

ICARO PATRÓN REACTIVO

MANUAL DE USO DE LOS COMPONENTES

Autores	Francisco J Garijo, Damiano Spina
----------------	-----------------------------------

Distribución:	Público
---------------	---------

Estatus Documento :	En desarrollo
---------------------	---------------

Fecha Elaboración :	10 de Marzo de 2011
---------------------	---------------------

Versión:	V1.5
----------	------

Version	Fecha	Autor	Descripción de Cambios
0.1	14/10/08	F Garijo	Producción de una versión reducida con un solo patrón de agente reactivo
1.0	10/02/2009	F Garijo	Revisión de los apartados y nueva redacción consistente con el código. Sin revisar el apartado que define el meta-modelo de la organización.
1.3	10/11/11	F Garijo	Nueva revisión de los apartado e inclusión de las modificaciones contrastadas en el código
1.4	18/11/2012	F Garijo	Actualización descripción a la infraestructura IcaroV0
1.5	21/01/2014	F Garijo	Actualización descripción de la infraestructura IcaroV0_2 instalada en GitHub

INDICE

1 INTRODUCCIÓN.....	5
2 ORGANIZACIÓN Y CONTENIDO Y DE LA INFRAESTRUCTURA.....	6
3 INSTALACIÓN Y ARRANQUE DE ICARO-T.....	9
3.1 REQUISITOS DE INSTALACIÓN	9
3.2 PROCESO DE INSTALACIÓN EN UN PC CON WINDOWS	9
3.3 PRUEBA DE FUNCIONAMIENTO DE LA INFRAESTRUCTURA	9
4 INTRODUCCIÓN AL FUNCIONAMIENTO DE LOS COMPONENTES DE ICARO: ESPECIFICACIÓN E IMPLEMENTACIÓN DEL SISTEMA DE ACCESO.....	12
4.1 IMPLEMENTACIÓN DEL SISTEMA DE ACCESO CON ICARO.	12
4.2 DEFINICIÓN DE LA ORGANIZACIÓN QUE IMPLEMENTA EL SISTEMA DE ACCESO	35
4.3 DISTRIBUCIÓN DE AGENTES Y RECURSOS EN UNA RED DE PROCESADORES	38
4.4 SISTEMA DE TRAZAS DE LA ORGANIZACIÓN.	42
4.5 PROBLEMAS Y SOLUCIONES EN LA UTILIZACIÓN DE ICARO	46
4.6 EL CICLO DE VIDA DE LOS GESTORES	53
4.7 IMPLEMENTACIÓN DE LOS CASOS DE USO DE LA APLICACIÓN	55
5 DESARROLLO DE APLICACIONES CON ICARO-T.....	57
5.1 IDEAS BÁSICAS	57
5.2 NORMAS DE NOMBRADO	57
5.3 EJEMPLOS	58
5.4 UN POCO MÁS DE DETALLE SOBRE LOS PATRONES	63
6 REFERENCIAS (INCOMPLETAS).....	75

TABLA DE FIGURAS

FIGURA 2-1 ESTRUCTURA DE DIRECTORIOS.....	6
FIGURA 2-2 ESTRUCTURA DE PAQUETES DEL CÓDIGO FUENTE.....	7
FIGURA 2-3 RELACIONES DE DEPENDENCIA ENTRE PAQUETES.....	8
FIGURA 3-4 VENTANA DE ACCESO.....	10
FIGURA 3-5 VENTANA DEL RECURSO DE TRAZAS.....	10
FIGURA 3-6 ESCENARIO DE INTERACCIÓN USUARIO-SISTEMA DE ACCESO.....	12
FIGURA 3-7 COMPONENTES DEL SISTEMA DE ACCESO.....	14
FIGURA 3-8 COMPORTAMIENTO DE LOS COMPONENTES DEL SISTEMA DE ACCESO.....	15
FIGURA 3-9 AUTÓMATA DEL AGENTE DE ACCESO.....	16
FIGURA 3-10 EJEMPLO DEFINICIÓN DEL AUTÓMATA DEL AGENTE DE ACCESO EN XML.....	17
FIGURA 3-11 ESQUEMA PARA MODELAR AUTÓMATAS.....	18
FIGURA 3-12 MODELO DEL AUTÓMATA CON UN DTD.....	19
FIGURA 3-13 ACCIONES SEMÁNTICAS DEL AGENTE REACTIVO DE ACCESO.....	22
FIGURA 3-14 ESTRUCTURA DE CARPETAS DE LOS RECURSOS DE LAS APLICACIONES.....	24
FIGURA 3-15 VISTA ESTÁTICA DE LOS RECURSOS DEL SISTEMA DE ACCESO.....	24
FIGURA 3-16 TAXONOMÍA DE EVENTOS Y MENSAJES.....	26
FIGURA 3-17 COMUNICACIÓN ENTRE EMISORES Y RECEPTORES DE EVENTOS.....	28
FIGURA 3-18 ENVÍO DE EVENTOS ENTRE ENTIDADES.....	30
FIGURA 3-19 ENVÍO Y PROCESAMIENTO DE UN EVENTO POR EL AGENTE REACTIVO.....	32
FIGURA 3-20 DESCRIPCIÓN DE LOS COMPONENTES DE LA ORGANIZACIÓN.....	35
FIGURA 3-21 DESCRIPCIÓN DEL MODELO DINÁMICO DE LA ORGANIZACIÓN.....	37
FIGURA 3-22 DIAGRAMA DE DESPLIEGUE DE LA APLICACIÓN DE ACCESO.....	41
FIGURA 3-23 DEFINICIÓN DEL FICHERO DE LA NUEVA DESCRIPCIÓN DE LA ORGANIZACIÓN-1....	42
FIGURA 3-24 DEFINICIÓN DEL FICHERO DE LA NUEVA DESCRIPCIÓN DE LA ORGANIZACIÓN-2....	42
FIGURA 3-25 RECURSO DE TRAZAS Y LA CLASE INFOTRAZA.....	44
FIGURA 3-26 VENTANAS MOSTRADAS AL EJECUTAR LA APLICACIÓN DE ACCESO.....	45
FIGURA 3-27 ARRANQUE DE LAS APLICACIONES.....	47
FIