado.

En [Pavón et al. 2005] y [Pavón et al. 2003] la organización describe el entorno en el que agentes, recursos, tareas y objetivos coexisten y se define por la estructura, la funcionalidad y las relaciones sociales. Desde un punto de vista estructural, la organización es un conjunto de entidades asociadas por relaciones de herencia y de agregación. Sobre esta estructura se definen una serie de relaciones que generan la formación de workflows (dinámico) y relaciones sociales. La organización estructura el sistema multiagente en grupos y workflows. Los grupos pueden estar formados por agentes, roles, recursos y aplicaciones. La asignación de las entidades a los grupos obedece a un propósito organizativo, bien porque se facilita la definición de workflows, bien porque los miembros tienen características similares, etc.

J. Hubner considera a la organización como un conjunto de restricciones de comportamiento que un grupo de agentes adopta para controla la autonomía de los agentes y conseguir alcanzar sus objetivos globales fácilmente [Hubner et al., 2006].

Dentro de los tipos de organizaciones, existe cierta discusión sobre el uso de organizaciones estáticas y dinámicas. Las estáticas se definen en tiempo de diseño,

donde los roles están predefinidos y asignados a los agentes. En las dinámicas los agentes pueden cambiar de rol e incluso la estructura de la organización puede variar. También existen soluciones como las propuestas en [Garijo et al. 2001] y en [Gómez-Sanz et al. 2006] que, desde el punto de vista de ingeniería del software se plantea un punto intermedio. Esta visión permite diseñar la estructura de la organización identificando tipos de agentes y recursos, pero luego en tiempo de ejecución se pueden crear y destruir en función de las necesidades del sistema.

Las características generales de una organización de agentes son:

- Está compuesta por agentes (software, físicos y/o humanos), con independencia de sus características internas y objetivos individuales.
- Persigue un objetivo común global, que no depende directamente de los objetivos individuales de los agentes particulares que participen en ella en cada momento.
- Existe una subdivisión de las tareas de los agentes, mediante su asignación a roles, los cuales describen las actividades y funcionalidad propia de la organización.
- Proporciona una partición del sistema en grupos o unidades, en los cuales tienen lugar la interacción de los agentes.
- Posee unos límites bien definidos, determinados por: el entorno de la organización; los agentes internos y externos; así como por la funcionalidad de la organización y sus servicios ofrecidos.

Si se comparan las características generales de las organizaciones humanas y las de agentes se observan bastantes similitudes, esto es, en cierta manera, lógico, dado que las organizaciones de agentes se han desarrollado a partir de la simulación y adaptación de los comportamientos humanos organizativos.

2.1.3. Conclusiones

Las organizaciones humanas establecen una serie de mecanismos para restringir y canalizar las actividades a realizar, en vistas a coordinarlas. Algunos de estos

mecanismos son la especificación de objetivos y sub-objetivos para determinar las tareas a llevar a cabo, las cuales requieren de ciertos roles con sus correspondientes actividades asociadas. Otro mecanismo sería el sistema de elección e incorporación de miembros. Por último estarían los sistemas de entrenamiento y socialización para crear la conciencia de grupo.

El concepto de organización aplicado a los sistemas multiagente permite la coordinación y colaboración de los componentes del sistema. Permite limitar el alcance de las interacciones de los agentes.

Tras el análisis realizado sobre los tipos de organizaciones se ha comprobado que, en general, las organizaciones de agentes se basan en las organizaciones humanas. Evidentemente, transformar las ideas que comparten ambos conceptos en su correspondiente representación computacional, no resulta nada sencillo. Tanto es así que no existen muchos ejemplos de organizaciones llevadas a la práctica. La plataforma en la que se basa este trabajo, ICARO-T, es un ejemplo de este tipo de sistemas, gracias al enfoque pragmático del concepto de organización que plantea, se ha podido llevar a la práctica y confirmar las ventajas que supone. Mediante la organización de ICARO-T, se define la estructura del sistema, los roles y comportamientos de los agentes, los recursos y la información de despliegue.

2.2. Plataformas de agentes

Existen múltiples plataformas y arquitecturas de agentes. En los desarrollos iniciales de estas plataformas los diseños se centraron en la estructura interna de los agentes y en su comportamiento. Los agentes eran vistos como entidades autónomas y dinámicas que evolucionaban en función de sus propios objetivos, sin que existieran restricciones explícitas externas sobre sus comportamientos ni comunicaciones [O. Boissier, 2007]. El concepto de organización sólo surgía si acaso de las interacciones de los agentes.

Muchas de las plataformas y arquitecturas de agente están basadas en la propuesta de arquitectura abstracta FIPA. El estándar FIPA propone una arquitectura de plataforma de agente que ofrece dos servicios básicos: el servicio de páginas blancas,

para obtener la dirección de los agentes del sistema; y el servicio de páginas amarillas, donde se describen los servicios o funcionalidades de los agentes. Algunas de las plataformas más conocidas, como JADE [Bellifemine et al., 1999], ofrecen estas funcionalidades. Sin embargo, los diseñadores deben implementar prácticamente todas las características organizativas por sí mismos.

Las plataformas de agente con soporte para los conceptos relacionados con las organizaciones deberían ofrecer principalmente: (i) la representación de la organización (su estructura, su reconocimiento por parte de los agentes); (ii) mecanismos de control, especialmente normas; (iii) descripción de la organización en un lenguaje estándar; (iv) mecanismos de monitorización; y (v) soporte a los conceptos de modelado de la organización.

A continuación se resumen algunas de las plataformas de agentes que resumen los conceptos expuestos anteriormente. En concreto se va a analizar la plataforma de agentes JADE, que aunque no tiene soporte para organizaciones es quizá la más popular y sobre la que se basan otras plataformas que sí tienen soporte para organizaciones. La plataforma EIDE que soporta el concepto de instituciones electrónicas y la plataforma JACK Teams, que sí tiene soporte para organizaciones.

2.2.1. JADE

JADE (Java Agent Development Framework) es una plataforma software, implementado en Java, para desarrollar aplicaciones basadas en agentes siguiendo las especificaciones FIPA (Figura 1) para los sistemas multiagente inteligentes. Como consecuencia de seguir esta especificación implementa el Agent Manager System (AMS), el DF (Directory Facilitator) y el ACC (Agent Communication Channel), estos tres agentes se inician con la plataforma.

- AMS: El *Agent Management System* es el agente que se encarga de supervisar el acceso y el uso a la plataforma. Es el responsable de la autenticación de los agentes residentes y del registro.
- DF: El *Directory Facilitator* es el agente que ofrece el servicio de páginas amarillas a la plataforma.

- ACC: El *Agent Communication Channel* es el agente que facilita las interacciones más básicas entre los agentes dentro y fuera de la plataforma.

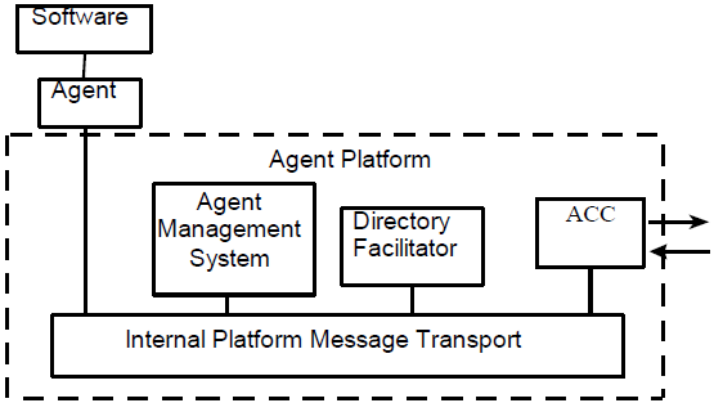


Figura 1 Modelo FIPA de una Plataforma de Agentes

JADE, podría ser considerado como un middleware de agentes que implementa una plataforma de agentes y un framework de desarrollo. Trata todos los aspectos que son independientes de la aplicación y comunes a los agentes, como transporte de mensajes, codificación, parsing o el ciclo de vida de los agentes.

JADE está basado en la arquitectura de comunicaciones punto a punto. Tanto la inteligencia, como la información, los recursos y el control pueden ser distribuidos tanto en terminales móviles como en equipos fijos en la red. El entorno evoluciona dinámicamente con los agentes, que pueden entrar y salir del sistema según los requisitos del entorno de la aplicación. La comunicación entre los agentes es simétrica, pudiendo ser tanto iniciadores como receptores. El lenguaje de los mensajes es el ACL de FIPA.

Los principios que dirigen el desarrollo de la plataforma son:

- Interoperabilidad. Al seguir las especificaciones FIPA, los agentes de JADE pueden comunicarse con agentes de otras plataformas que cumplan con el estándar.
- Uniformidad y portabilidad. Las APIs de JADE son independientes de la red y de la versión de JAVA. Se proveen las mismas APIs para los

entornos J2EE, J2SE y J2ME, así que es el desarrollador el que elige el entorno que más le convenga.

- Facilidad de uso. Toda la complejidad del middleware es transparente al desarrollador que sólo tiene que usar las APIs.
- Filosofía "pay-as-you-go". Los desarrolladores no tienen porqué usar todas las facilidades del middleware. Las que no se utilicen no suponen una sobrecarga para el sistema.

La plataforma de agentes se puede distribuir en varios servidores, en los que se ejecutará como una aplicación dentro de una JVM (Java Virtual Machine). Cada JVM actuará como contenedor y entorno de ejecución de agentes, permitiendo la ejecución de varios agentes al mismo tiempo. Los agentes se comunican a través de un modelo de comunicaciones distribuido y con bajo acoplamiento basado en el envío asíncrono de mensajes. Los mensajes, que siguen el formato ACL definido por FIPA, incluyen campos que indican el contexto en el que se envía el mensaje, tiempos esperados de respuesta, etc. De esta manera se permite el establecimiento de interacciones múltiples entres los agentes. Para poder implementar modelos de conversación más complejos, JADE ofrece estructuras de algunos de los patrones de interacción más comunes para realizar algunas tareas concretas, como negociaciones, subastas o delegación de tareas. Gracias a estos patrones se libera a los desarrolladores de tratar ciertos aspectos de las comunicaciones que no están directamente relacionados con la lógica de la aplicación.

En definitiva, el foco de Jade son las comunicaciones entre agentes siguiendo el estándar de FIPA [FIPA] y da algunos servicios para gestionar el deployment de los agentes en varios nodos, utilizando el concepto de agencia, que también se usa en otras plataformas. Pero no se ofrecen facilidades para definir relaciones entre agentes y recursos, agrupaciones lógicas, etc.

2.2.2. EIDE

El Electronic Institutions Development Environment [EIDE] es un conjunto de herramientas con el objetivo de proporcionar soporte a la creación de aplicaciones inteligentes distribuidas como instituciones electrónicas [Esteva et al, 2008]. Las

instituciones electrónicas son una manera de definir las interacciones entre agentes - humanos o de software - que se pueden dar en un entorno abierto.

Las instituciones electrónicas proporcionan una analogía a les organizaciones humanas donde los agentes inteligentes pueden participar con diferentes roles para conseguir objetivos individuales y de la organización. En este escenario, los agentes permiten reducir los costes y el tiempo al proporcionar mercados distribuidos, representados por los agentes, y donde se consiguen una mejor coordinación. Las instituciones electrónicas aparecen como el pegamento que une los intereses individuales, la coordinación, y la regulación.

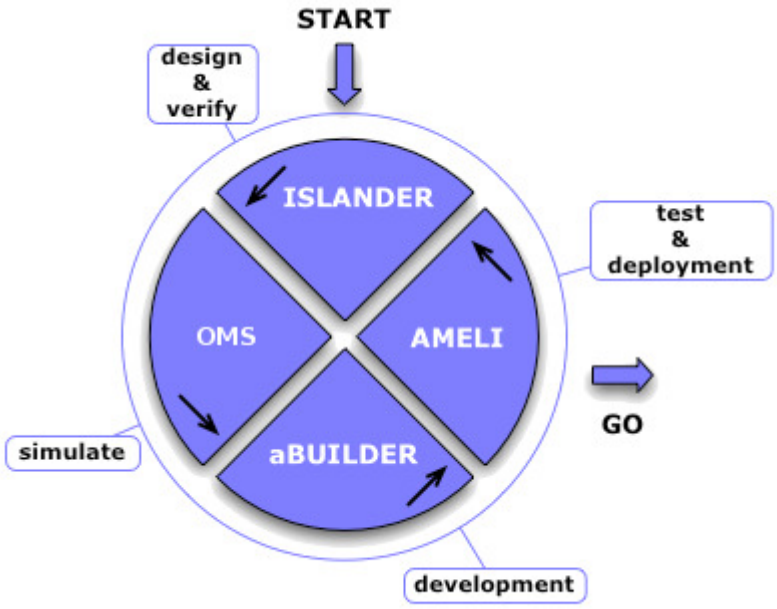


Figura 2 Diagrama Plataforma EIDE

Los agentes de software son la clave tecnológica en la visión de las instituciones electrónicas. Por ello los agentes de software son encapsulados en diferentes comportamientos para su coordinación e interacción dentro de las instituciones electrónicas. El EIDE permite el desarrollo tanto de las instituciones electrónicas como de sus agentes participantes. EIDE sigue un modelo de programación organizativo, más próximo a como se entienden las aplicaciones distribuidas. Así se permite un flujo de

desarrollo de arriba abajo, es decir, primero la organización y luego los individuos.

Las herramientas del EIDE son:

- Islander: Una herramienta gráfica que permite crear la especificación de una institución electrónica, es decir las reglas y los protocolos de interacción entre agentes.
- aBuilder: Herramienta para el desarrollo de agentes.
- AMELI: Plataforma software donde se ejecutan las instituciones electrónicas. Las especificaciones generadas con Islander se pueden ejecutar directamente en AMELI.
- OMS: Módulo para la simulación de instituciones electrónicas.

El paradigma en el que se basa el EIDE resulta atractivo desde el punto de vista humano, pero carece de madurez desde el punto de vista de ingeniería, pues no ha sido probado realmente. Es el tipo de sistema que se ha centrado mucho en la parte teórica del comportamiento del agente y pretende proveer al mismo de capacidades que son utópicas de conseguir con una computadora. El modelo de instituciones electrónicas implementado con EIDE se centra en la sincronización de las interacciones de los agentes y la aplicación de normas en dichas interacciones, pero no aborda los problemas de estructura de organizaciones, importantes desde un punto de vista de ingeniería del software, especialmente cuando se trata de diseñar sistemas escalables y robustos. De hecho, las aplicaciones con el paradigma de instituciones electrónicas son, por el momento, bastante modestas [Rodríguez-Aguilar et al. 1998].

2.2.3. JACK Teams

La plataforma JACK Teams es una extensión del entorno JACK Intelligent Agents [Howden et al., 2001] que proporciona un entorno de programación de agentes orientado a equipos. Al igual que JACK, está basada en el modelo de agente BDI (Belief-Desire-Intention) [Rao et al. 1991] y se ha construido sobre Java.

En la programación orientada a equipos, la colaboración entre agentes se especifica desde el punto de vista del grupo como un todo, para obtener así

comportamientos coordinados. Los conceptos que usa la plataforma son:

- *Equipo*: entidad caracterizada por los roles que tiene y/o los roles que necesita que otros asuman. Se representa como una estructura recursiva que a su vez puede contener cualquier combinación de equipos o subequipos. Es más, la formación de un equipo se consigue mediante la unión de subequipos capaces de realizar los roles requeridos por el equipo.
- *Rol*: un rol en un equipo es una entidad que contiene una descripción de las funcionalidades que los participantes en un equipo/subequipo deben proporcionar. Además define una relación entre equipos y subequipos.
- *Teamdata*: permite la propagación de creencias de un equipo a un subequipo y viceversa. Un elemento teamdata define como una creencia propagada es aceptada por el equipo receptor y como se incorpora a sus estructuras de creencias.
- *Teamplan*: especifica como una tarea se consigue en términos de uno o más roles. Normalmente contiene pasos que determinan cual de sus equipos asumirá cada rol.

Por tanto, la plataforma permite la definición de lo que un equipo es capaz de realizar, el conocimiento que comparte y los roles que participan en él. Además, los equipos tienen las mismas propiedades que un agente JACK, es decir, son entidades BDI, que ejecutan planes (*teamplan*) y razonan mediante la propagación de creencias dentro del equipo (*teamdata*).

JACK Teams proporciona soporte para la coordinación de actividades entre agentes mediante la formación de equipos complejos que contienen subequipos. Sin embargo la plataforma JACK, aunque permite la creación de equipos de agentes, no ofrece soporte para topologías de organizaciones más complejas ni tampoco tiene en cuenta aspectos fundamentales de las organizaciones como son las normas y las reglas.

2.2.4. Conclusiones

En las plataformas de sistemas multiagente se observa una evolución que va desde la visión centrada en los aspectos individuales de los agentes hasta la visión del sistema como una organización, en la que los agentes están estructurados, formando grupos y jerarquías, con normas de comportamiento.

El uso de organizaciones aporta muchas ventajas a la hora de construir sistemas multiagente especialmente desde el punto de vista de ingeniería del software. Por ejemplo, facilita la coordinación y el despliegue de los componentes, reduciendo drásticamente el acoplamiento de los mismos en el sistema. Se simplifica la ingeniería de los sistemas de gran envergadura así como la escalabilidad de los mismos.

Dentro de la evolución hacia la concepción de los sistemas como organizaciones, algunas plataformas multiagente ya poseen los componentes necesarios para soportar estos conceptos. El problema actual con las plataformas tiene dos vertientes, o bien, directamente, no incluyen el concepto de organización o, en el caso de que lo incluyan, carecen de la estabilidad arquitectónica para ser aplicados a sistemas en entornos industriales.

3. ICARO-T UN FRAMEWORK DE DESARROLLO DE APLICACIONES CON ORGANIZACIONES DE AGENTES

ICARO-T es una plataforma de desarrollo de sistemas multiagente, fruto de la experiencia acumulada, durante diez años, en el desarrollo de agentes. El principal factor diferenciador de ICARO-T respecto del resto de plataformas es el uso de patrones de componentes para modelar los sistemas multiagente. Estos patrones están descritos en UML incluyendo aspectos dinámicos y estáticos. El código Java de la plataforma es perfectamente consistente con estos patrones. De esta manera se provee al desarrollador, no sólo de conceptos y modelos, sino de componentes software, compatibles con los estándares de ingeniería software. Mientras otras plataformas de agentes centran sus desarrollos en estándares de comunicaciones, como FIPA, ICARO-T se centra en ofrecer componentes software del alto nivel para facilitar el desarrollo de complejos comportamientos de agentes, coordinación de los mismos y su organización.

En este capítulo se van a describir, brevemente, los aspectos fundamentales de la plataforma, los componentes de la misma y las características básicas de una aplicación [Garijo et al. 2008].

3.1. Componentes

ICARO-T ofrece tres tipos de componentes, el modelo de organización de componentes, para describir la estructura general del sistema, los modelos de agentes, basados en los comportamientos reactivos y cognitivos y el modelo de recursos para encapsular entidades computacionales que ofrecen servicios a los agentes.

En la Figura 3 se muestra de manera general los paquetes que forman la infraestructura y sus dependencias. La plataforma ofrece el marco de trabajo, que incluye los patrones para la construcción de sistemas y es en la parte de aplicaciones en la que se distinguen los componentes que forman una aplicación.

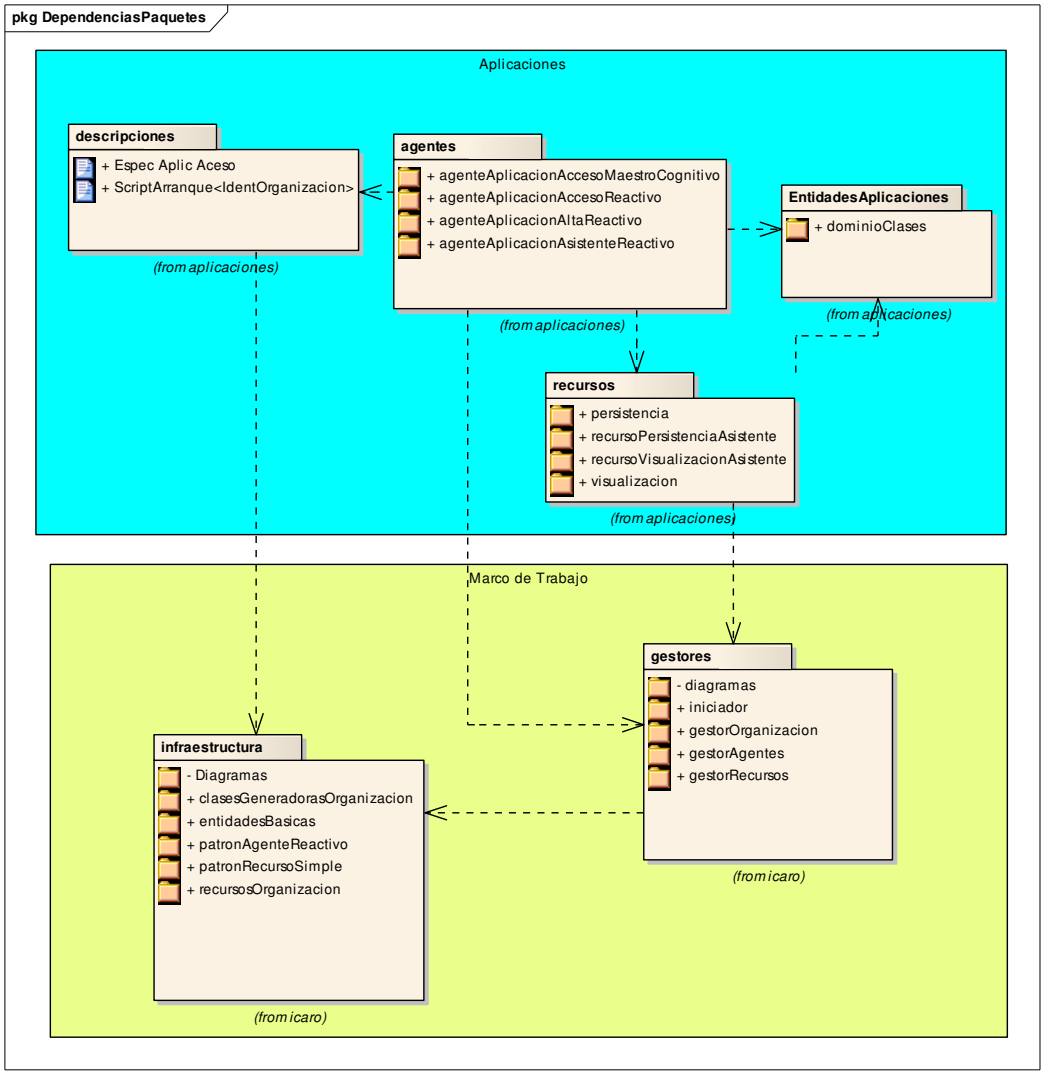


Figura 3 Dependencias de paquetes ICARO-T

3.1.1. Modelo de Agentes

El patrón de agentes de ICARO-T provee a los agentes con interfaces externos y estructuras internas comunes, lo que facilita mucho su desarrollo, evolución y la inclusión de los mismos dentro de aplicaciones ya desarrolladas. De los dos modelos de comportamiento, reactivo y cognitivo, se va a hacer hincapié en el reactivo, pues es el que se ha utilizado en este trabajo. El modelo cognitivo se basa en la definición de un modelo de reglas y el uso de una base de conocimiento para satisfacer los objetivos del sistema.

La arquitectura del agente reactivo se basa en tres componentes que ya vienen

incluidos en el patrón de agente de ICARO-T, percepción, control y ejecución (Figura 4). La percepción guarda los eventos que recibe el agente y se los entrega al control bajo demanda. El control del agente se representa con autómatas de estados finitos. El autómata consume los eventos de la percepción y genera transiciones que cambien el estado del autómata y ejecuta acciones del modelo de ejecución. Para definir un comportamiento de un agente basta con definir la tabla de transiciones del autómata y el modelo de ejecución (acciones semánticas). Se puede definir más de un comportamiento para un mismo agente.

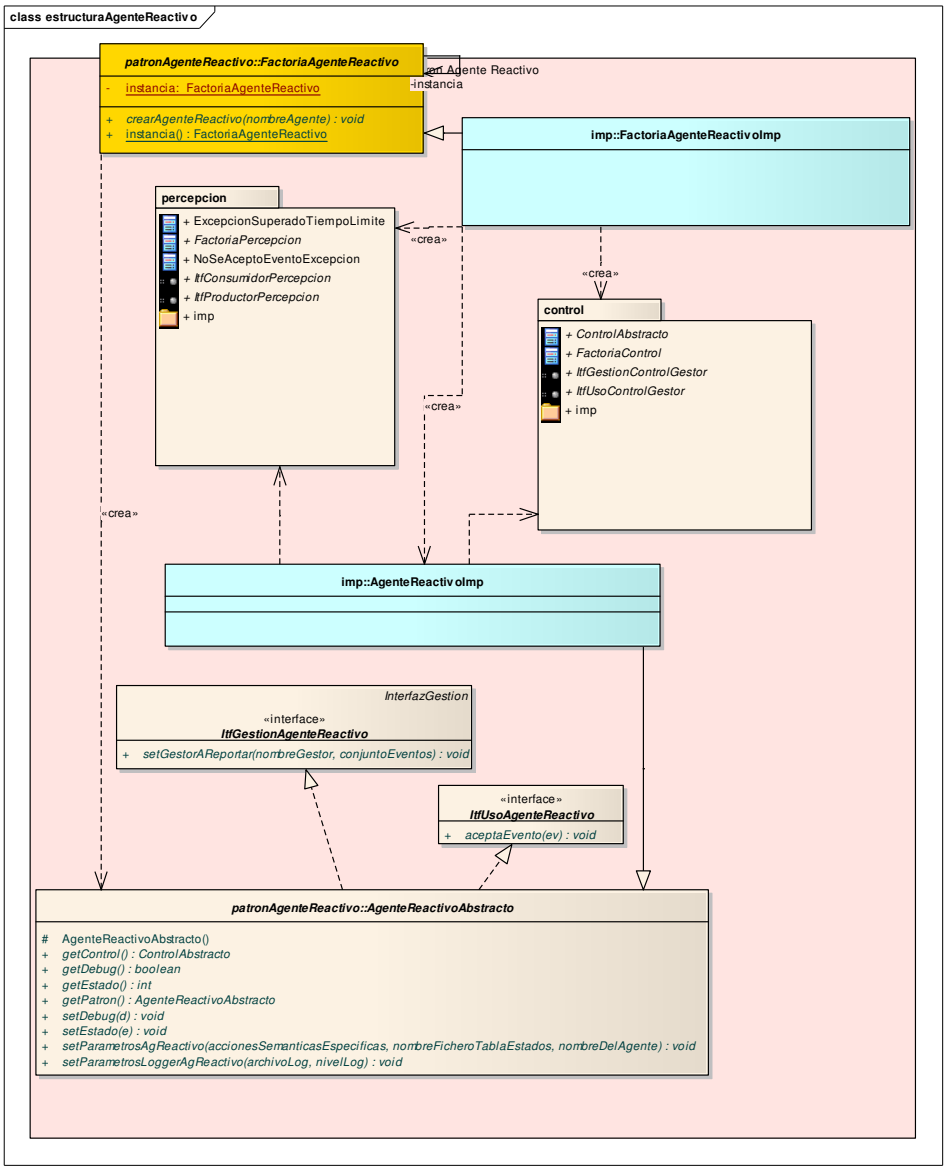


Figura 4 Componentes agente reactivo

En la Figura 5 se puede ver el ciclo de funcionamiento del agente reactivo. El agente reactivo puede recibir eventos de múltiples componentes, agentes y recursos, a través de su interfaz de uso. La percepción del agente ofrece un interfaz para encolar eventos y consumirlos, y provee eventos al control cuando este lo requiere a través del interfaz de consumo de la percepción. El control ejecuta el siguiente ciclo de funcionamiento. Una vez que el agente ha sido creado inicia el proceso de obtención y procesamiento de los eventos de la percepción. Se pide el primer evento, en caso de que no hubiera se queda esperando a que haya alguno disponible. A continuación de haber obtenido el evento se inicia el siguiente ciclo de procesado de eventos:

- Extracción de la información del evento. Se obtiene la acción semántica que habrá que ejecutar y la lista de objetos que habrá que pasar como parámetros.
- Se manda la información al autómata para que transite al estado correspondiente.
- Se espera a que se termine la transición para esperar a un nuevo evento.

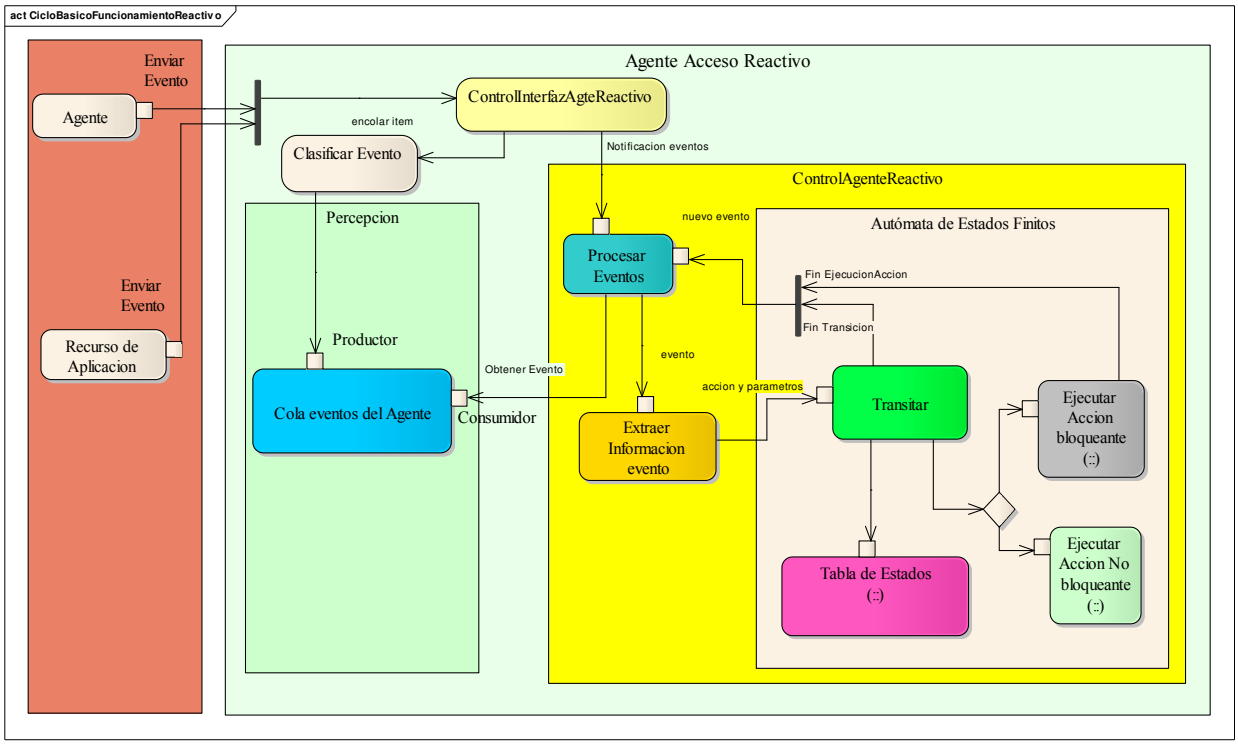


Figura 5 Ciclo de ejecución de un agente reactivo

Cuando el control está en estado de espera de eventos, recibe una notificación de la percepción indicándole que se ha producido un nuevo evento, para que comience el procesado del mismo.

3.1.2. Modelo de Recursos

Los agentes, para alcanzar sus objetivos, interactúan con otras entidades computacionales del entorno. Para los agentes, estas entidades son *recursos.* En ICARO-T, los recursos son las entidades computacionales que no son agentes y se utilizan para obtener información y satisfacer los objetivos. Los recursos de ICARO-T, al igual que los agentes, ofrecen interfaces similares para mostrar las operaciones que pueden realizar.

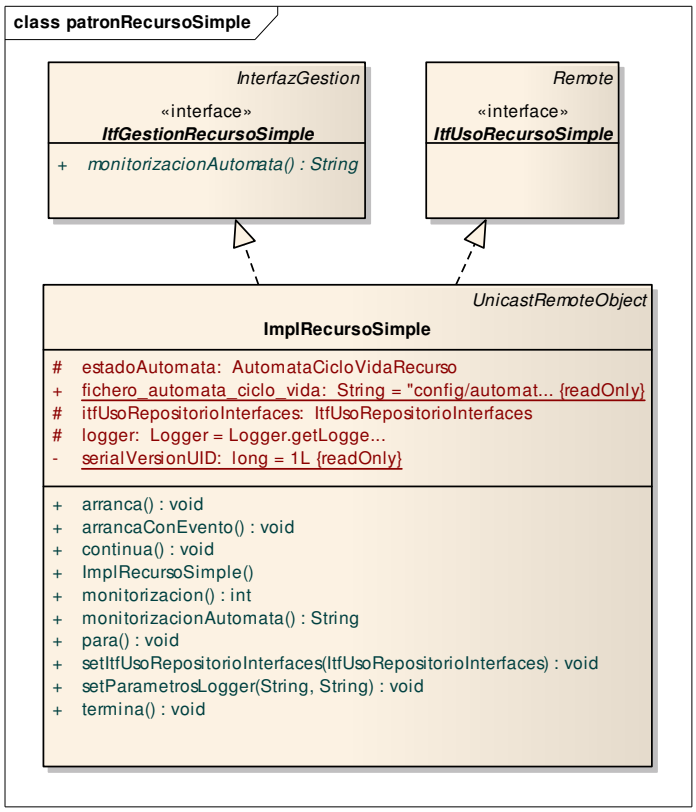


Figura 6 Patrón de recurso

La plataforma provee un patrón de recurso (Figura 6) pensado para definir recursos de aplicaciones como componentes, encapsulando su estructura interna y ofreciendo su funcionalidad a través de dos interfaces, uso y gestión. Ejemplos de este

tipo de recursos son, por ejemplo, los recursos de persistencia que ofrecen funciones de acceso y control de bases de datos relacionales, o los recursos de visualización, que permiten al sistema presentar pantallas de la aplicación o capturar acciones del usuario a través del interfaz gráfico. Otro tipo de recursos podrían ser, reconocedores de voz, analizadores sintácticos, etc.

3.2. Aplicaciones

Una aplicación de ICARO-T se modela como una organización de componentes de control, los agentes, y recursos. En la organización se pueden identificar tres capas (Figura 7), la capa de control, que está formada por los componentes de control, la capa de recursos, formada por los componentes que sirven información o proveen de funcionalidad a los agentes para alcanzar sus objetivos y la capa de información, que contiene las entidades de información necesarias para modelar los datos de las aplicaciones (ontologías, clases, etc.).

La capa de control está formada por dos tipos de componentes, gestores y agentes especializados. Su estructura interna es idéntica, sin embargo sus roles en la infraestructura son diferentes. Los gestores son responsables de las tareas de gestión de una aplicación, como la configuración, la activación, la monitorización y el manejo de excepciones. Los agentes especializados son los encargados de llevar a cabo la funcionalidad propia de la aplicación.

Los agentes gestores y los especializados colaboran para alcanzar los objetivos de la aplicación durante todo el ciclo de vida de la misma. Cabe destacar la figura de los agentes gestores, ignorada en, prácticamente, la totalidad de las plataformas de agentes. Gracias a estos patrones, la plataforma ofrece facilidades para tratar los típicos problemas relacionados con la inicialización, instalación, monitorización y reconfiguración de los agentes y los recursos del sistema. Utilizando los patrones, los desarrolladores se ven guiados a cumplir ciertas estructuras, a la hora de construir los componentes, para que el sistema pueda gestionarlos.

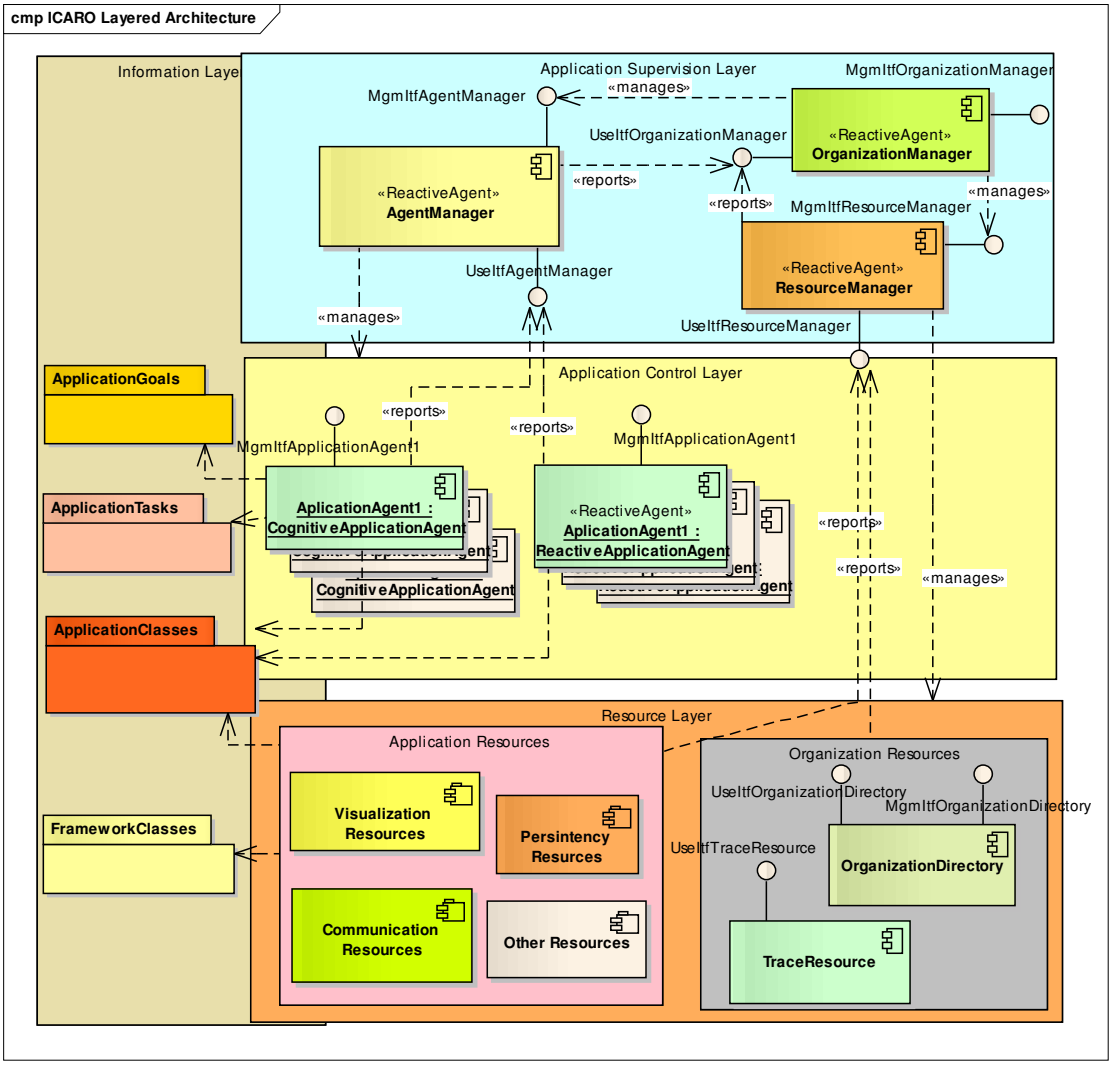


Figura 7 Capas de una aplicación ICARO-T

La capa de control está dividida en dos partes. El área de control de la aplicación contiene los agentes que implementan la funcionalidad de la aplicación y el área de supervisión de la aplicación contiene tres agentes gestores, el gestor de organización, el gestor de agentes y el gestor de recursos, que están implementados como agentes reactivos. Estos tres componentes siguen una estructura jerárquica, el gestor de organización se encarga de la creación y supervisión de la estructura general de la aplicación, el gestor de agentes se ocupa de la creación y supervisión de los agentes de la aplicación y el gestor de recursos crea y monitoriza los recursos que los agentes necesiten. Tanto el gestor de agentes como el gestor de recursos reportan sus estados al gestor de organización que es el que toma decisiones.

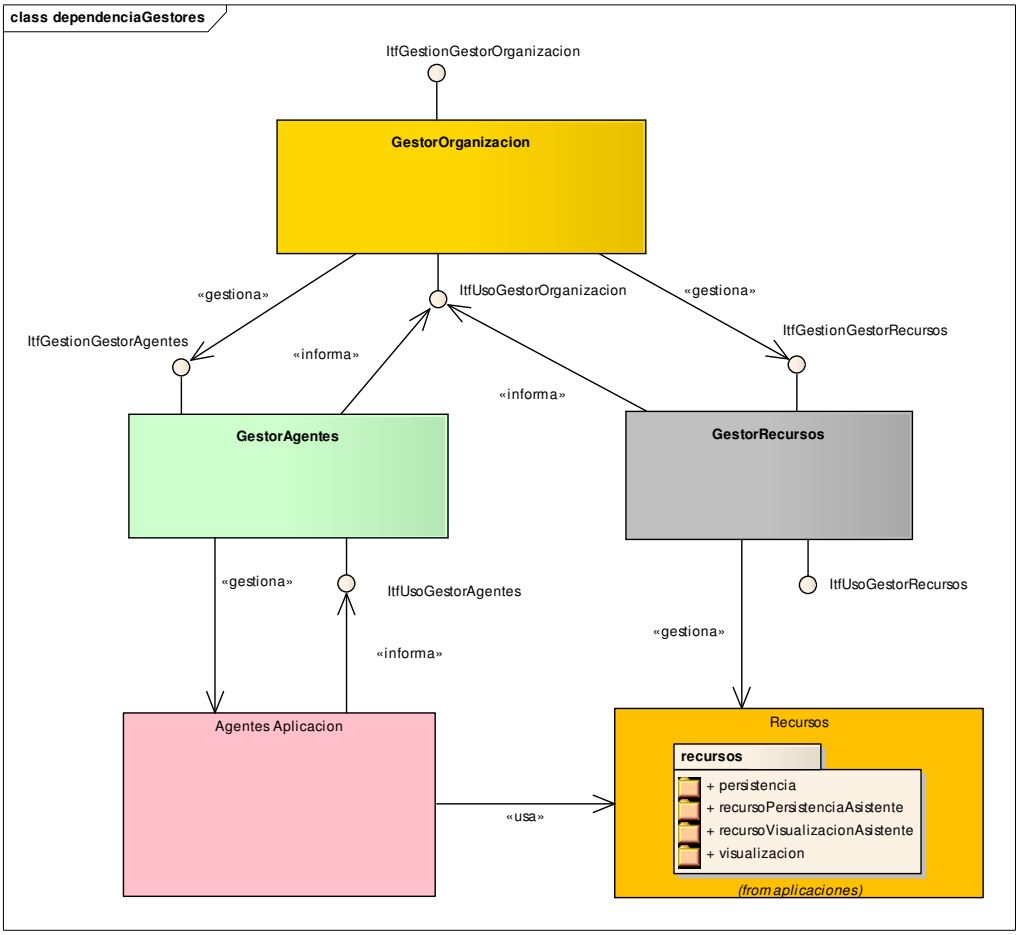


Figura 8 Dependencias gestores

La capa de recursos alberga diferentes tipos de recursos, como por ejemplo recursos de persistencia y visualización. En la plataforma existen dos recursos disponibles para cualquier aplicación, el directorio de organización, un repositorio que contiene la descripción de la organización, esto es nombre y tipo de los agentes y recursos de la organización, sus interfaces, propiedades específicas y nodos de despliegue. El directorio se utiliza para crear una instancia de la aplicación y posteriormente para guardar y validar las relaciones entre los miembros de la organización. El otro recurso que ofrece la plataforma es un recurso de trazas unificado para toda la plataforma.

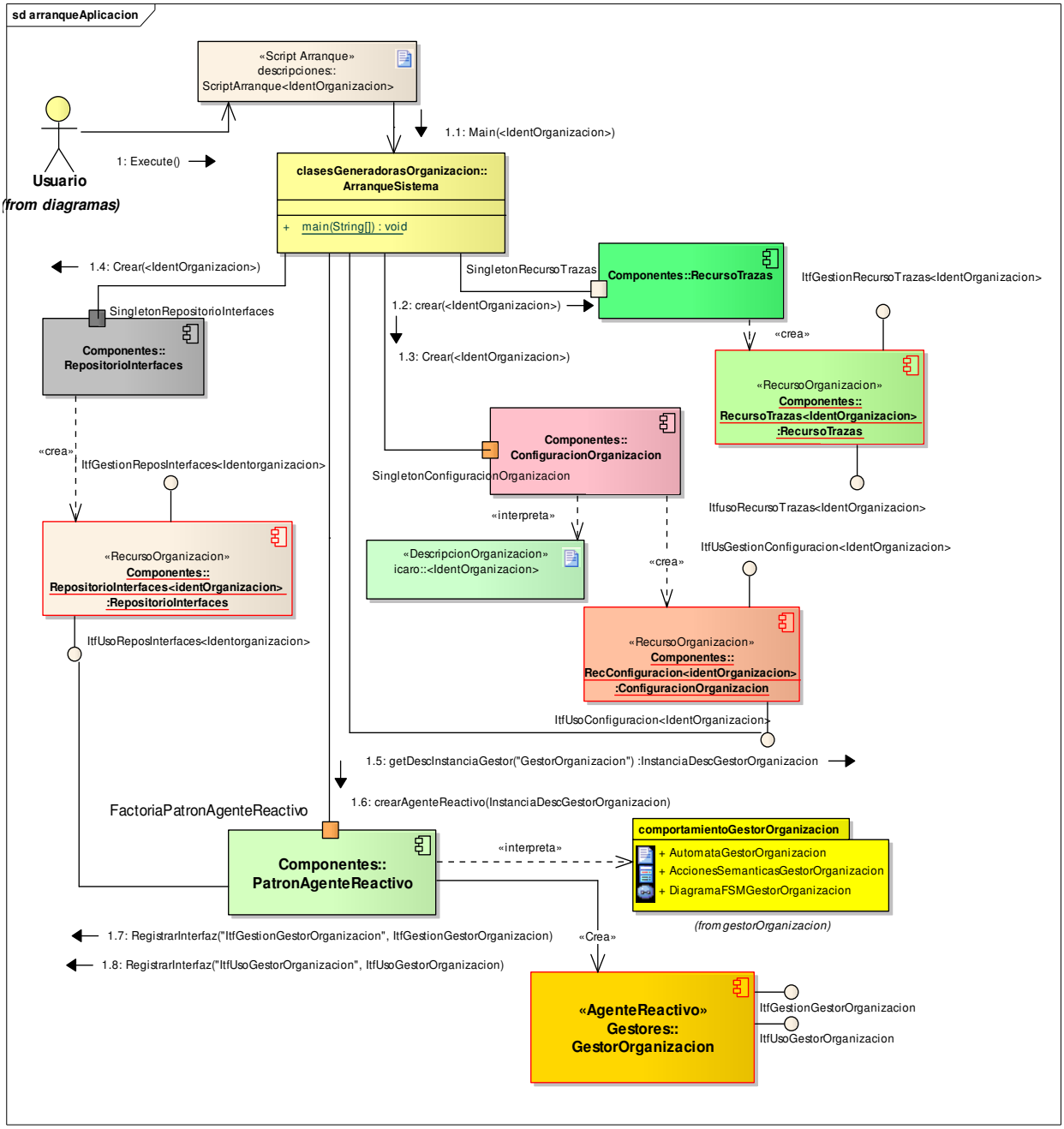


Figura 9 Arranque aplicación

La Figura 9 representa el diagrama de arranque de una aplicación en ICARO-T. El usuario arranca el sistema, pasándole como parámetro una descripción de organización. La clase generadora del sistema crea tres componentes, el recurso de trazas, el componente de configuración y el repositorio de interfaces. El componente de configuración interpreta la descripción de la organización. Una vez interpretada se

ordena al patrón de agente reactivo que cree el agente gestor de organización que, una vez creado, toma el control de la creación del resto de componentes de la organización.

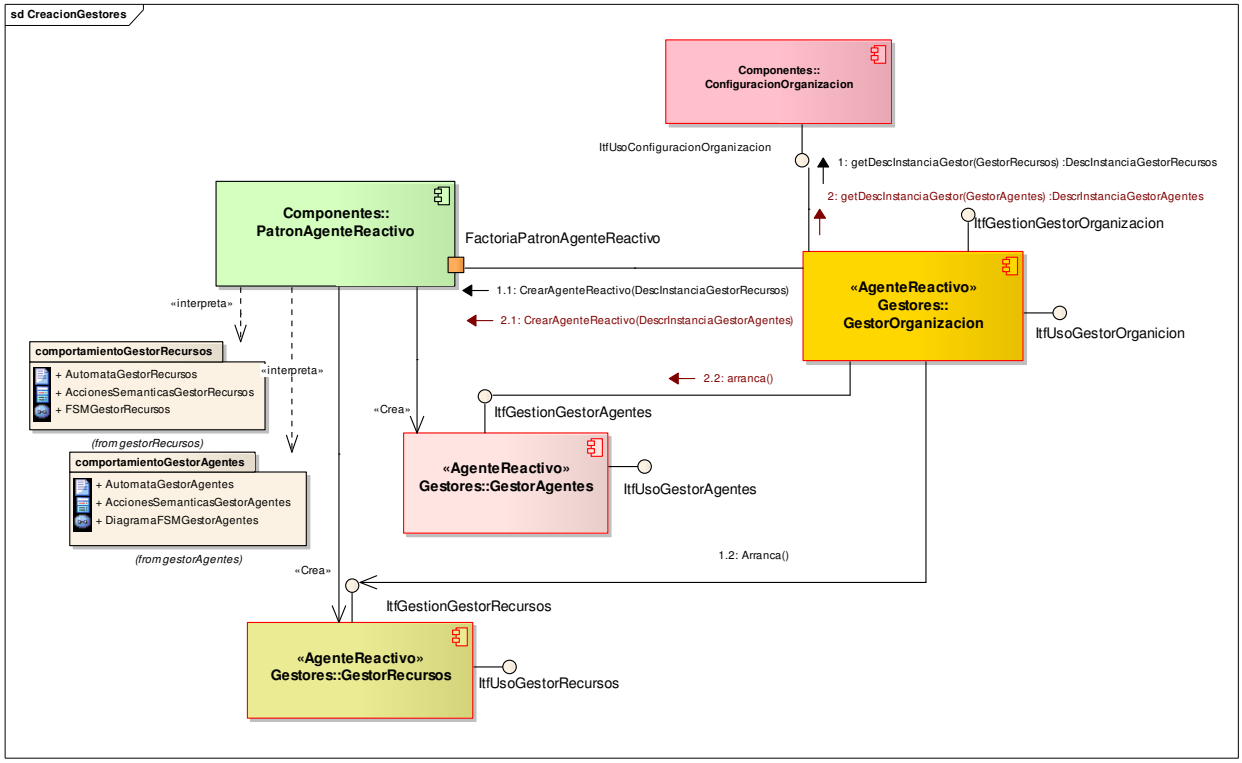


Figura 10 Creación de los gestores

En la Figura 10 se describe la continuación del proceso de arranque del sistema. El proceso sigue con las responsabilidades de cada agente. El gestor de organización crea, arranca y posteriormente gestiona el gestor de recursos y el gestor de agentes, mediante el patrón de agente reactivo. Una vez han sido creados, tanto el gestor de recursos como el de agentes toman el control de la creación de los componentes que deben gestionar, recursos y agentes respectivamente. De manera análoga al gestor de organización, los crean, los arrancan y los gestionan (Figura 11).

Como se ha explicado anteriormente cada componente reporta su estado a la entidad que lo ha creado y es esta última la que debe tomar decisiones en función de la información que recibe. Los agentes reportan al gestor de agentes, los recursos al gestor de recursos y el gestor de agentes y el gestor de recursos al gestor de organización.

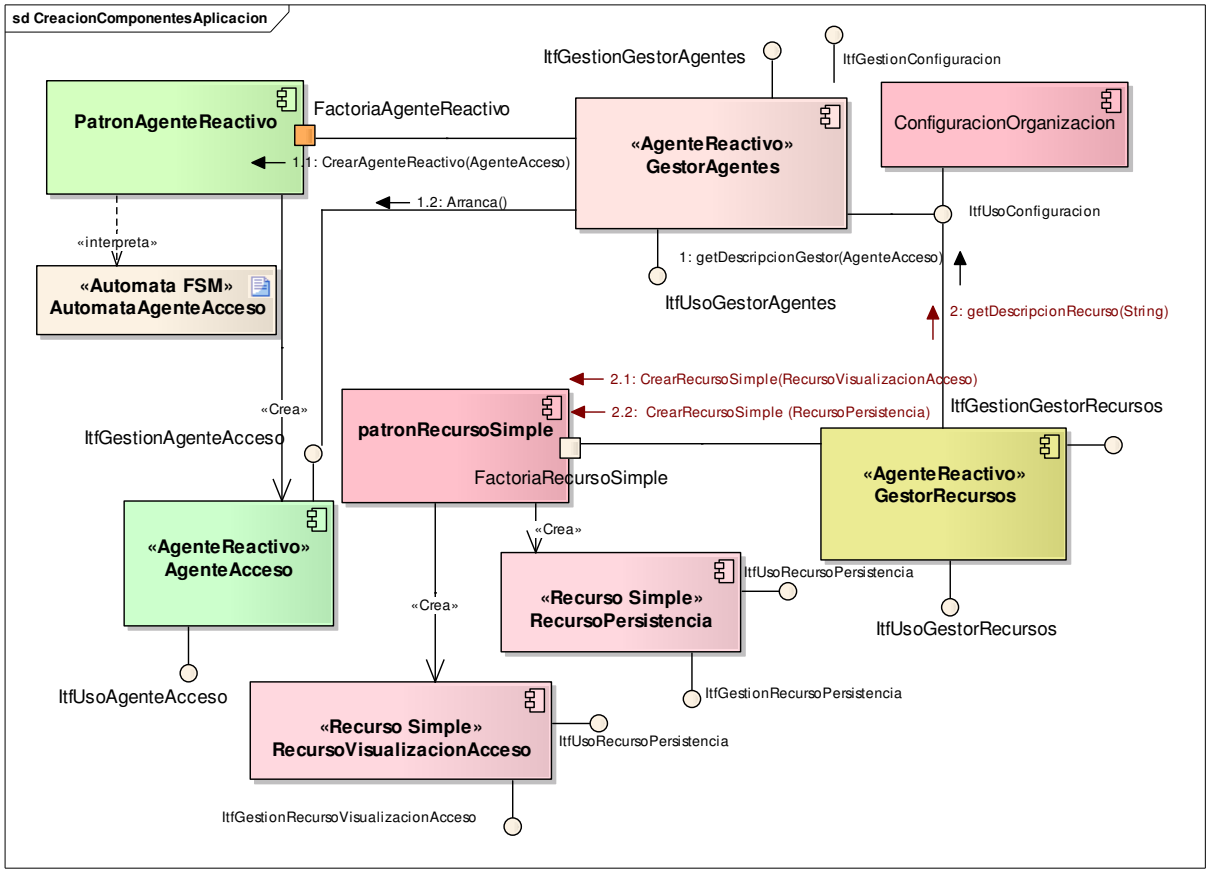


Figura 11 Creación de los componentes

3.3. Conclusiones

ICARO-T es un framework que proporciona patrones de diseño y arquitecturales para facilitar la construcción de aplicaciones distribuidas como sociedades de agentes. Existen muchas ventajas que se derivan de la utilización de patrones, en el caso concreto de ICARO-T se libra al desarrollador de preocuparse de la inicialización, monitorización o finalización del sistema. Asuntos que no son para nada triviales en el desarrollo de aplicaciones. Por otra parte, el uso de patrones también implica una serie de restricciones o requisitos a la hora de desarrollar. Se impone una manera de realizar las cosas, aunque esté basada en buenas prácticas de desarrollo software. Prueba de ello es que las aplicaciones realizadas en ICARO-T son sistemas con un muy bajo grado de acoplamiento gracias al uso de patrones y componentes que, además son luego fácilmente reutilizables.

El framework es extensible y en este trabajo se aborda facilitar la configuración de las organizaciones. ICARO proporciona actualmente lo esencial pero hay algunos aspectos que están implementados solo a bajo nivel. Por ejemplo la configuración del sistema se especifica como ficheros XML y sería más apropiado disponer de herramientas que permitieran hacerlo de manera más sencilla que editar directamente ese tipo de ficheros. Así surge la idea de realizar un asistente de configuración de organizaciones en este trabajo, que se expone en el capítulo siguiente. Además este trabajo servirá para plantear la realización de otros servicios de organización (trabajo futuro).

4. ASISTENTE DE ORGANIZACIONES PARA ICARO-T

El objetivo de este trabajo consiste en facilitar la definición de organizaciones para ICARO-T, la generación de nuevos componentes y su validación en función de los agentes que participan en la organización y de los recursos necesarios para alcanzar los objetivos de funcionamiento. Para ello se ha desarrollado una aplicación asistente que analiza el código de la infraestructura e interpreta ficheros de descripción.

El asistente se desarrolla como un componente funcional de la infraestructura de la plataforma ICARO-T, implementado con los mismos patrones y la misma metodología de desarrollo (Figura 12). Como parte de la metodología se identifican en la arquitectura del asistente las tres capas en las que se divide cualquier aplicación de ICARO-T, capara de gestión, capa de aplicación y capa de recursos.

Este componente tiene dos formas de activación:

- El desarrollador arranca el asistente manualmente para generar el fichero de descripción de una aplicación.
- La plataforma, al arrancar con un fichero de descripción mal construido, detecta un error y pregunta al desarrollador si quiere arrancar el asistente para ayudarle a encontrar los errores del fichero.

Al estar desarrollado de la misma manera que una aplicación de la plataforma, posee los mismos componentes que ésta (agentes, recursos, clases de dominio y un fichero de descripción). La funcionalidad del asistente está incorporada a la funcionalidad básica de ICARO al mismo nivel que la creación de los gestores o la de los componentes dinámicos a partir de la descripción de la organización.

En este capítulo se va a describir el diseño del asistente, los componentes que lo forman y una muestra de funcionamiento del mismo. Como parte del diseño se van a introducir los casos de uso del asistente y los diagramas de actividad e interacción relacionados con las actividades más relevantes.

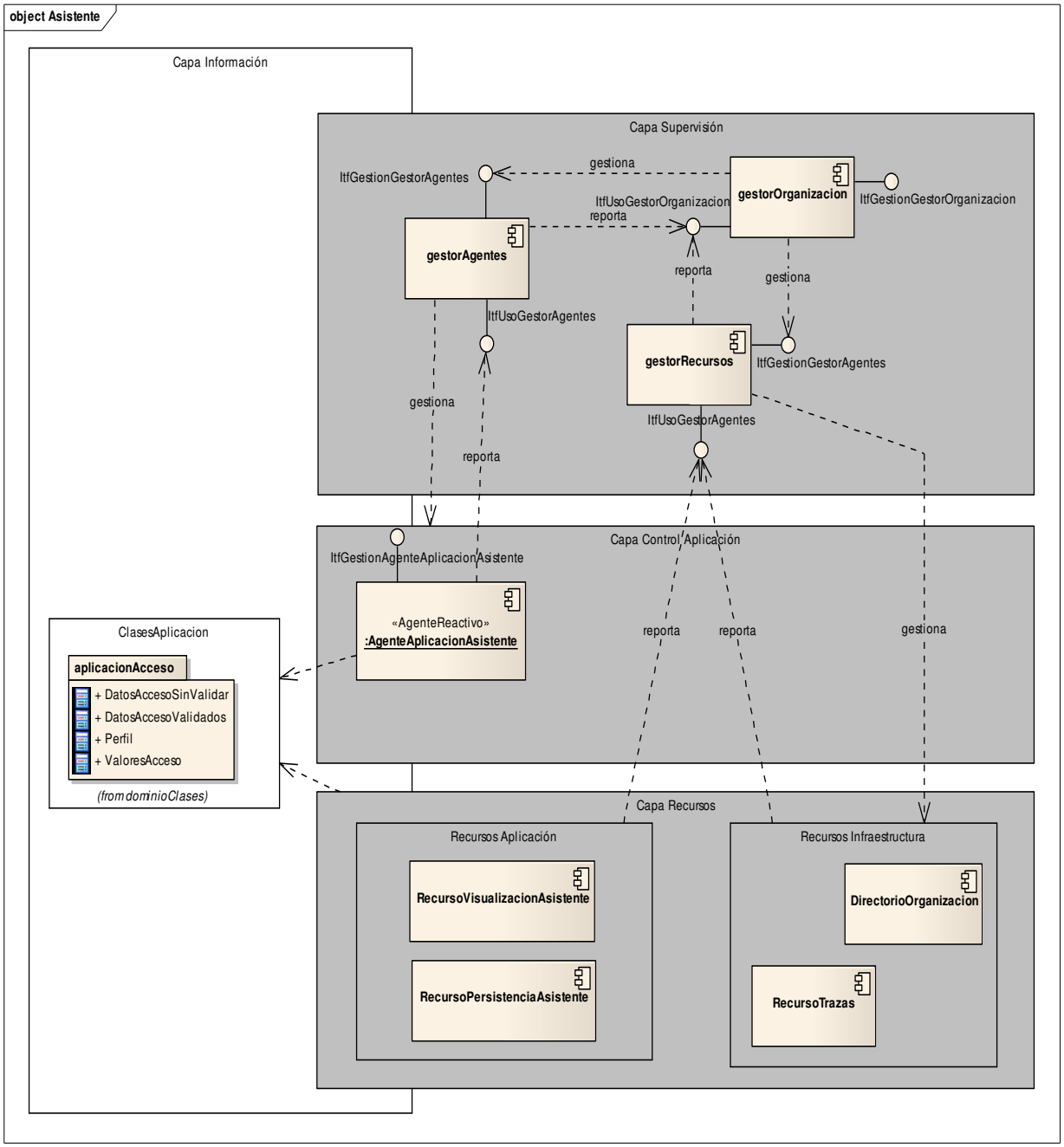


Figura 12 Arquitectura aplicación ICARO-T

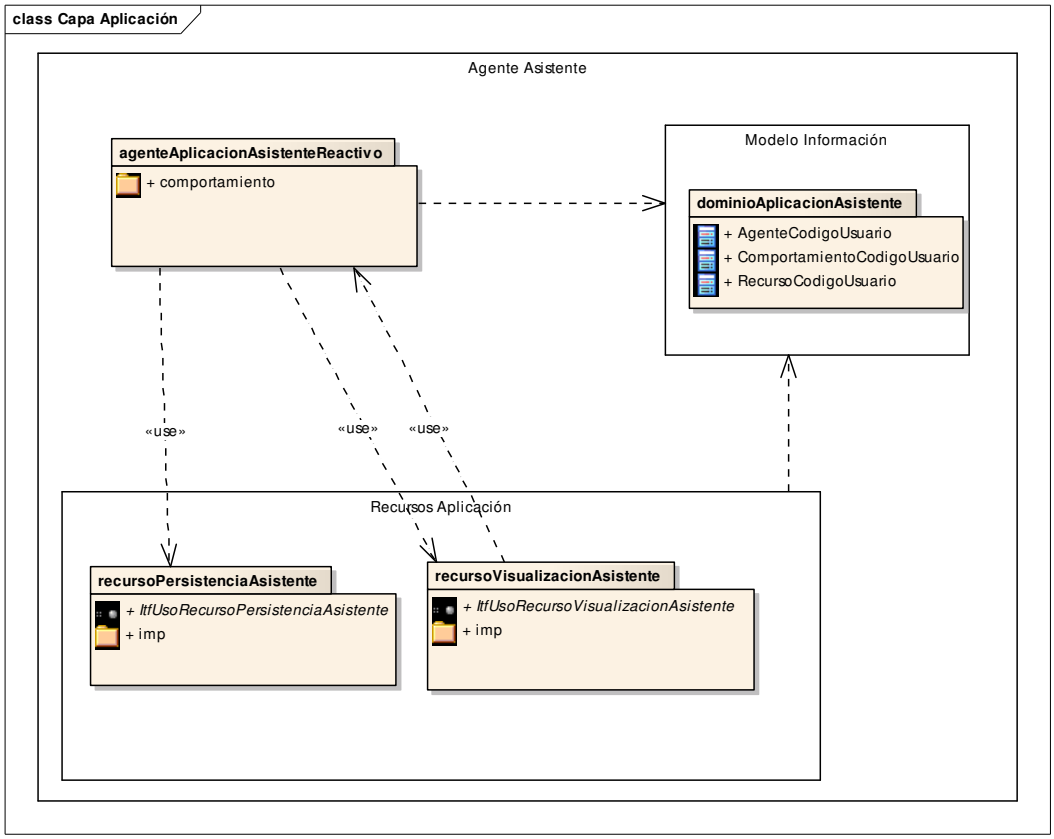


Figura 13 Componentes aplicación ICARO-T

4.1. Diseño del asistente

La aplicación asistente, como cualquier aplicación de ICARO-T, se divide en agentes, recursos y clases de dominio (Figura 13). Más concretamente el asistente está compuesto por un agente de aplicación de tipo reactivo, denominado AgenteAplicaciónAsistente, dos recursos de aplicación, uno de persistencia y otro de visualización, respectivamente y unas clases de dominio.

4.1.1. Diseño

El asistente ha sido diseñado para, por un lado ser capaz de interpretar descripciones de organizaciones de la plataforma ICARO-T y presentárselas al usuario, crear y modificar estas descripciones y por último poder analizar el código de la infraestructura para detectar los componentes que hay desarrollados en el sistema (agentes y recursos).

4.1.1.1. Casos de uso

En la Figura 14 se observa el diagrama de casos de uso del asistente. En este diagrama se pueden ver todas las operaciones que el usuario puede hacer sobre el sistema: arranque y terminación del asistente, creación, modificación y borrado de componentes.

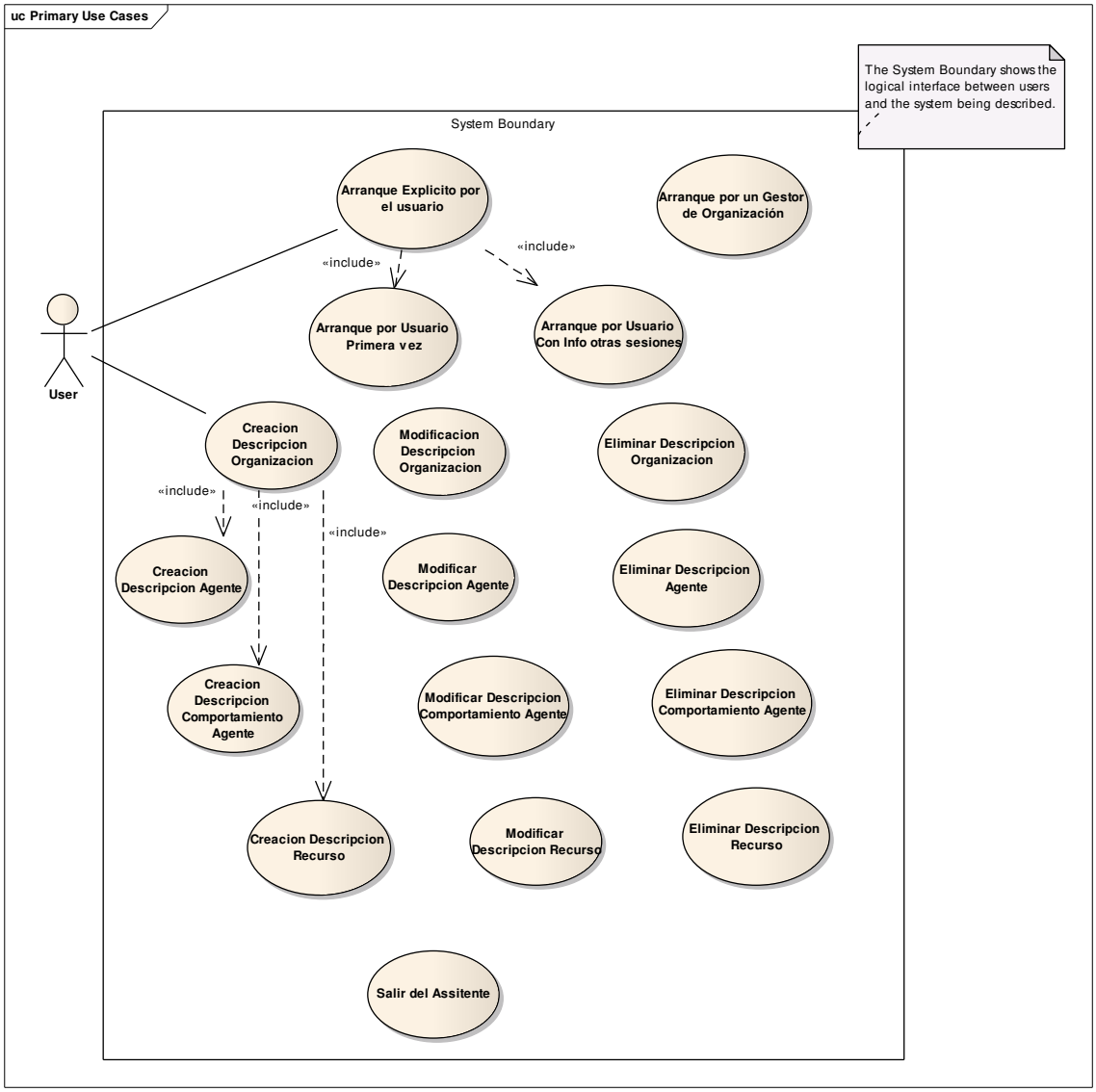


Figura 14 Casos de uso asistente

A continuación se muestran los flujos de actividades más relevantes permitidos para el usuario. En el diagrama (Figura 15) se observan tres flujos diferentes. El primero es el relacionado con la gestión de las descripciones de organizaciones, el segundo está

relacionado con la gestión de los elementos de la organización y el último flujo muestra la gestión de instancias de los elementos. A través de estos diagramas se pueden ver el orden en que el usuario realiza las acciones y las acciones que se generan a raíz de éstas.

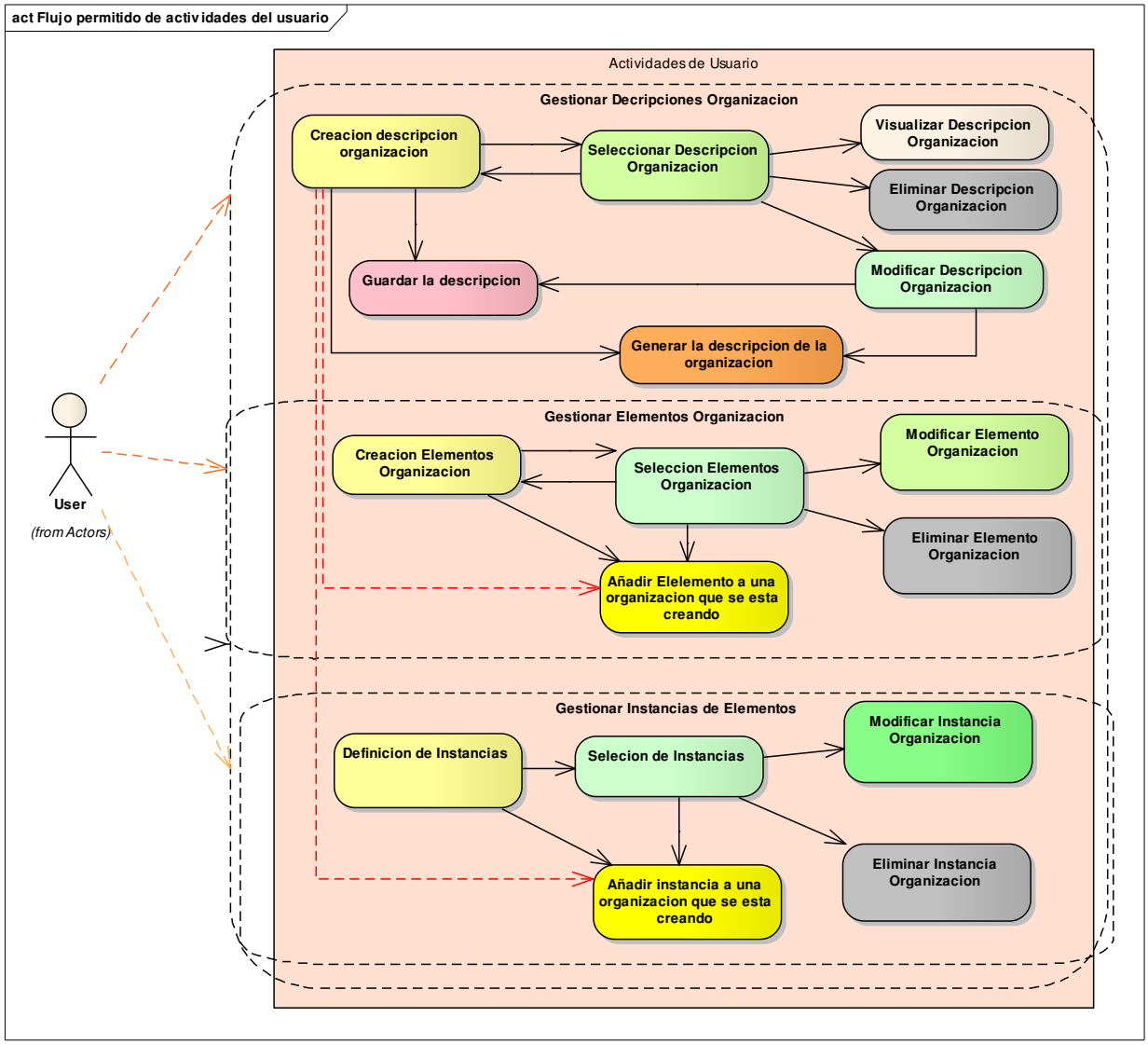


Figura 15 Flujos de actividad

4.1.1.2. Diagramas de actividad

A través de los diagramas de actividad de algunas de las operaciones que puede realizar el usuario en el sistema se observan los módulos que intervienen en estas acciones y las interacciones que se dan entre los mismos.

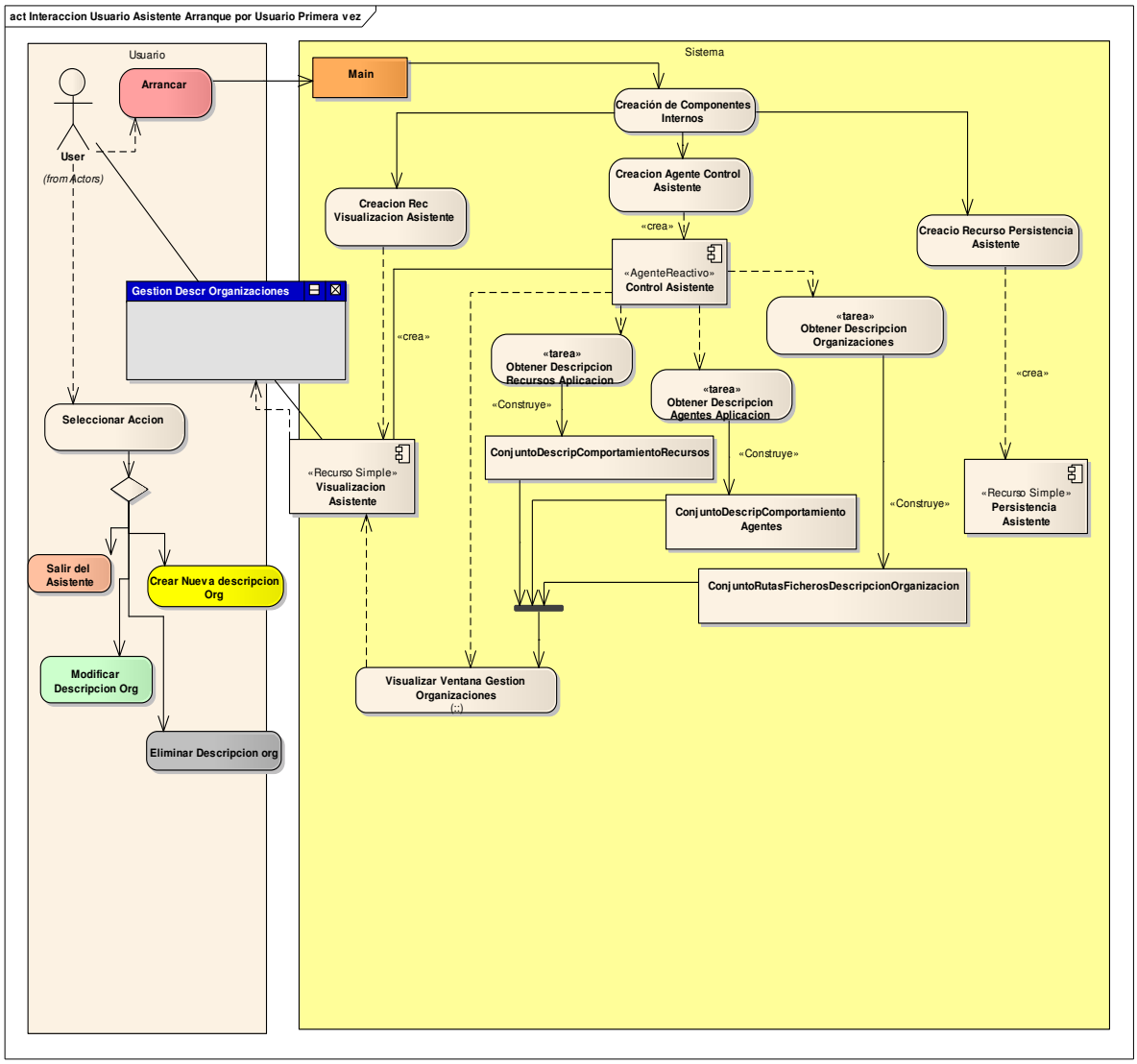


Figura 16 Diagrama de actividad arranque primera vez.

En la Figura 16 se presenta el diagrama de interacción que muestra las acciones derivadas del primer arranque del sistema, bajo demanda del usuario. En el diagrama se pueden ver representados los tres componentes que utiliza el asistente, el agente reactivo que representa el control del asistente y los recursos de persistencia y visualización. Además de los componentes, se marcan las acciones derivadas de la interacción del usuario y las que se ejecutan a raíz de éstas.

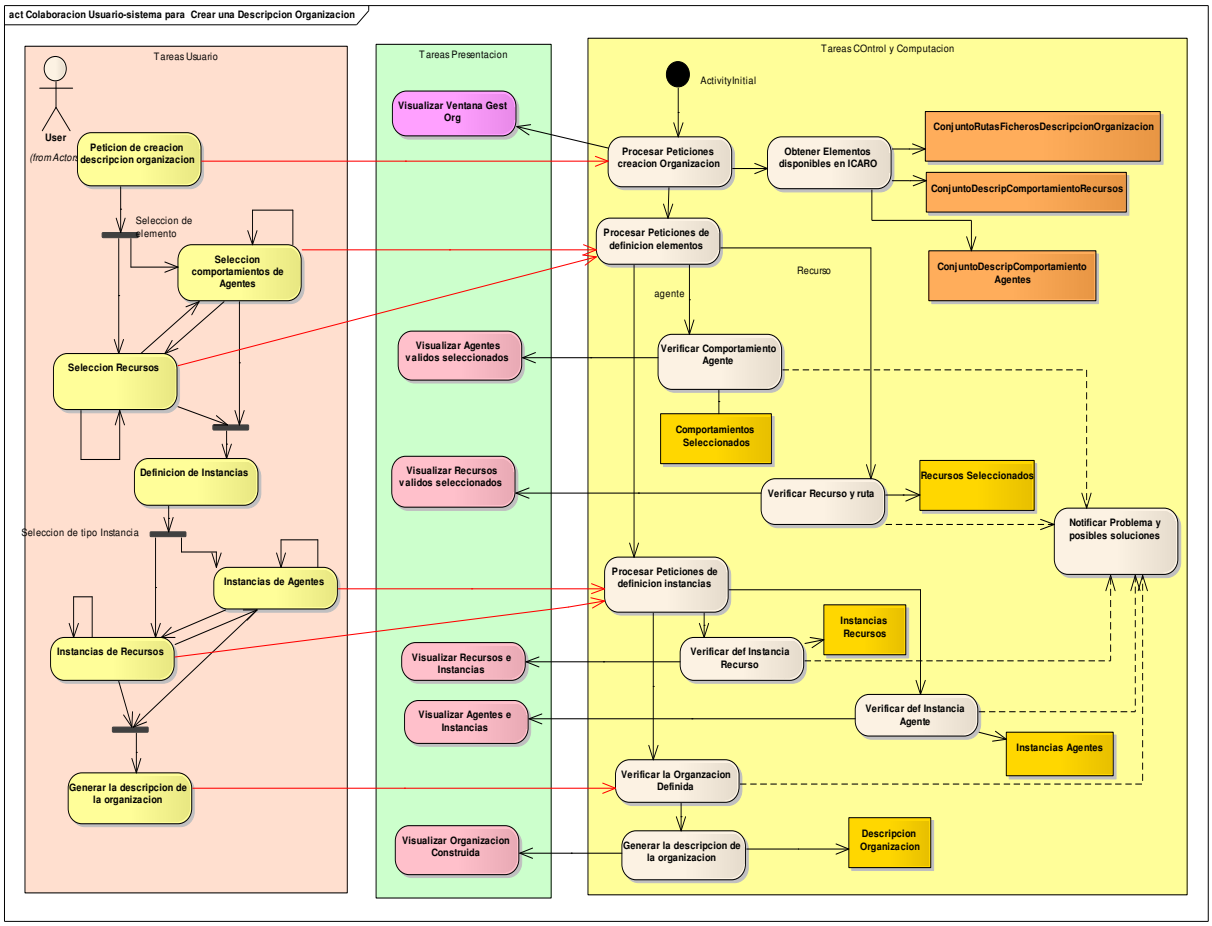


Figura 17 Diagrama de actividad crear descripción

La Figura 17 muestra el diagrama de actividad que representa las acciones, a alto nivel, relativas a crear una nueva descripción de organización de principio a fin. En este diagrama se distinguen tres partes: la parte en la que se muestran las tareas relacionadas con el usuario, las tareas relacionadas con la visualización y las tareas relacionadas con el control y la computación. Gracias a este diagrama se puede relacionar muy fácilmente las acciones que realiza el usuario con las acciones de control que se ejecutan y la visualización que se genera por pantalla.

4.1.1.3. Diagramas de comportamiento

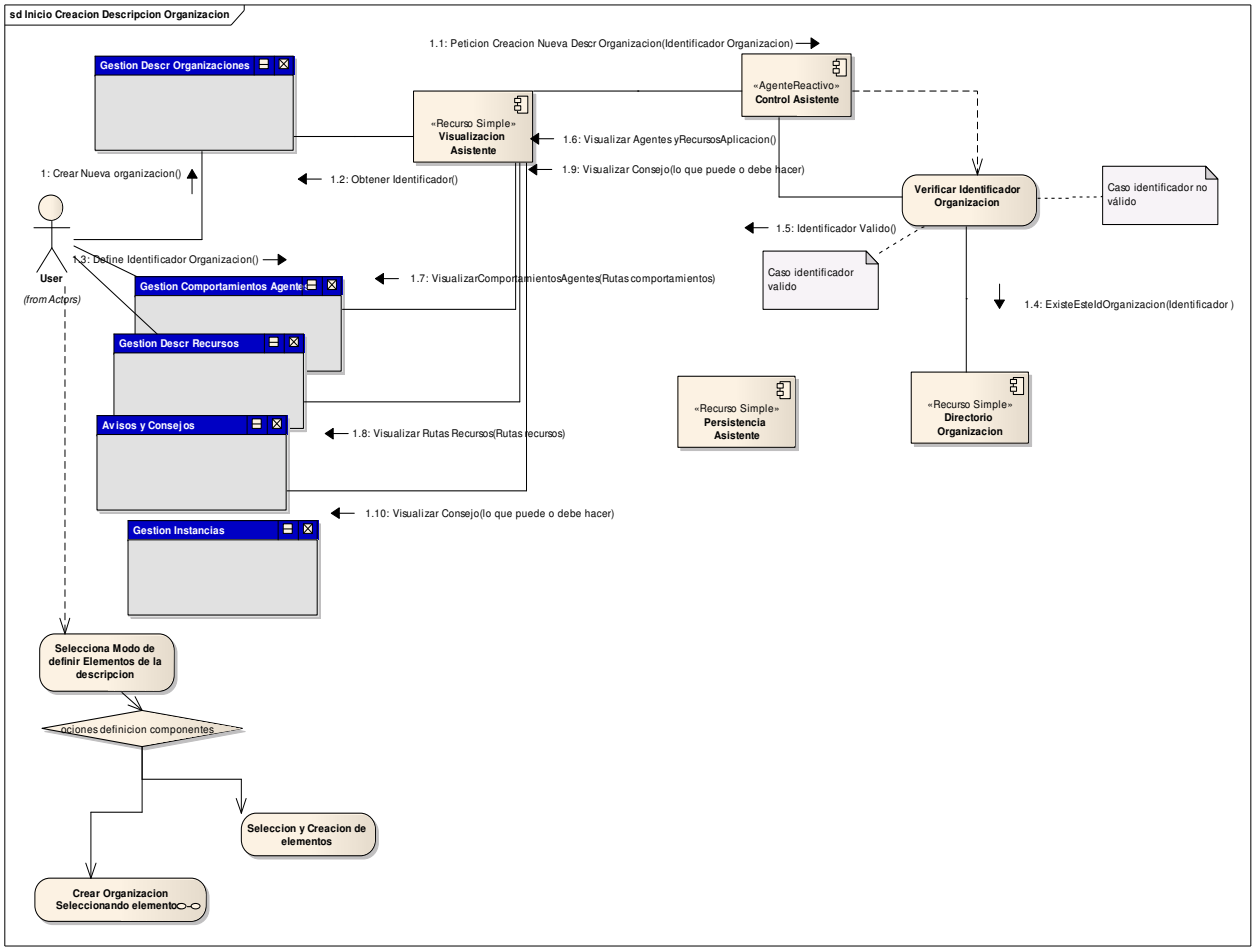


Figura 18 Comportamiento creación descripción organización

En la Figura 18 se muestra el diagrama de comportamiento asociado a la creación de una descripción. Se pueden identificar las interacciones del usuario con el interfaz del sistema y las interacciones que se generan entre los componentes internos del sistema que se derivan de las primeras.

En la Figura 19 se presenta el diagrama de comportamiento que se deriva de la selección de los componentes para la generación de una descripción de organización. De manera análoga al caso anterior, se pueden observar las interacciones que se producen en el sistema para llevar a cabo los deseos del usuario.

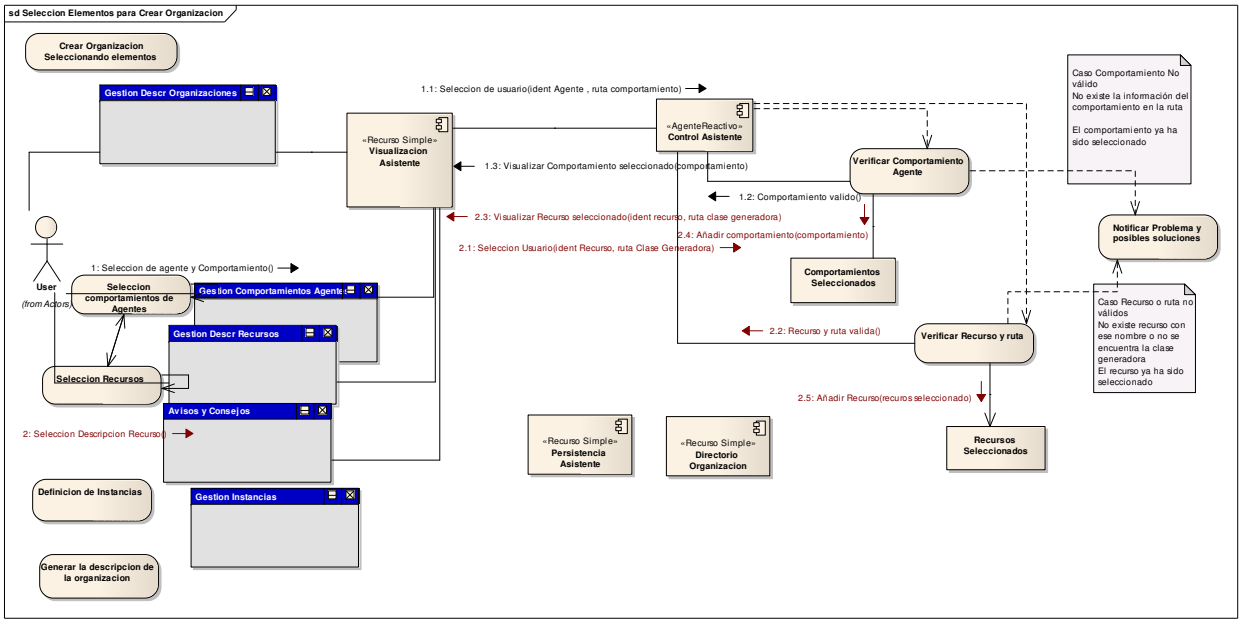


Figura 19 Comportamiento selección elementos creación organización

4.1.2. Agente

El agente que forma parte del asistente es un agente de aplicación de tipo reactivo. Como todos los agentes reactivos de ICARO-T, el comportamiento del agente se describe con un autómata de estados finitos. En la plataforma para definir el comportamiento de un agente se crea un fichero XML que representa el autómata y una clase Java denominada que contiene las acciones semánticas, que son las acciones que se ejecutan en cada transición del autómata.

Para la creación de autómatas de control de agentes reactivos se ha utilizado un meta-modelo (Figura 20). Este meta-modelo representa las características y restricciones que debe cumplir un autómata de control de un agente reactivo. Este esquema se traduce a una entidad computacional a través de un DTD (Figura 21). En este tipo de entidades se representan tres cosas:

- Elementos: Qué etiquetas son permitidas y el contenido de dichas etiquetas.
- Estructura: El orden de las etiquetas dentro del documento.
- Anidamiento: Indica qué etiquetas van dentro de otras.

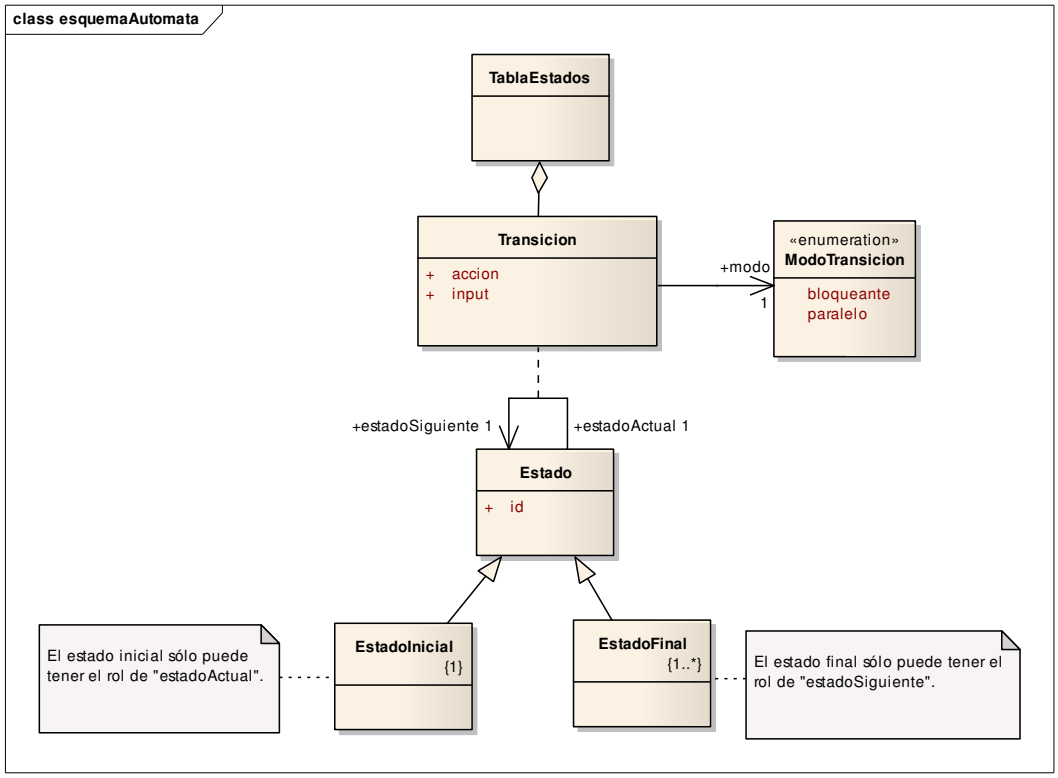


Figura 20 Esquema para modelar autómatas

```
<!ELEMENT tablaEstados (estadoInicial, estado*, estadoFinal+, transicionUniversal*)>
<!ATTLIST tablaEstados
  descripcionTabla CDATA #IMPLIED
>
<!ELEMENT estadoInicial (transicion+)>
<!ATTLIST estadoInicial
  idInicial ID #REQUIRED
>
<!ELEMENT estado (transicion+)>
<!ATTLIST estado
  idIntermedio ID #REQUIRED
>
<!ELEMENT estadoFinal EMPTY>
<!ATTLIST estadoFinal
  idFinal ID #REQUIRED
>
<!ELEMENT transicionUniversal EMPTY>
<!ATTLIST transicionUniversal
  input CDATA #REQUIRED
  accion CDATA #REQUIRED
  estadoSiguiente IDREF #REQUIRED
  modoDeTransicion (bloqueante | paralelo) #REQUIRED
>
<!ELEMENT transicion EMPTY>
<!ATTLIST transicion
  input CDATA #REQUIRED
  accion CDATA #REQUIRED
  estadoSiguiente IDREF #REQUIRED
  modoDeTransicion (bloqueante | paralelo) #REQUIRED
>
```

Figura 21 DTD esquema de modelado de autómatas

En el meta-modelo se observa que es necesario tener un solo estado inicial, al menos un estado final y todos los estados, transiciones y transiciones universales que necesitemos.

Las transiciones universales se distinguen de las transiciones estándar en que la transición es válida para cualquier estado del autómata; es decir, cuando llega el input especificado, se ejecutan las acciones y se transita al estado siguiente, independientemente del estado en que se encuentre el autómata. El uso de transiciones universales evita tener que definir la transición en todos los estados del autómata.

El autómata que representa el comportamiento del agente del asistente está compuesto por 17 estados, contando el estado inicial y el estado final. Las transiciones del autómata producen 14 acciones diferentes, que son los métodos de la clase Java que representa las acciones semánticas.

Los primeros estados son los correspondientes al arranque del asistente (Figura 22), una vez que se ha recibido el input "comenzar" se ejecuta el arranque del sistema y se pasa al estado "esperaAccionInicial" a partir del cual se pueden recibir dos inputs diferentes. Bien el usuario ha optado por abrir una descripción ya existente o por crear una nueva partiendo de cero. En el caso de que se abra una existente se pasa al estado de comprobación de la descripción. A continuación, si todo ha ido bien se pasa al estado en el que se muestran los componentes.

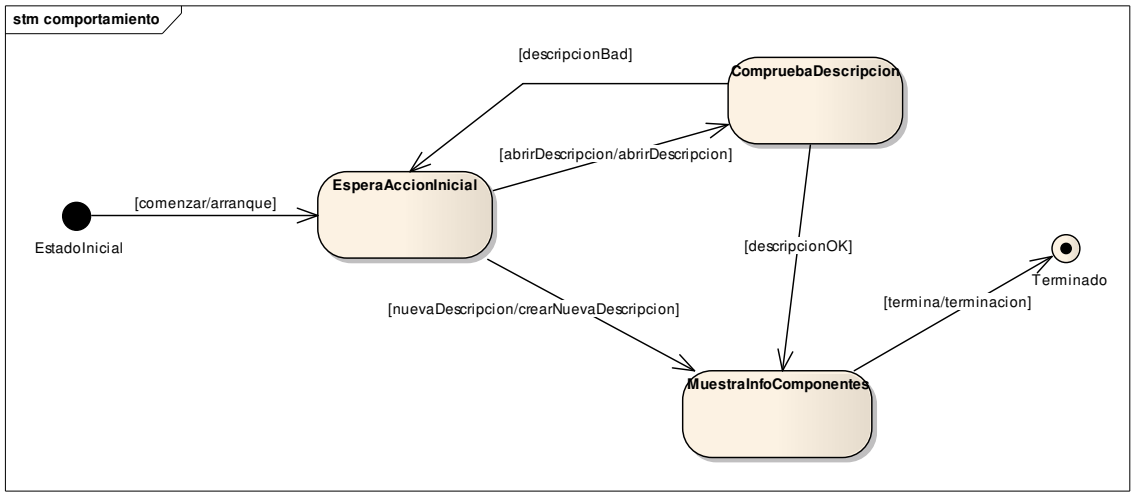
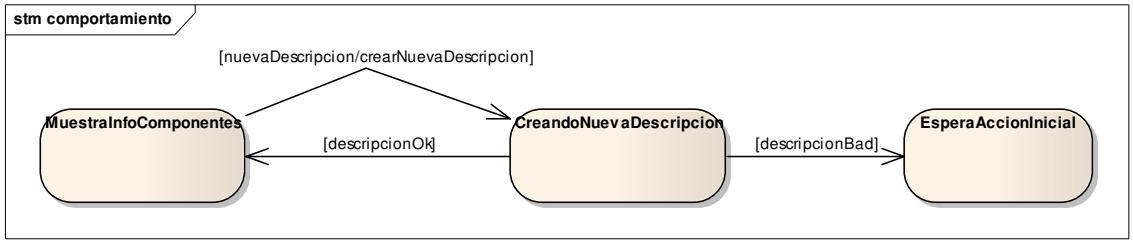
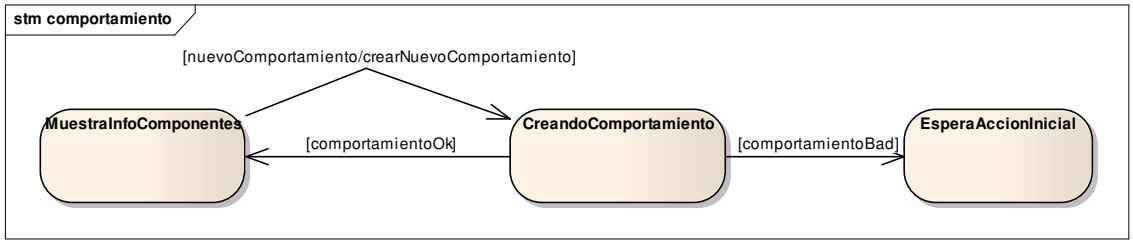


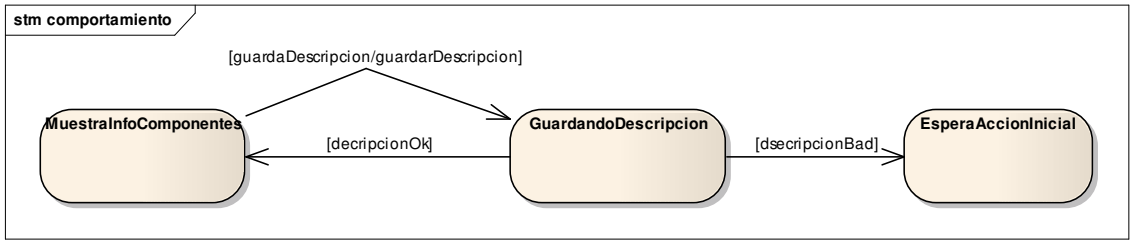
Figura 22 Autómata de arranque del asistente

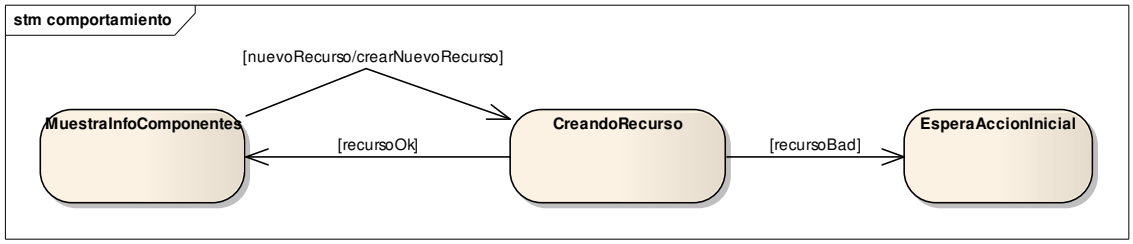
Desde el estado de muestra de los componentes se pasa a esperar las ordenes del usuario con respecto a los componentes, creación, modificación o borrado de agentes, comportamientos o recursos (Figura 23). La representación de este tipo de autómatas es bastante similar en todos los casos. Una vez que se recibe el input para ejecutar la acción asociada se transita a un estado intermedio en el que se comprueba que la operación ha finalizado correctamente. Si la acción se ha ejecutado bien, se vuelve al estado de muestra de los componentes. En caso contrario se transita al inicio del asistente.

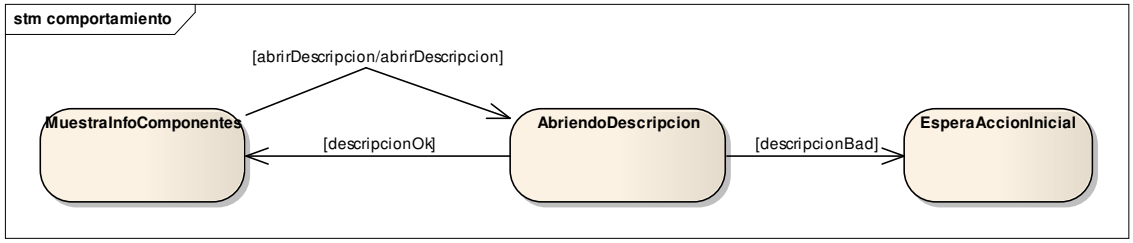
Este tipo de autómata es muy utilizado en los desarrollos de interfaces de usuario, como es el caso que nos ocupa. Partiendo de un estado inicial, en este caso el estado de muestra de la información, se desea ejecutar una acción y se crea un estado intermedio para comprobar la correcta ejecución de la acción.

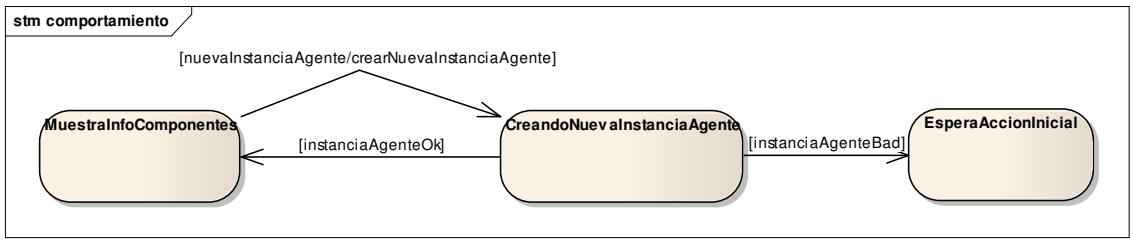


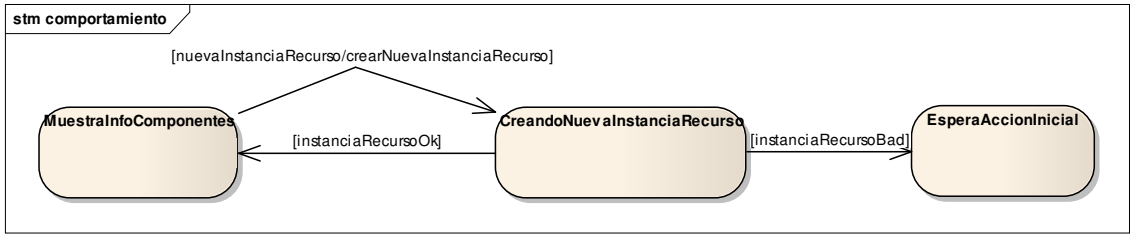


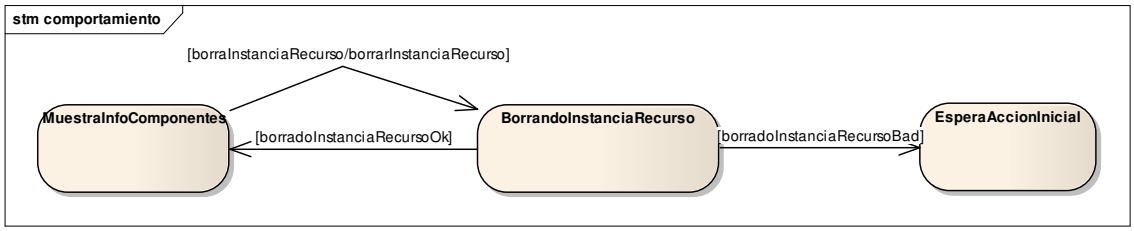


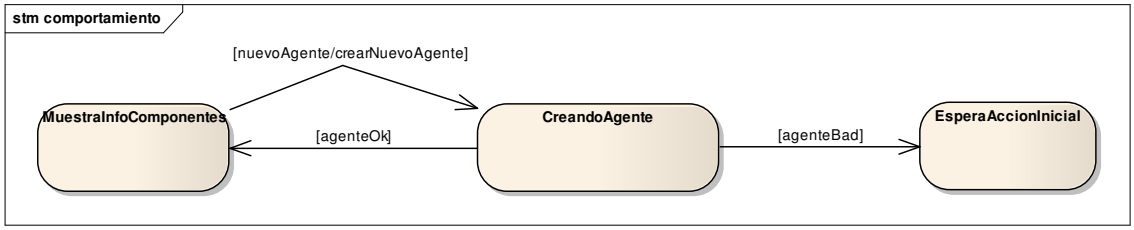

stm comportamiento
[nuevoRecurso/crearNuevoRecurso]
MuestraInfoComponentes
[recursoOk]
CreandoRecurso
[recursoBad]
EsperaAccionInicial


stm comportamiento
[abrirDescripcion/abrirDescripcion]
MuestraInfoComponentes
[descripcionOk]
AbriendoDescripcion
[descripcionBad]
EsperaAccionInicial


stm comportamiento
[nuevaInstanciaAgente/crearNuevaInstanciaAgente]
MuestraInfoComponentes
[instanciaAgenteOk]
CreandoNuevaInstanciaAgente
[instanciaAgenteBad]
EsperaAccionInicial


stm comportamiento
[nuevaInstanciaRecurso/crearNuevaInstanciaRecurso]
MuestraInfoComponentes
[instanciaRecursoOk]
CreandoNuevaInstanciaRecurso
[instanciaRecursoBad]
EsperaAccionInicial


stm comportamiento
[borraInstanciaRecurso/borrarInstanciaRecurso]
MuestraInfoComponentes
[borradoInstanciaRecursoOk]
BorrandoInstanciaRecurso
[borradoInstanciaRecursoBad]
EsperaAccionInicial


stm comportamiento
[nuevoAgente/crearNuevoAgente]
MuestraInfoComponentes
[agenteOk]
CreandoAgente
[agenteBad]
EsperaAccionInicial

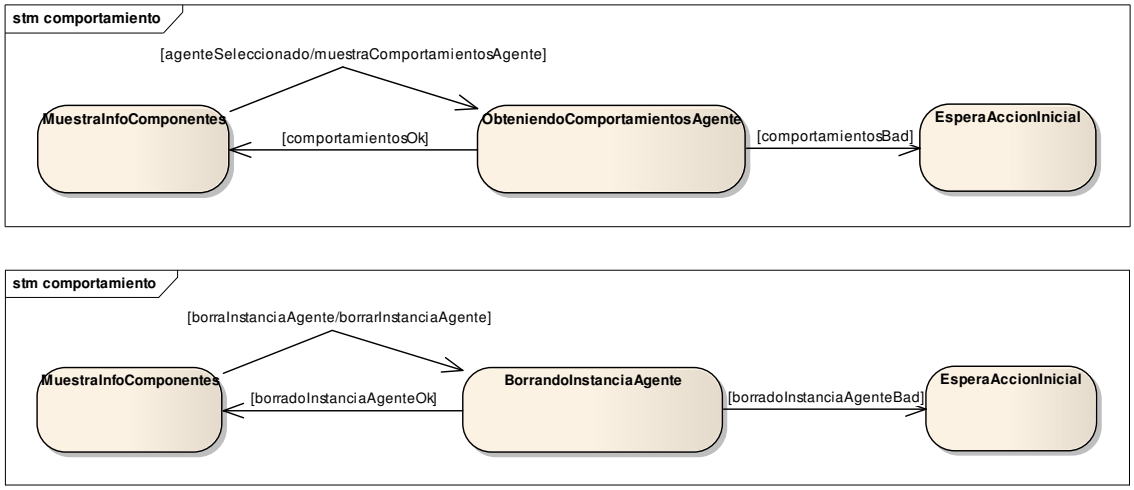


Figura 23 Autómata acciones asistente

A continuación (Figura 24) se puede ver el archivo XML que representa el autómata desarrollado para el agente reactivo del asistente.

```
<?xml version="1.0"?>

<!DOCTYPE tablaEstados SYSTEM "schemas/TablaEstados.dtd">

<tablaEstados descripcionTabla="Tabla de estados del agente del asistente">

   <estadoInicial idInicial="estadoInicial">
      <transicion input="comenzar" accion="arranque"
      estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
   </estadoInicial>
   <estado idIntermedio="esperaAccionInicial">
      <transicion input="nuevaDescripcion" accion="crearNuevaDescripcion"
      estadoSiguiente="muestraInfoComponentes" modoDeTransicion="bloqueante"/>
      <transicion input="abrirDescripcion" accion="abrirDescripcion"
      estadoSiguiente="compruebaDescripcion" modoDeTransicion="bloqueante"/>
   </estado>
   <estado idIntermedio="compruebaDescripcion">
      <transicion input="descripcionOk" accion="nula"
      estadoSiguiente="muestraInfoComponentes" modoDeTransicion="bloqueante"/>
      <transicion input="descripcionBad" accion="nula"
      estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
   </estado>
   <estado idIntermedio="muestraInfoComponentes">
      <transicion input="nuevaDescripcion" accion="crearNuevaDescripcion"
      estadoSiguiente="creandoNuevaDescripcion" modoDeTransicion="bloqueante"/>
      <transicion input="abrirDescripcion" accion="abrirDescripcion"
      estadoSiguiente="abriendoDescripcion" modoDeTransicion="bloqueante"/>
      <transicion input="agenteSeleccionado"
      accion="muestraComportamientosAgente"
      estadoSiguiente="obteniendoComportamientosAgente"
      modoDeTransicion="bloqueante"/>
```

```
      <transicion input="nuevaInstanciaAgente"
      accion="crearNuevaInstanciaAgente"
      estadoSiguiente="creandoNuevaInstanciaAgente"
      modoDeTransicion="bloqueante"/>
      <transicion input="nuevaInstanciaRecurso"
      accion="crearNuevaInstanciaRecurso"
      estadoSiguiente="creandoNuevaInstanciaRecurso"
      modoDeTransicion="bloqueante"/>
      <transicion input="guardaDescripcion" accion="guardarDescripcion"
      estadoSiguiente="guardandoDescripcion" modoDeTransicion="bloqueante"/>
      <transicion input="borraInstanciaAgente" accion="borrarInstanciaAgente"
      estadoSiguiente="borrandoInstanciaAgente" modoDeTransicion="bloqueante"/>
      <transicion input="borraInstanciaRecurso" accion="borrarInstanciaRecurso"
      estadoSiguiente="borrandoInstanciaRecurso"
      modoDeTransicion="bloqueante"/>
      <transicion input="nuevoRecurso" accion="crearNuevoRecurso"
      estadoSiguiente="creandoRecurso" modoDeTransicion="bloqueante"/>
      <transicion input="nuevoAgente" accion="crearNuevoAgente"
      estadoSiguiente="creandoAgente" modoDeTransicion="bloqueante"/>
      <transicion input="nuevoComportamiento" accion="crearNuevoComportamiento"
      estadoSiguiente="creandoComportamiento" modoDeTransicion="bloqueante"/>
   </estado>
   <estado idIntermedio="creandoNuevaDescripcion">
      <transicion input="descripcionOk" accion="nula"
      estadoSiguiente="muestraInfoComponentes" modoDeTransicion="bloqueante"/>
      <transicion input="descripcionBad" accion="nula"
      estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
   </estado>
   <estado idIntermedio="abriendoDescripcion">
      <transicion input="descripcionOk" accion="nula"
      estadoSiguiente="muestraInfoComponentes" modoDeTransicion="bloqueante"/>
      <transicion input="descripcionBad" accion="nula"
      estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
   </estado>
   <estado idIntermedio="obteniendoComportamientosAgente">
      <transicion input="comportamientosOk" accion="nula"
      estadoSiguiente="muestraInfoComponentes" modoDeTransicion="bloqueante"/>
      <transicion input="comportamientosBad" accion="nula"
      estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
      </estado>
   <estado idIntermedio="creandoNuevaInstanciaAgente">
      <transicion input="instanciaAgenteOk" accion="nula"
      estadoSiguiente="muestraInfoComponentes" modoDeTransicion="bloqueante"/>
      <transicion input="instanciaAgenteBad" accion="nula"
      estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
   </estado>
   <estado idIntermedio="creandoNuevaInstanciaRecurso">
      <transicion input="instanciaRecursoOk" accion="nula"
      estadoSiguiente="muestraInfoComponentes" modoDeTransicion="bloqueante"/>
      <transicion input="instanciaRecursoBad" accion="nula"
      estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
   </estado>
   <estado idIntermedio="guardandoDescripcion">
      <transicion input="descripcionOk" accion="nula"
      estadoSiguiente="muestraInfoComponentes" modoDeTransicion="bloqueante"/>
      <transicion input="descripcionBad" accion="nula"
      estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
   </estado>
```

```
    <estado idIntermedio="borrandoInstanciaAgente">
       <transicion input="borradoInstanciaAgenteOk" accion="nula"
       estadoSiguiente="muestraInfoComponentes" modoDeTransicion="bloqueante"/>
       <transicion input="borradoInstanciaAgenteBad" accion="nula"
       estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
    </estado>
    <estado idIntermedio="borrandoInstanciaRecurso">
       <transicion input="borradoInstanciaRecursoOk" accion="nula"
       estadoSiguiente="muestraInfoComponentes" modoDeTransicion="bloqueante"/>
       <transicion input="borradoInstanciaRecursoBad" accion="nula"
       estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
    </estado>
    <estado idIntermedio="creandoRecurso">
       <transicion input="descripcionOk" accion="nula"
       estadoSiguiente="muestraInfoComponentes" modoDeTransicion="bloqueante"/>
       <transicion input="descripcionBad" accion="nula"
       estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
    </estado>
    <estado idIntermedio="creandoAgente">
       <transicion input="agenteOk" accion="nula"
       estadoSiguiente="muestraInfoComponentes" modoDeTransicion="bloqueante"/>
       <transicion input="agenteBad" accion="nula"
       estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
    </estado>
    <estado idIntermedio="creandoComportamiento">
       <transicion input="comportamientoOk" accion="nula"
       estadoSiguiente="muestraInfoComponentes" modoDeTransicion="bloqueante"/>
       <transicion input="comportamientoBad" accion="nula"
       estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
    </estado>
    <estado idIntermedio="tratamientoErrores">
       <transicion input="errorIrrecuperable" accion="terminacion"
       estadoSiguiente="terminado" modoDeTransicion="bloqueante"/>
       <transicion input="errorRecuperable" accion="nula"
       estadoSiguiente="esperaAccionInicial" modoDeTransicion="bloqueante"/>
    </estado>
    <estadoFinal idFinal="terminado"/>
    <transicionUniversal input="termina" accion="terminacion"
    estadoSiguiente="terminado" modoDeTransicion="bloqueante"/>
    <transicionUniversal input="error" accion="clasificaError"
    estadoSiguiente="tratamientoErrores" modoDeTransicion="bloqueante"/>
</tablaEstados>
```

Figura 24 Autómata agente aplicación asistente

Cabe destacar que para no dificultar mucho la apreciación de los autómatas, no se han pintado las dos transiciones universales que se han definido en el fichero XML. La primera de ellas que desde cualquier estado se puede recibir un input de terminación y se terminaría la aplicación. La segunda representa el que en cualquier estado se pueda recibir un input de error lo que haría transitar al autómata a un estado de clasificación de errores para su posterior análisis y tratamiento.

Las 14 acciones que se pueden obtener del autómata y que son los métodos de la

clase de acciones semánticas del agente serían los siguientes:

- *public void arranque()*

  Inicializa los recursos del asistente, obtiene las descripciones que ha definido el usuario y las muestra por pantalla (pantalla inicial de la aplicación).

- *public void crearNuevaDescripcion(String idDescripcion)*

  Dado un identificador de descripción, se crea una nueva descripción de organización en memoria y, posteriormente se obtienen los agentes y los recursos definidos en el código de la infraestructura y se muestran en las pantallas de edición de componentes.

- *public void abrirDescripcion(String idDescripcion)*

  Dado el identificador de la descripción seleccionada para su apertura, el sistema la carga en memoria, obtiene los componentes que vienen definidos en la misma y los muestra por pantalla. Además obtienen los agentes y los recursos definidos en el código de la infraestructura y se muestran en las pantallas de edición de componentes.

- *public void muestraComportamientosAgente(String nombreAgente)*

  Dado un identificador de agente, seleccionado en la pantalla de edición de agentes, se obtienen los comportamientos asociados al mismo y se muestran por pantalla.

- *public void crearNuevaInstanciaAgente(String nombreAgente, String nombreComportamiento, String nombreInstancia)*

  Dado un identificador de agente, un identificador de comportamiento y un identificador de instancia, el asistente crea la instancia en la descripción cargada en memoria y la muestra por pantalla en el apartado de instancias de agentes.

- *public void crearNuevaInstanciaRecurso(String nombreRecurso, String nombreInstancia)*

  Dado un identificador de recurso y un identificador de instancia, el asistente crea

la instancia de recurso en la descripción cargada en memoria y la muestra por pantalla en el apartado de instancias de recursos.

- *public void borrarInstanciaAgente(String nombreAgente, String nombreComportamiento, String nombreInstancia)*

  Dado un identificador de agente, un identificador de comportamiento y un identificador de instancia, el asistente elimina la instancia de la descripción cargada en memoria y la elimina del apartado de instancias de agentes.

- *public void borrarInstanciaRecurso(String nombreRecurso, String nombreInstancia)*

  Dado un identificador de recurso y un identificador de instancia, el asistente elimina la instancia de la descripción cargada en memoria y la elimina del apartado de instancias de recursos.

- *public void crearNuevoAgente(String nombreAgente)*

  Dado un identificador de agente, el asistente genera el código de un nuevo agente y su comportamiento por defecto en el código de la infraestructura y muestra el agente en el apartado de agentes de la pantalla de edición de agentes.

- *public void crearNuevoRecurso(String nombreRecurso)*

  Dado un identificador de recurso, el asistente genera el código de un nuevo recurso y su clase generadora en el código de la infraestructura y muestra el nuevo recurso en el apartado de recursos de la pantalla de edición de recursos.

- *public void crearNuevoComportamiento(String nombreAgente, String nombreComportamiento)*

  Dado un identificador de agente y un identificador de comportamiento, el asistente genera el código de un nuevo comportamiento asociado al agente seleccionado y lo muestra por pantalla en el apartado de comportamientos de agente.

- *public void guardarDescripcion()*

  El asistente guarda en un fichero XML la descripción guardada en memoria, previa comprobación de que la descripción contiene tanto instancias de agentes como instancias de recursos. El nombre del fichero es que se seleccionó en la pantalla inicial.

- *public void clasificaError()*

  Una vez que se ha detectado un error en el sistema, se clasifica y se decide qué acciones se deben de tomar al respecto. Esta es una de las dos operaciones obligatorias en todos los agentes de ICARO-T, así se impone el control de errores en las aplicaciones.

- *public void terminacion()*

  Este método se ejecuta cuando el usuario decide terminar la aplicación. Guarda correctamente las estructuras generadas y reporta al gestor de agentes que el agente en cuestión termina su ejecución y que actúe en consecuencia.

4.1.3. Recursos

El asistente utiliza dos recursos. Un recurso de persistencia, denominado recursoPersistenciaAsistente y un recurso de visualización con identificador recursoVisualizacionAsistente. Como todos los recursos de ICARO-T, los recursos se comportan como componentes independientes que ofrecen un interfaz público. Este interfaz público es utilizado por los agentes.

4.1.3.1. Recurso de persistencia

El recurso de persistencia (Figura 25) es utilizado para tener guardado en memoria una representación abstracta de la información de los componentes que el usuario va seleccionando para su aplicación y de los que tiene disponibles.

El interfaz público de este contiene los siguientes métodos:

- *public String[] obtenerDescripciones()*

  Devuelve una lista con los nombres los ficheros de descripciones que hay en el directorio destinado a contener este tipo de ficheros.

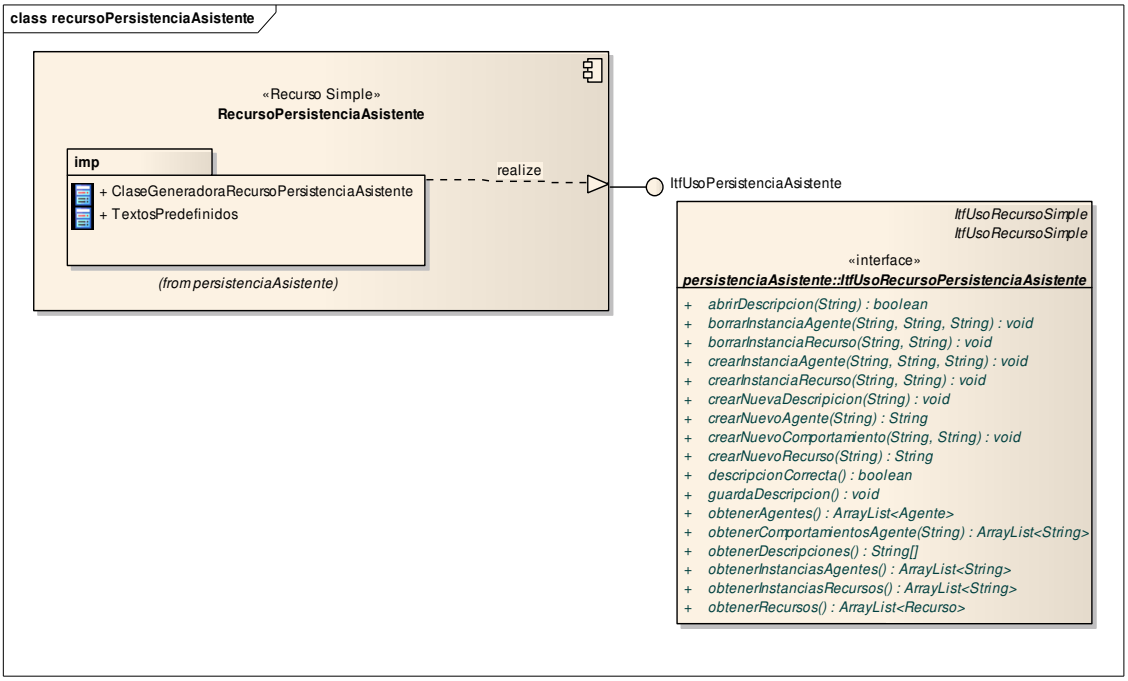


Figura 25 Recurso de persistencia del asistente

- *public void crearNuevaDescripicion(String nombreDescripcion)*

  Crea una representación en memoria de una nueva descripción de aplicación. Crea una descripción vacía salvo por las partes que son comunes a todas las aplicaciones, como las propiedades globales y los gestores.

- *public boolean abrirDescripcion(String nombreDescripcion)*

  Abre y carga en memoria una descripción existente, previa comprobación de que la descripción es una descripción correcta.

- *public ArrayList<Agente> obtenerAgentes()*

  Devuelve una lista con los agentes que hay codificados en el sistema. En la carpeta de los ficheros fuente de la aplicación busca las estructuras que coinciden con la de un agente.

- *public ArrayList<Recurso> obtenerRecursos()*

  Devuelve una lista con los recursos que hay codificados en el sistema. En la carpeta de los ficheros fuente de la aplicación busca las estructuras que coinciden con la de un recurso.

- *public ArrayList<String> obtenerComportamientosAgente(String agente)*

  Dado un agente concreto, el recurso busca, dentro de la estructura de código del agente, comportamientos que el agente tenga definidos.

- *public void crearInstanciaAgente(String nombreAgente, String nombreComportamiento, String nombreInstancia)*

  Dado un identificador de agente, un identificador de comportamiento y un identificador de instancia, el recurso añade a la descripción actual cargada en memoria una instancia del agente correspondiente.

- *public void crearInstanciaRecurso(String nombreRecurso, String nombreInstancia)*

  Dado un identificador de recurso y un identificador de instancia el recurso añade a la descripción actual cargada en memoria una instancia del recurso correspondiente.

- *public void borrarInstanciaAgente(String nombreAgente, String nombreComportamiento, String nombreInstancia)*

  Dado un identificador de agente, un identificador de comportamiento y un identificador de instancia, el recurso elimina de la descripción actual cargada en memoria la instancia del agente correspondiente.

- *public void borrarInstanciaRecurso(String nombreRecurso, String nombreInstancia)*

  Dado un identificador de recurso y un identificador de instancia, el recurso elimina de la descripción actual cargada en memoria la instancia

del recurso correspondiente.

- *public String crearNuevoAgente(String nombreAgente)*

  Dado un identificador de agente, el recurso crea un nuevo agente en el sistema. No sólo crea una representación abstracta de un nuevo agente, sino que crea una estructura de agente completa dentro del código del sistema, es decir, crea la carpeta del agente en el código y la carpeta con el comportamiento por defecto que contiene un fichero XML que representa el autómata y el fichero Java de las acciones semánticas del agente. La única labor del desarrollador es rellenar estos ficheros. Este nuevo agente queda disponible desde el momento de su creación para ser añadido al fichero de descripción.

- *public String crearNuevoRecurso(String nuevoRecurso)*

  Dado un identificador de recurso, el recurso crea un nuevo recurso en el sistema. No sólo crea una representación abstracta de un nuevo recurso, sino que crea una estructura de agente completa dentro del código del sistema, es decir, crea la carpeta del recurso en código, que contiene el fichero Java del interfaz del recurso, y la carpeta de implementación del recurso con el fichero de la clase Java que genera el recurso en el sistema. La única labor del desarrollador es rellenar estos ficheros. Este nuevo recurso queda disponible desde el momento de su creación para ser añadido al fichero de descripción.

- *public void crearNuevoComportamiento(String nombreAgente, String nombreComportamiento)*

  Dado un identificador de agente y un identificador de comportamiento, el recurso crea un nuevo comportamiento en el sistema. Crea además la estructura del recurso completa en el código del sistema, es decir, la carpeta del comportamiento, dentro de la estructura del agente correspondiente, que contiene el fichero XML que representa el autómata del comportamiento y el fichero que contiene la clase Java con las

acciones semánticas correspondientes al nuevo comportamiento. La única labor del desarrollador es rellenar estos ficheros. Este nuevo comportamiento queda disponible desde el momento de su creación para ser añadido al fichero de descripción.

- *public ArrayList<String> obtenerInstanciasAgentes()*

  Devuelve una lista con las instancias de agentes definidas en la descripción actualmente cargada en memoria.

- *public ArrayList<String> obtenerInstanciasRecursos()*

  Devuelve una lista con las instancias de recursos definidas en la descripción actualmente cargada en memoria.

- *public boolean descripcionCorrecta()*

  Devuelve cierto si la descripción actualmente cargada en memoria contiene instancias tanto de agentes como de recursos.

- *public void guardaDescripcion()*

  Escribe en un fichero XML, en caso de que sea correcta, la descripción de la organización que el usuario ha definido con el asistente.

4.1.3.2. Recurso de visualización

El recurso de visualización (Figura 26) publica en su interfaz público los métodos necesarios para generar y controlar la parte gráfica del asistente.

Los métodos que ofrece este componente son los siguientes:

- *public void muestraDescripciones(String[] descripciones)*

  Muestra en la pantalla inicial del asistente los ficheros de descripciones que se han encontrado en el sistema.

- *public void muestraAgentes(ArrayList<Agente> agentes)*

  Muestra en la pantalla de agentes los agentes que se han encontrado en el código del sistema.

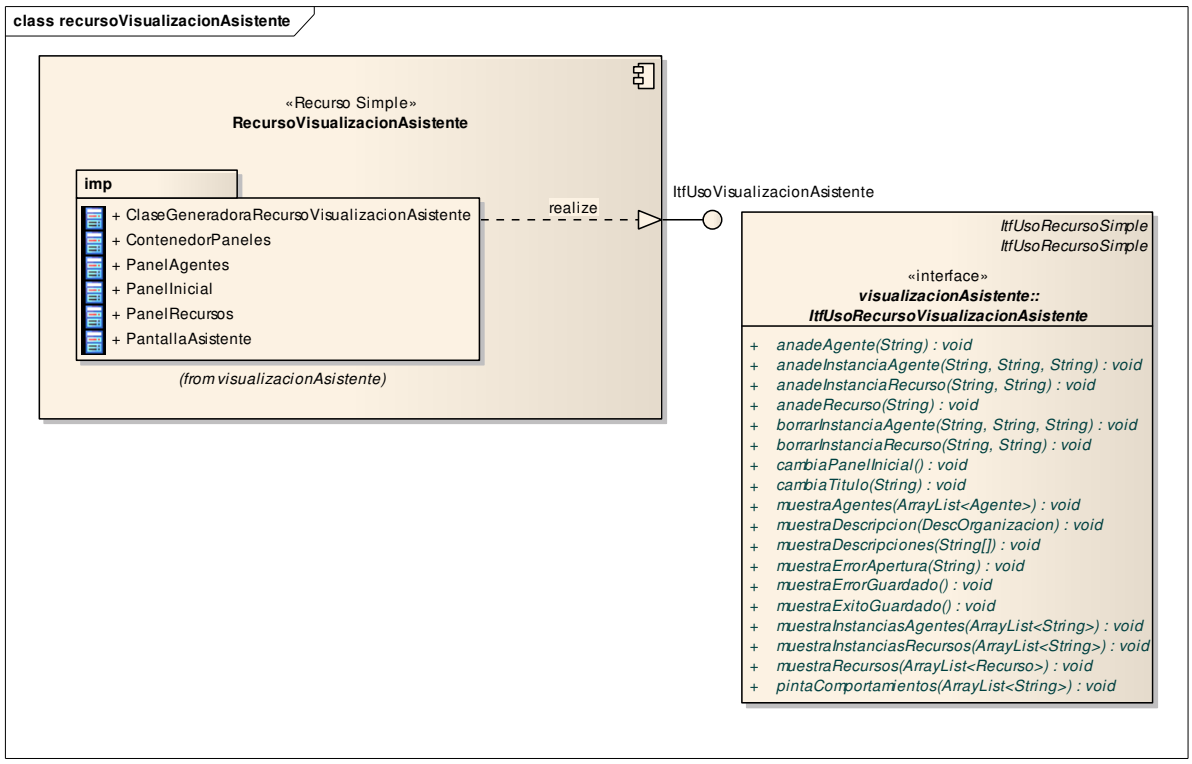


Figura 26 Recurso de visualización del asistente

- *public void cambiaPanelInicial()*

  Cambia la visibilidad de la pantalla inicial a la pantalla de agentes.

- *public void pintaComportamientos(ArrayList<String> comportamientos)*

  Muestra los comportamientos asociados que tiene un agente en la pantalla de agentes.

- *public void anadeInstanciaAgente(String nombreAgente, String nombreComportamiento, String nombreInstancia)*

  Añade una instancia de comportamiento de agente a la lista de instancias que hay en la pantalla de agentes.

- *public void anadeInstanciaRecurso(String nombreRecurso, String nombreInstancia)*

  Añade una instancia de recurso a la lista de instancias que hay en la

pantalla de recursos.

- *public void borrarInstanciaAgente(String nombreAgente, String nombreComportamiento, String nombreInstancia)*

  Elimina una instancia de comportamiento de agente de la lista de instancias que hay en la pantalla de agentes.

- *public void borrarInstanciaRecurso(String nombreRecurso, String nombreInstancia)*

  Elimina una instancia de recurso de la lista de instancias que hay en la pantalla de recursos.

- *public void anadeAgente(String nombreAgente)*

  Añade un nuevo agente a la lista de agentes que hay en la pantalla de agentes.

- *public void anadeRecurso(String nombreRecurso)*

  Añade un nuevo recurso a la lista de recursos que hay en la pantalla de recursos.

- *public void muestraErrorApertura(String idDescripcion)*

  Muestra por pantalla un mensaje de error relacionado con la apertura de un fichero de descripción.

- *public void muestraInstanciasAgentes(ArrayList<String> instancias)*

  Muestra en la lista de instancias de comportamientos de agente las instancias que contiene un fichero de descripción.

- *public void muestraInstanciasRecursos(ArrayList<String> instancias)*

  Muestra en la lista de instancias de recursos las instancias de recursos que contiene un fichero de descripción.

- *public void muestraErrorGuardado()*

  Muestra por pantalla un mensaje de error relacionado con el guardado de

la descripción actual en un fichero.

- *public void muestraExitoGuardado()*

  Muestra por pantalla un mensaje de éxito relacionado con el guardado de la descripción actual en un fichero.

- *public void cambiaTitulo(String titulo)*

  Cambia el título de la ventana principal del asistente.

4.1.4. Clases de Dominio

Las clases de dominio que utiliza el asistente están divididas en dos conjuntos, por un lado están las que comparte con el núcleo de la plataforma que son las que utiliza para crear la representación abstracta de los componentes definidos en un fichero de descripción. Por otro lado están las clases de dominio que se utilizan para la abstracción de los componentes que hay disponibles en la plataforma (una vez analizado el código). Estas últimas son las que se han creado nuevas pues antes, esta abstracción no era necesaria.

Las clases de dominio del asistente son tres:

- AgenteCodigoUsuario

  Esta clase contiene tres atributos, el nombre de la carpeta que contiene al agente, el nombre del agente y la lista de comportamientos que tiene el agente.

- ComportamientoCodigoUsuario

  Esta clase contiene tres atributos, el nombre de la carpeta que contiene el comportamiento y dos atributos de tipo *boolean* que indican si la carpeta contiene un fichero XML y un fichero Java de acciones semánticas.

- RecursoCodigoUsuario

  Esta clase contiene cuatro atributos, el nombre de la carpeta que contiene el recurso, el nombre del recurso y dos atributos de tipo *boolean* que

indican si la carpeta del recurso contiene un fichero Java que es el interfaz de uso y otro que sea la clase generadora del mismo.

4.2. Descripción de organizaciones

Como se ha apuntado antes, las aplicaciones desarrolladas para la plataforma ICARO-T, requieren de un fichero de descripción en el que se detalla el comportamiento estático y el comportamiento dinámico de las mismas. El fichero de descripción está escrito en lenguaje XML cuya estructura y restricciones de contenido están definido por un fichero XSD (*xml Schema*). En las siguientes figuras se muestran diagramas que expresan gráficamente la composición del metamodelo XSD de las descripciones de organizaciones.

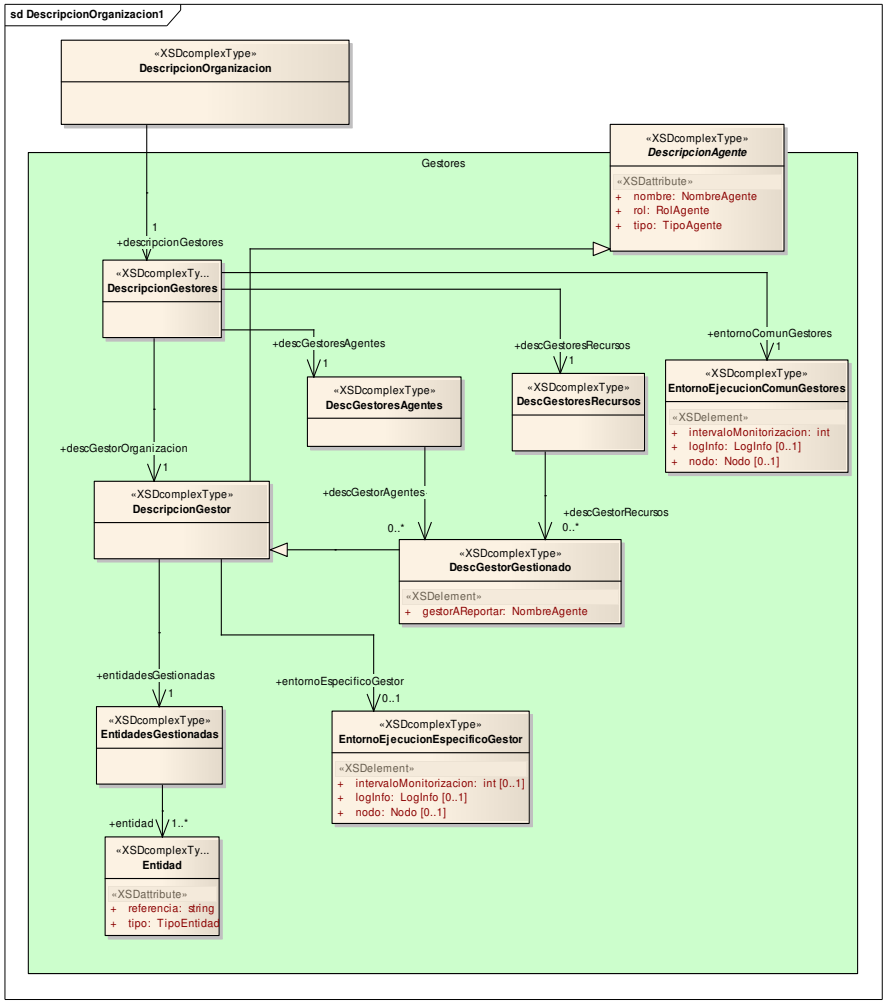


Figura 27 Metamodelo XSD de la organización (I)

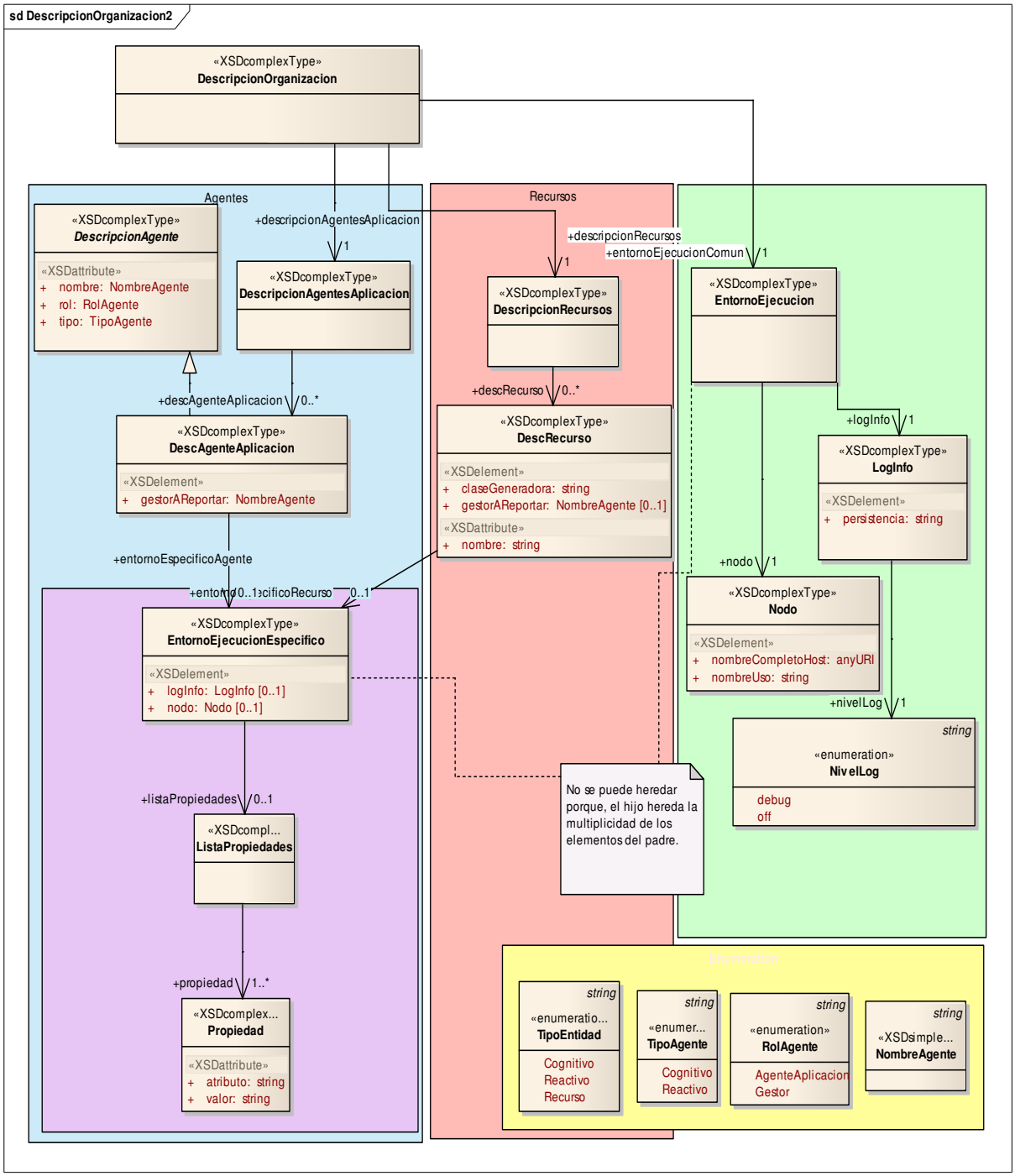


Figura 28 Metamodelo XSD de la organización (II)

A continuación, en la Figura 29 se muestra la estructura del esquema, haciendo énfasis en la composición de cada uno de sus elementos.

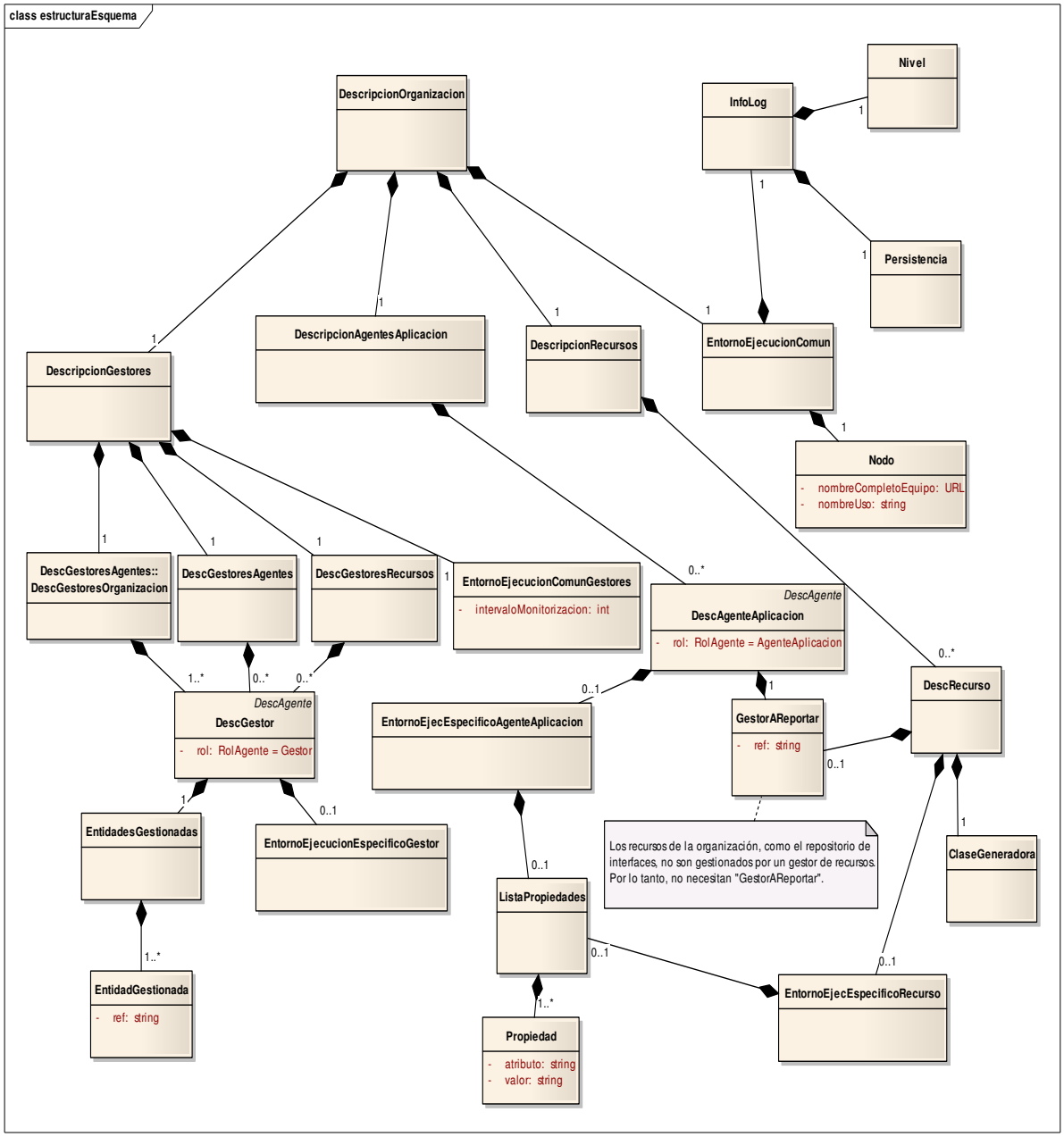


Figura 29 Esquema elementos de la descripción

En la Figura 30 se muestran las relaciones de herencia entre los distintos elementos que componen el sistema.

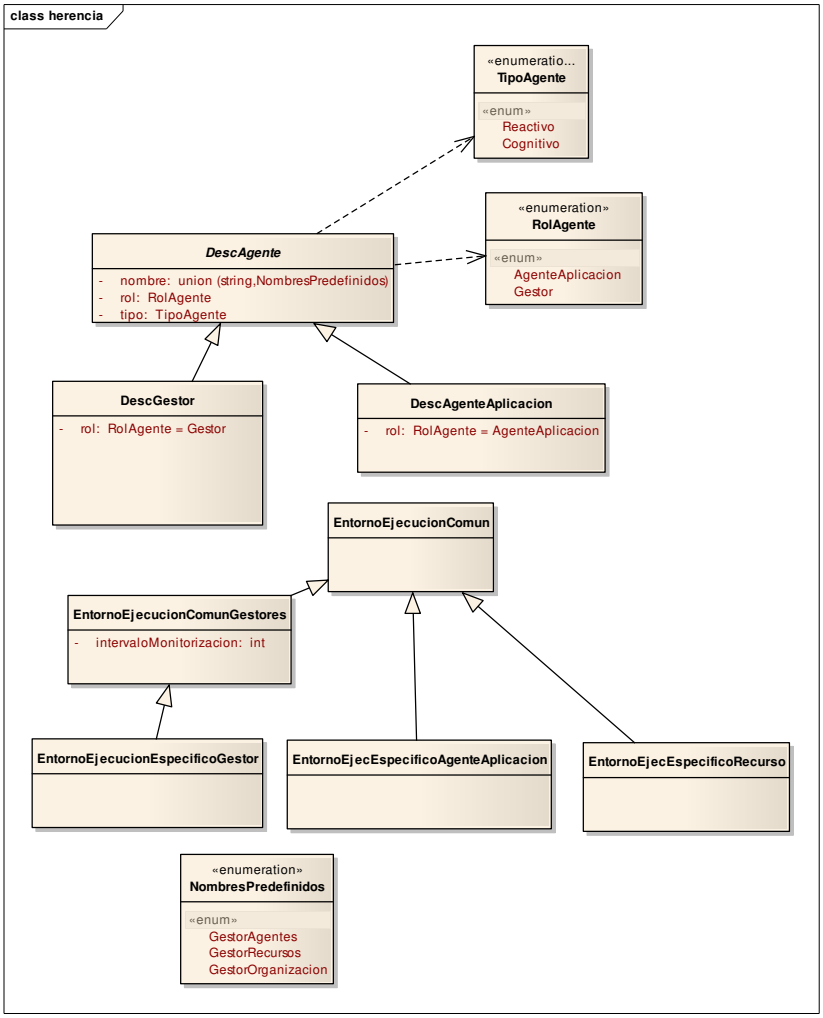


Figura 30 Relaciones de herencia entre los elementos de la descripción

El fichero de descripción se pasa como parámetro en el arranque de la plataforma y es en ese momento cuando es analizado. Primero se comprueba que el fichero no tiene errores estructurales, al tener un fichero XML y un fichero XSD se puede comprobar la correcta construcción del primero. A continuación se analiza su contenido, gracias al cual el control de la plataforma conoce la descripción de los componentes y sabe cuales debe generar, arrancar y gestionar.

La descripción está dividida en cinco partes, cabecera, propiedades globales, parte estática, parte dinámica y parte de despliegue. De estas cinco partes, tanto la cabecera como las propiedades globales son comunes a todas las aplicaciones.

4.2.1. Cabecera

En la parte de cabecera del fichero, se definen elementos relacionados con el espacio de nombrado del fichero XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<icaro:DescOrganizacion
xmlns:icaro="urn:icaro:aplicaciones:descripcionOrganizaciones"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:icaro:aplicaciones:descripcionOrganizacio
nes ../../../../schemas/DescripcionOrganizacionSchema.xsd ">
```

Figura 31 Cabecera fichero descripción

En la Figura 31 se puede observar cómo, en la cabecera, se define el espacio de nombrado, "xmlns:icaro="urn:icaro:aplicaciones:descripcionOrganizaciones"", la utilización de los elementos definidos en *http://www.w3.org/2001/XMLSchema-instance*, "xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"" y la localización del fichero XSD que representa el *xml schema,* "xsi:schemaLocation="urn:icaro:aplicaciones:descripcionOrganizaciones ../../../../schemas/DescripcionOrganizacionSchema.xsd ""

4.2.2. Propiedades Globales

En la parte de propiedades globales se definen propiedades del tipo atributo-valor que son globales a todo el sistema, es decir, parámetros que utiliza la plataforma para su funcionamiento interno.

```
<icaro:PropiedadesGlobales>
   <icaro:intervaloMonitorizacionGestores>
      5000
   </icaro:intervaloMonitorizacionGestores>
   <icaro:activarPanelTrazasDebug>true</icaro:activarPanelTrazasDebu
g>
</icaro:PropiedadesGlobales>
```

Figura 32 Propiedades globales del fichero de descripción

En la Figura 32 se ve como en este caso se han definido dos propiedades globales, el intervalo de monitorización de los gestores, que es el tiempo que debe esperar el control de la plataforma para monitorizar el estado de los componentes, en

este caso 5000 milisegundos y el otro parámetro es la activación del nivel *Debug* de trazas.

4.2.3. Parte estática

En la parte estática del fichero de descripción se definen los componentes que forman la aplicación. Esta parte se subdivide en otras dos partes, la descripción del comportamiento de los agentes y la definición de los recursos.

```
<icaro:DescComportamientoAgentes>
  <icaro:DescComportamientoGestores>
    <icaro:DescComportamientoAgente
      nombreComportamiento="GestorOrganizacion" rol="Gestor"
      tipo="Reactivo" />
    <icaro:DescComportamientoAgente
      nombreComportamiento="GestorAgentes" rol="Gestor"
      tipo="Reactivo" />
    <icaro:DescComportamientoAgente
      nombreComportamiento="GestorRecursos" rol="Gestor"
      tipo="Reactivo" />
  </icaro:DescComportamientoGestores>
  <icaro:DescComportamientoAgentesAplicacion>
    <icaro:DescComportamientoAgente
      nombreComportamiento="AgenteAplicacionAcceso"
      rol="AgenteAplicacion" localizacionComportamiento="..."
      tipo="Reactivo" />
  </icaro:DescComportamientoAgentesAplicacion>
</icaro:DescComportamientoAgentes>
```

Figura 33 Descripción comportamiento agentes

Se denomina estática porque sólo se definen los componentes, no se indica si hay que instanciarlos o no ni el número de veces.

La descripción del comportamiento de los agentes (Figura 33) se divide en dos partes, la parte que describe el comportamiento de los gestores (que también son agentes) y la parte que describe el comportamiento de los agentes de la aplicación. Los gestores de la aplicación son comunes a todas las aplicaciones, por lo que sólo se identifican con el identificador, el rol de gestor y el tipo de agente (los gestores son reactivos). A continuación en la parte destinada a los agentes de la aplicación se describen los agentes que son propios de la aplicación, en este caso el AgenteAplicacionAcceso. Estos agentes se definen de la misma manera que los gestores, identificador, rol (en este caso AgenteAplicacion), tipo y además hay que

indicarle al sistema donde se localiza el comportamiento del agente en el código.

```
<icaro:DescRecursosAplicacion>
   <icaro:DescRecursoAplicacion nombre="Persistencia"
   localizacionClaseGeneradora="..."/>
   <icaro:DescRecursoAplicacion nombre="VisualizacionAcceso"
   localizacionClaseGeneradora="..."/>
</icaro:DescRecursosAplicacion>
```

Figura 34 Descripción recursos aplicación

En la parte de definición de recursos se enumeran los recursos que componen el sistema. Para definir un recurso en la plataforma simplemente hay que identificar el recurso con un nombre e indicarle al control la localización de la clase raíz a partir de la cual se puede instanciar el recurso.

4.2.4. Parte dinámica

En parte dinámica se especifican las instancias de los componentes, definidos en la parte estática. Este apartado se divide en tres partes, las instancias de gestores, las instancias de agentes y las instancias de recursos.

Los gestores son agentes de la plataforma, por lo que en la parte de instanciación sólo hay que indicar los componentes que deben gestionar. En este caso (Figura 35), el gestor de organización gestiona a los otros dos gestores, el de recursos y el de agentes. El gestor de agentes gestiona una instancia del agente AgenteAplicaciónAcceso y al gestor de recursos se le asignan una instancia de cada uno de los recursos que tiene definidos la aplicación, una del recurso de persistencia y otra del recurso de visualización.

```
<icaro:Gestores>
   <icaro:InstanciaGestor id="GestorOrganizacion"
   refDescripcion="GestorOrganizacion">
      <icaro:componentesGestionados>
         <icaro:componenteGestionado refId="GestorAgentes"
         tipoComponente="Gestor"/>
         <icaro:componenteGestionado refId="GestorRecursos"
         tipoComponente="Gestor" />
      </icaro:componentesGestionados>
   </icaro:InstanciaGestor>
   <icaro:InstanciaGestor id="GestorAgentes"
   refDescripcion="GestorAgentes">
      <icaro:componentesGestionados>
         <icaro:componenteGestionado refId="AgenteAplicacionAcceso1"
         tipoComponente="AgenteAplicacion"/>
      </icaro:componentesGestionados>
   </icaro:InstanciaGestor>
   <icaro:InstanciaGestor id="GestorRecursos"
   refDescripcion="GestorRecursos" >
      <icaro:componentesGestionados>
         <icaro:componenteGestionado refId="Persistencia1"
         tipoComponente="RecursoAplicacion"/>
         <icaro:componenteGestionado refId="VisualizacionAcceso1"
         tipoComponente="RecursoAplicacion"/>
      </icaro:componentesGestionados>
   </icaro:InstanciaGestor>
</icaro:Gestores>
```

Figura 35 Instancias de gestores

En el caso de las instancias de los agentes y los recursos de la aplicación, simplemente hay que definir el nombre de la instancia, que es el que se utilizará para recuperarla en el sistema durante la ejecución y el componente, de los definidos en la parte estática, al que se refiere. Además se pueden declarar propiedades del tipo atributo-valor que podrán ser recuperadas por el desarrollador en el código.

En el ejemplo (Figura 36) se puede observar que se define una instancia de agente de aplicación, con identificador AgenteAplicacionAcceso1, sin atributos y dos instancias de recursos, una del recurso de persistencia con identificador Persistencia1 y otra del recurso de visualización con identificador VisualizacionAcceso1. Cabe destacar que en el caso del recurso Persistencia1 como se necesitan datos para acceder a una base de datos se han definido cuatro parámetros relacionados con la misma, usuario, password, script de creación y la url del acceso remoto.

```
<icaro:AgentesAplicacion>
  <icaro:Instancia id="AgenteAplicacionAcceso1"
  refDescripcion="AgenteAplicacionAcceso">
    <icaro:listaPropiedades>
      <icaro:propiedad atributo="" valor=""/>
    </icaro:listaPropiedades>
  </icaro:Instancia>
</icaro:AgentesAplicacion>
<icaro:RecursosAplicacion>
  <icaro:Instancia id="Persistencia1" refDescripcion="Persistencia"
  xsi:type="icaro:Instancia">
    <icaro:listaPropiedades>
      <icaro:propiedad atributo="MYSQL_USER" valor="root" />
      <icaro:propiedad atributo="MYSQL_PASSWORD" valor="root" />
      <icaro:propiedad atributo="MYSQL_SCRIPT_ITEMS"
      valor="config/bbdd.SQL" />
      <icaro:propiedad atributo="MYSQL_URL"
      valor="jdbc:mysql://localhost:3306/bbddejemplo" />
    </icaro:listaPropiedades>
  </icaro:Instancia>
  <icaro:Instancia id="VisualizacionAcceso1"
  refDescripcion="VisualizacionAcceso" xsi:type="icaro:Instancia"/>
</icaro:RecursosAplicacion>
```

Figura 36 Instancias de agentes y recursos

4.2.5. Despliegue

En la parte de despliegue se identifican los nodos en los que deben desplegarse los ejemplares, para que se puedan localizar de manera distribuida los recursos que residen en la plataforma.

```
<icaro:nodoComun>
   <icaro:nombreUso>NodoPrincipal</icaro:nombreUso>
   <icaro:nombreCompletoHost>localhost</icaro:nombreCompletoHost>
</icaro:nodoComun>
```

Figura 37 Despliegue de los componentes

En este caso se indica que la plataforma se despliega en el nodo principal que corresponde a la dirección *localhost.*

4.3. Ejemplo de Funcionamiento

El asistente funciona mediante una interfaz de usuario, que permite realizar diferentes operaciones sobre los componentes de la plataforma ICARO-T. Para esta muestra de funcionamiento se supondrá que el asistente ha sido arrancado

explícitamente por el desarrollador.

4.3.1.1. Gestión de descripciones

Una vez arrancada la plataforma lo primero que se le presenta al desarrollador es la pantalla inicial (Figura 38). En esta pantalla se pueden realizar tres acciones, salir del asistente, crear una nueva descripción de organización o abrir una ya existente, cada una de ellas asociadas a un botón del interfaz.

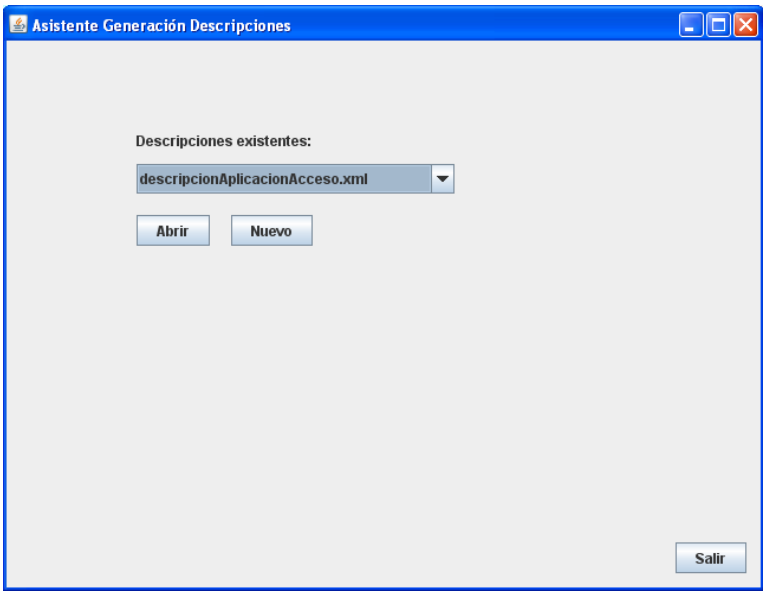


Figura 38 Pantalla inicial

Para abrir una descripción de organización existente, primero se debe de escoger el fichero a abrir del desplegable. Si se opta por crear una nueva descripción de organización el sistema pedirá un nombre para la nueva organización (Figura 39).

Figura 39 Pantalla nueva descripción

Tanto si se abre una descripción de organización existente como si se decide crear una nueva, se continúa hacia la parte del asistente de gestión de componentes. La única diferencia entre los dos casos anteriores es que si se ha optado por abrir una descripción de organización existente, los datos de esa descripción aparecerán en pantalla, más concretamente las instancias de componentes que haya en esa descripción. En el caso de una nueva descripción la parte de instancias aparecerá vacía. En este caso se supondrá que se va a crear una nueva descripción.

La pantalla de gestión de componentes está organizada en dos partes, agentes y recursos, divididas por pestañas.

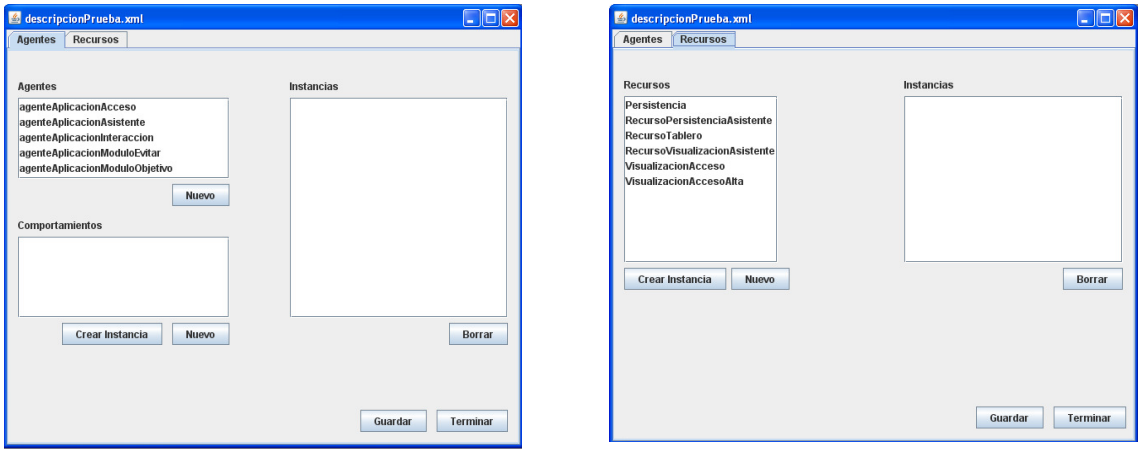


Figura 40 Pantalla gestión de componentes

4.3.1.2. Gestión de Agentes

En la pantalla de agentes se distinguen tres partes, en la parte superior izquierda se sitúa la lista de agentes que se han detectado en el código. Debajo de esta parte está el apartado de comportamientos de los agentes y por último en la parte derecha está el apartado de instancias en el que se irán guardando las instancias que se vayan creando. Si se selecciona un agente, automáticamente aparecen en el apartado de comportamientos los comportamientos de ese agente. Por ejemplo si en este caso se selecciona el "agenteAplicacionAcceso" aparecerán los dos comportamientos que hay codificados para ese agente, "comportamiento" y "comportamientoAlta" (Figura 41).

Una vez que se ha seleccionado el agente y el comportamiento deseado se puede crear una instancia del agente con ese comportamiento asociado. Para crear una instancia se selecciona el botón "Crear Instancia" y el asistente preguntará por un identificador para la nueva instancia. Tras escribir el nombre y aceptar el asistente creará la instancia, la guardará en memoria y la presentará en el apartado de instancias como una línea de texto con tres campos, el nombre del agente, el nombre del comportamiento y el identificador de la instancia. En este caso se va a crear una instancia del "agenteAplicacionAcceso" con el comportamiento "comportamiento" asociado y con identificador "AgenteAplicacionAcceso1" (Figura 42).

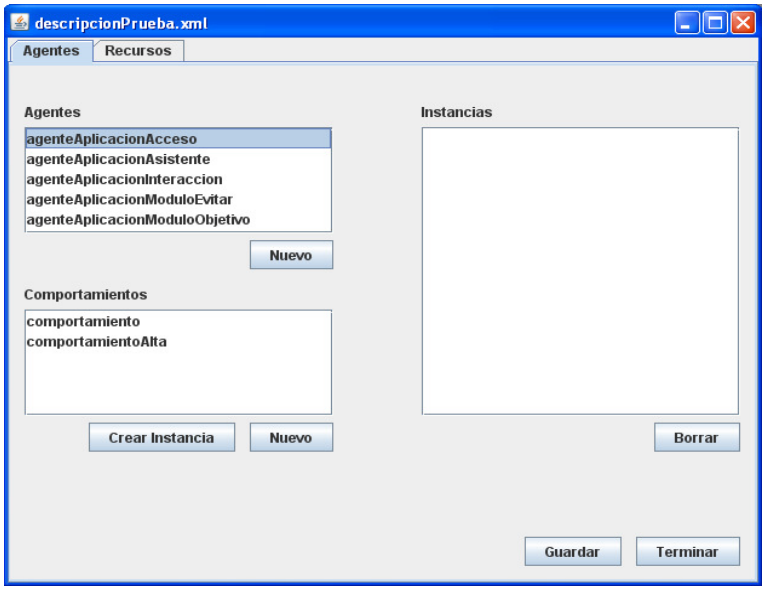


Figura 41 Pantalla agente seleccionado

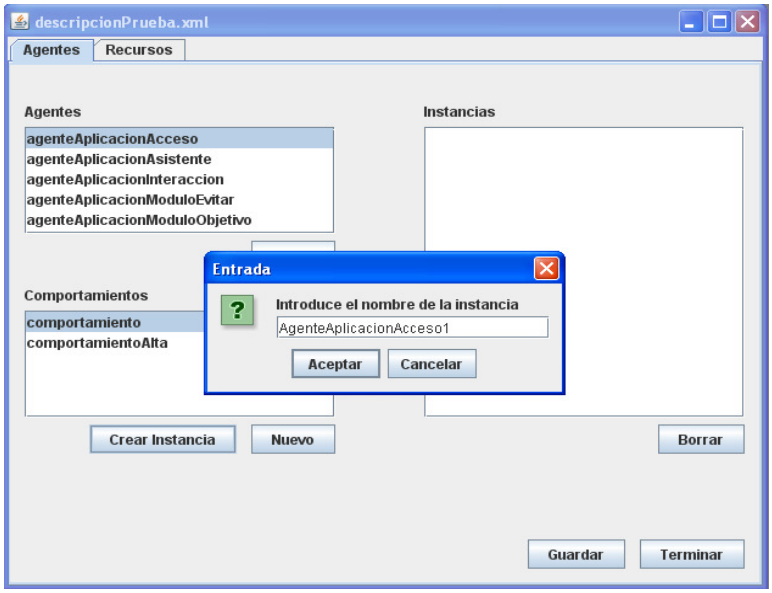


Figura 42 Pantalla creación instancia agente

Para borrar una instancia creada, basta con marcarla y presionar el botón "Borrar", el asistente pedirá la confirmación de la acción y, una vez confirmada, la instancia se borrará de la descripción.

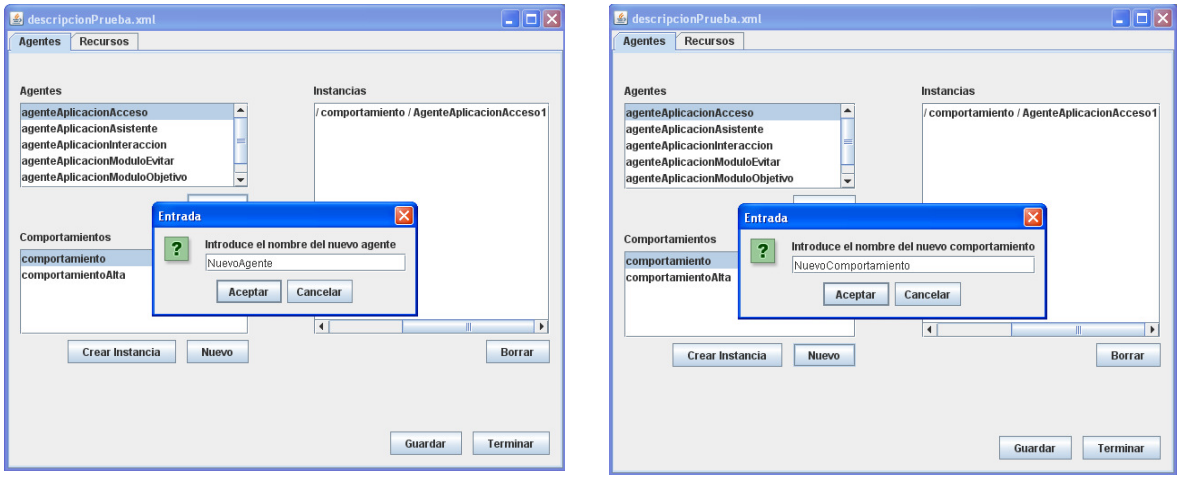


Figura 43 Creación de agentes y comportamientos

En la pantalla de agentes se pueden crear nuevos agentes y nuevos comportamientos (Figura 43). Para crear un nuevo agente hay que pulsar el botón "Nuevo" situado justo debajo del apartado de agentes, a continuación se pregunta por un identificador para el nuevo agente y posteriormente se crea el agente en el código del sistema. De manera similar una vez que se ha seleccionado un agente se puede crear un

nuevo comportamiento presionando el botón "Nuevo" del apartado de comportamientos. Al ser pulsado se pregunta por un identificador para el nuevo comportamiento y por último se crea el comportamiento y se asocia al agente seleccionado.

4.3.1.3. Gestión de Recursos

En la pantalla de recursos (Figura 40), se distinguen dos partes. En la parte izquierda se sitúa la parte destinada a presentar los recursos y en la parte derecha se muestran las instancias de estos recursos. Para crear una instancia de un recurso simplemente hay que seleccionar el recurso del que se desea crear la instancia y pulsar el botón "CrearInstancia", el asistente preguntará entonces por un identificador para la nueva instancia y una vez aceptada la creación, la nueva instancia aparecerá en la parte destinada a mostrar las instancias como una línea de texto con dos campos, el identificador del recurso y el identificador de la instancia (Figura 44). Para borrar una instancia de recurso creada sólo hay que pulsar el botón "Borrar" del apartado de instancias y el asistente pedirá la confirmación de la acción y, una vez confirmada, se eliminará la instancia de la descripción. En este caso se va a crear una instancia del recurso "VisualizacionAcceso" y se identificará como "VisualizaciónAcceso1".

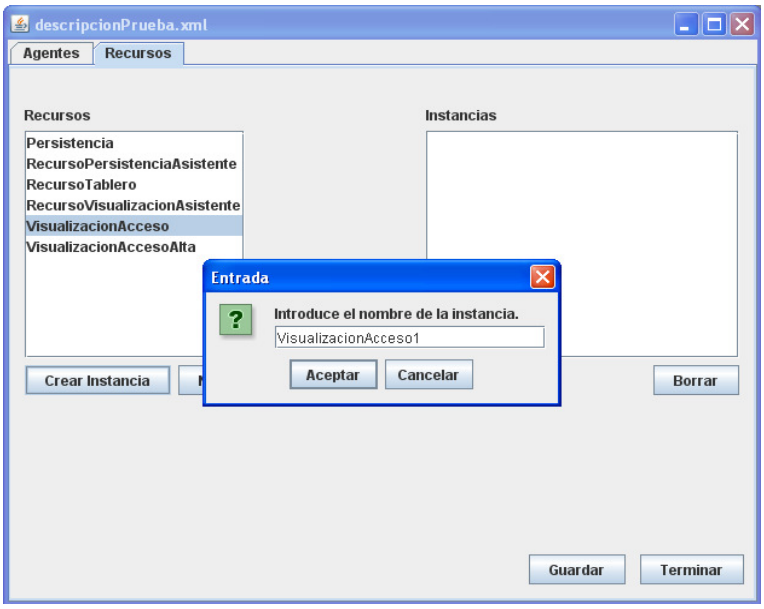


Figura 44 Creación instancia recurso

En la pantalla de recursos también se pueden crear nuevos recursos. Para la creación de un nuevo recurso hay que presionar el botón "Nuevo" de la parte destinada a presentar los recursos. A continuación se pregunta por un identificador para el nuevo recurso y, una vez dado, se crea el nuevo recurso en el código del sistema. En este ejemplo se va a crear un nuevo recurso con identificador "NuevoRecurso" (Figura 45).

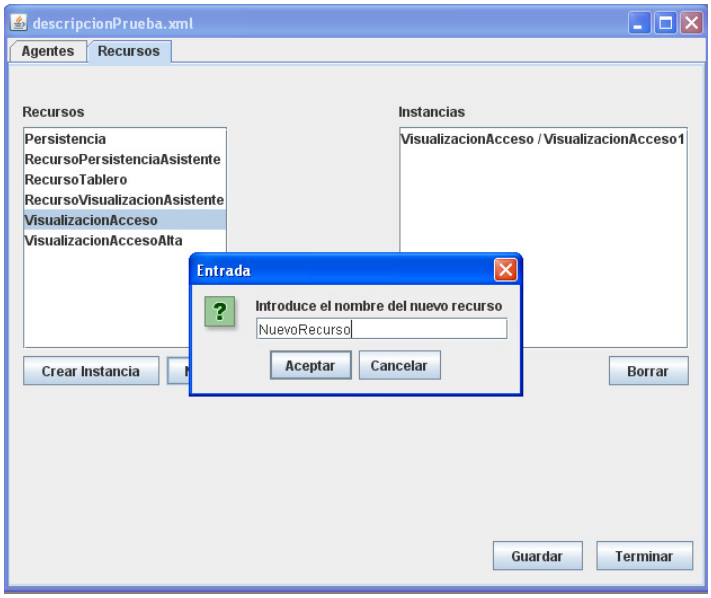


Figura 45 Creación nuevo recurso

4.3.1.4. Guardado y gestión de errores

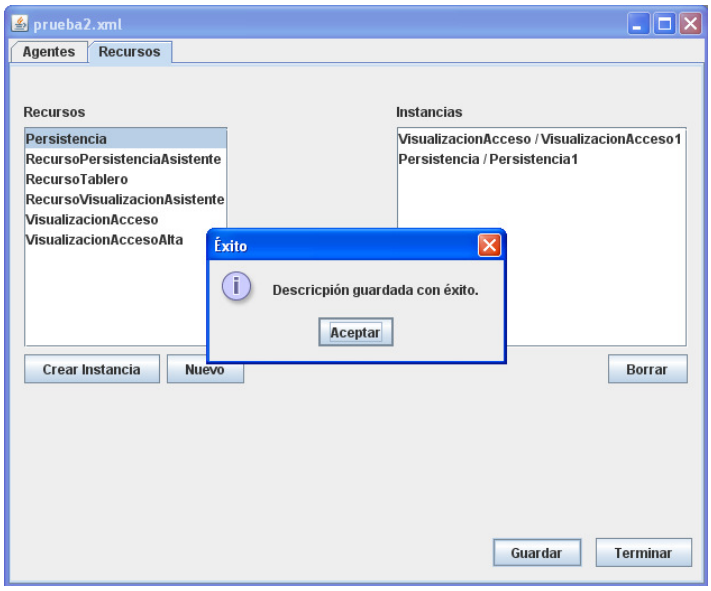


Figura 46 Descripción guardada

Por último una vez que se ha terminado de editar la descripción de organización, se puede guardar en un archivo XML. El fichero en el que se guarda es el mismo que se había cargado en el caso de haber abierto una descripción existente o en un fichero nuevo en el caso de haber creado una descripción nueva. Para guardar la descripción hay que presionar el botón guardar y, si la descripción se guarda correctamente, se mostrará un mensaje de éxito por pantalla (Figura 46).

En caso de que hubiese habido algún problema en el guardado, el sistema mostraría por pantalla un mensaje de error explicando el error. Por ejemplo generaría un mensaje de error si se intentase guardar una descripción que no tuviese instancias de recursos o de agentes.

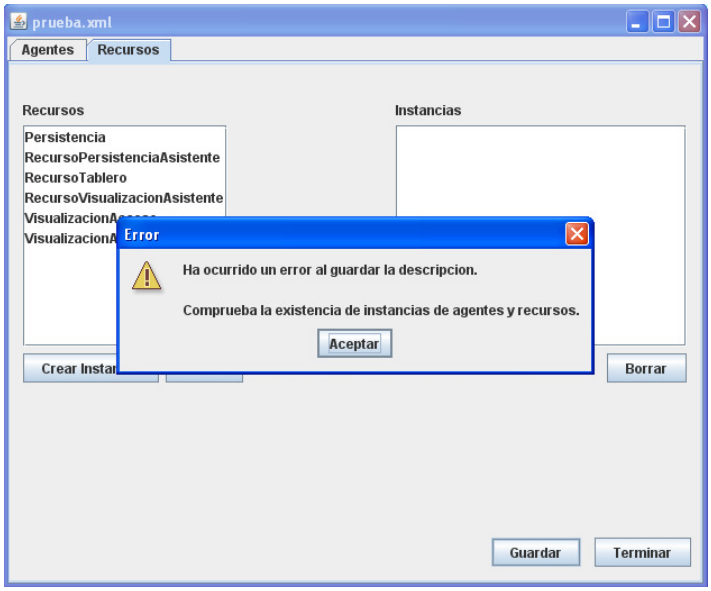


Figura 47 Error en el guardado de la descripción

5. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se recogen las conclusiones del trabajo realizado y las líneas futuras de desarrollo e investigación relacionadas con el mismo.

5.1. Conclusiones

Como resultado de este trabajo se ha diseñado y desarrollado un asistente que permite crear y modificar descripciones de organizaciones de la plataforma ICARO-T.

El concepto de organización aplicado a las plataformas de sistemas multiagente aporta una nueva dimensión a estos sistemas facilitando, sobre todo, la coordinación de los agentes con la asignación de roles y capacidades. Las organizaciones de agentes están basadas en gran medida en las organizaciones humanas, por lo que se han analizado las características tanto de las humanas como las de agentes, obteniendo las características más relevantes de cada una de ellas.

La aplicación del concepto de organización en los sistemas multiagente actuales presenta dos problemas fundamentalmente. Por un lado están las plataformas que directamente no aplican el concepto de organización, por lo que es el desarrollador el que tiene que implementar los mecanismos necesarios manualmente. Por otro lado está el caso de las plataformas que sí aplican el concepto de organización, pero carecen de la robustez arquitectónica necesaria para su aplicación en entornos industriales, en los que la escalabilidad y la fiabilidad suponen un factor determinante. En el análisis de plataformas realizado se han podido ver ejemplos de estos casos.

La plataforma ICARO-T, sobre la que se basa este trabajo, supone un modesto acercamiento a la adaptación del concepto de organización a un sistema multiagente que, en este caso, sí posee la capacidad arquitectónica para ser utilizado en entornos reales. Evidentemente, no todas las características de las organizaciones están representadas en la plataforma dado que se está hablando de elementos computacionales, pero se pueden ver los beneficios aportados por el uso de las mismas, como la flexibilidad o el control de dependencias y de roles.

El asistente ha sido creado con dos finalidades principales, por un lado ayuda al desarrollador a crear, con total fiabilidad, los ficheros de descripción de las organizaciones que desarrolla en la plataforma. Por otro lado, dado que el asistente permite crear nuevos componentes en el código de la plataforma, se puede utilizar para crear, además de la descripción, la estructura básica de una aplicación de ICARO-T, dejando al desarrollador sólo con la tarea de rellenar esa estructura, sin tener que preocuparse de restricciones estructurales o relacionadas con la nomenclatura de los componentes, por ejemplo.

Extender el uso de este asistente para otras plataformas sería posible, pero habría que implementar además una capa con las facilidades que proporciona ICAROT. Esto implicaría prácticamente una reimplementación de ICARO-T sobre los mecanismos de comunicación que ofrecen esas plataformas (Jade, Jack).

5.2. Trabajos Futuros

El asistente ha sido desarrollado como una aplicación de ICARO T utilizando los componentes de la infraestructura, esto implica que puede modificarse o extender la funcionalidad sin mayores problemas. Concretamente, se podría mejorar la ayuda al usuario, explicando mejor los pasos. También puede mejorar la interacción con el usuario dando más ayuda para corregir errores o para continuar la definición de los componentes de la organización. Actualmente si se carga en el sistema una descripción errónea el sistema lo detecta y sólo se lo comunica al usuario.

A partir del desarrollo de este asistente, se podrían ampliar las facilidades para la gestión de organizaciones en ICARO-T. Por ejemplo, sería muy útil poder definir en el asistente despliegues distribuidos, diferenciando los nodos y las características de los mismos. Otro ejemplo de ampliación sería el ampliar el funcionamiento del asistente en tiempo real, permitiendo modificar la organización en tiempo de ejecución. También resultaría interesante permitir al asistente realizar una prueba de ejecución con los componentes seleccionados, es decir, comprobar que las interacciones que se realizan entre estos son correctas y, por lo tanto, ayudan a alcanzar el objetivo del sistema.

Además de las mejoras que se podrían realizar en el asistente, también existen

mejoras relacionadas con la descripción de organizaciones en la plataforma. Por ejemplo se podría ampliar la descripción especificando las interacciones entre los componentes, es decir, en el caso de un agente identificar los recursos de los que requiere funcionalidad, o en el caso de un recurso, identificar los agentes a los que envían eventos.

6. BIBLIOGRAFÍA

[Bellifemine et al. 2005] Bellifemine, F.; Poggi, A.; y Rimassa, G. *Jade: A fipa-compliant agent framework*. In Proc. of PAAM, 97–108. 1999.

[Boissier et al. 2007] O. Boissier, J.F. Hubner, J. S. *Organization oriented programming, from closed to open organizations*. In Proc. Engineering Societies in the Agents World VI, Sixth International Workshop, ESAW06, Lecture Notes in Computer Science. Dublin, Ireland: Springer. 2007.

[Caire et al. 2004] Caire, G., Rimassa, G., Bellifemine, F. *JADE: a versatile run-time for distributed applications on mobile terminals and networks*. SMC (2) 2004: 1882-188

[Dignum et al. 2007] Dignum, F., Rogier M. van Eijk: *Agent communication and social concepts. Autonomous Agents and Multi-Agent Systems* 14(2): 119-120 (2007)

[Dignum et al. 2006] Dignum, V., y Dignum, F. *A landscape of agent systems for the real world*. Technical Report 44-CS-2006-061, Institute of INformation and Computing Sciences, Utrecht University. 2006.

[EIDE] EIDE (2005). *Electronic Institutions Development Environment* (EIDE). http://einstitutor.iiia.csic.es/islander/pub/ (última consulta el 8 sep 2009)

[Esteva et al, 2008] Esteva, M., Rodríguez-Aguilar, J.A., Arcos, J.L., Sierra, C., Noriega, P., Rosell, B., de la Cruz, D.: *Electronic institutions development environment. AAMAS (Demos)* 2008: 1657-1658

[Esteva et al. 2004] Esteva, M., de la Cruz, D., Bruno Rosell, Arcos, J.L., Rodríguez-Aguilar, J.A., Cuní, G.: *Engineering Open Multi-Agent Systems as Electronic Institutions*. AAAI 2004: 1010-1011

[Ferber et al. 2003] Ferber, J.; Gutkenecht, O.; y Michel, F. *From agents to organizations : an organizational view of multi-agent systems*. In Proc. AAMAS03 -

Agent-Oriented Software Engineering Workshop (AOSE). 2003.

[FIPA] FIPA (2002). *FIPA Abstract Architecture Specification*. http://www.fipa.org/specs/fipa00001 (última consulta el 8 sep 2009)

[Fuentes et al. 2004] Fuentes, R., Gómez-Sanz, J., Pavón, J., *A Sociological Framework for Multi-agent Systems Validation and Verification,* Conceptual Modelling, 1st International Workshop on COnceptual MOdeling for Agents (CoMoA 2004), in the ER-2004, vol. 3289, Shanghai, China, Springer-Verlag, pp. 458-469, 08/11/2004

[Garijo et al. 2008] Garijo, F., Polo, F., Spina, D., Rodríguez C., *Manual de Uso de los componentes de ICARO-T.* 2008.

[Garijo et al. 04] Garijo, F J., Bravo S., Gonzalez, J. Bobadilla E. (2004) *BOGAR_LN: An Agent Based Component Framework for Developing Multi-modal Services using Natural Language*. Lecture Notes in Artificial Intelligence, Vol 3040. pp 207-220 Conejo Urretavizcaya Perez de la cruz Eds. Springer-Verlag.2004.

[Garijo et al. 2001] Garijo, F., Gomez-Sanz, J., Pavon, J., Massonet, P. *Multi-Agent System Organization. An Engineering Perspective*. Proceedings MAAMAW 2001, 1-15.

[Gascueña et al. 2008] Gascueña, J.M., Fernández, A., Garijo, F., *Programming reactive agent-based mobile robots using ICARO-T framework.* 2008.

[Gasser. 2001] Gasser, L. *Perspectives on organizations in multi-agent systems*. In Luck, M.; Marik, V.; Stepankova, O.; y Trappl, R., eds., *Multi-agent Systems and Applications*. 9th ECCAI Advanced Course, EASSS 2001, Lecture Notes in Computer Science, 1–16. Springer. 2001.

[Gómez-Sanz et al. 2006] Gómez-Sanz, J., Pavón, J. *Implementing Multi-agent Systems Organizations with INGENIAS*. In: Programming Multi-Agent Systems: Third International Workshop, ProMAS 2005, Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, Amal ElFallah Seghrouchni (Eds.), LNCS 3862, Springer-Verlag 236 - 251.

[Guzmán. 1983] Guzmán, I. Reflexiones en torno al Orden Social. México: Jus. 1983.

[Hodge et al. 2003] Hodge, B. J.; Anthony, W.; y Gales, L. Teoría de la Organización: un enfoque estratégico. Pearson Educación. 2003.

[Howden et al. 2001] Howden, N.; Ronnquist, R.; Hodgson, A.; y Lucas, A. *Jack intelligent agents: Summary of an agent infrastructure*. In Proc. 5th Int. Conference on Autonomous Agents. 2001.

[Hubner et al. 2006] Hubner, J.; Sichman, J.; y Boissier, O. *S-moise+: A middleware for developing organised multi-agent systems*. In Proc. Int. Workshop on Organizations in Multi-Agent Systems, from Organizations to Organization Oriented Programming in MAS, volumen 3913 of Lecture Notes in Computer Science, 64–78. Springer. 2006.

[ICARO] ICARO Project Web Site. http://icaro.morfeo-project.org/ (última consulta el 5 sep 2009).

[Ishida et al. 2004] Ishida, T., Gasser, L., Nakashima, H.: *Massively Multi-Agent Systems I*, First International Workshop, MMAS 2004, Kyoto, Japan, December 10-11, 2004, Revised Selected and Invited Papers Springer 2005

[Jack 2009] JACK Project website. http://www.agent-software.com.au/products/jack/ (última consulta el 1 sep. 2009)

[JADE] JADE Home. http://jade.tilab.com/ (última consulta el 20 ago. 2009)

[Kitio et al. 2007] Kitio, R., Boissier, O., Hübner, J.F., Ricci, A.: Organisational Artifacts and Agents for Open Multi-Agent Organisations: "Giving the Power Back to the Agents". COIN 2007: 171-186

[MADKIT] MADKIT Home. http://www.madkit.org/ (última consulta el 28 ago. 2009)

[Pavón et al. 2008] Pavón, J., Garijo, F., Gómez-Sanz, J. *Complex Systems and Agent-Oriented Software Engineering*. 2008.

[Pavón et al 2007] Juan Pavón, Francisco Garijo, and Jorge J. Gómez-Sanz *Complex Systems and Agent-Oriented Software Engineering in Engineering Environment-Mediated Multi-Agent Systems*. Lecture Notes in Computer Science Publisher Springer Berlin / Heidelberg ISBN 978-3-540-85028-1Vol 5049/2008 Pages 3-16

[Pavón et al. 2005] Pavón, J., Gómez-Sanz, J.J., Fuentes, R. *The INGENIAS Methodology and Tools.* In: Agent Oriented Methodologies. B. Henderson-Sellers and P.Giorgini (Eds.) IDEA Group Publishing (2005) 236-276.

[Pavón et al. 2003] Pavón, J., Gómez-Sanz, J. *Agent Oriented Software Engineering with INGENIAS*. In: Proc. 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003), V. Marik, J. Müller, M. Pechoucek (Eds.), Multi-Agent Systems and Applications II, LNAI 2691, Springer-Verlag (2003) 394-403.

[Peiro. 1991] Peiro, J. *Psicología de la organización.* Universidad Nacional de Educacion a Distancia. Ed. Toran, S.A 1991.

[RAE] RAE. Real Academia Española de la Lengua. http://www.rae.es/rae.html (última consulta el 8 sep. 2009)

[Rao et al. 1991] Rao, A., Georgeff, M. *Modeling Rational Agents within a BDI Architecture. Proceedings of the 2nd international conference on Principles of Knowledge Representation and Reasoning* (1991) 473-484.

[Rodríguez-Aguilar et al. 1998] Rodríguez-Aguilar, J.A., Martín, F.J., Noriega, P., Garcia, P., Sierra, C.: *Towards a Test-Bed for Trading Agents in Electronic Auction Markets*. AI Commun. 11(1): 5-19 (1998)

[Zambonelli et al. 2003] Zambonelli, F., Jennings, N. R., Wooldridge, M. *Developing multiagent systems: The gaia methodology. ACM Transactions on Software Engineering and Methodology* 12:317–370. 2003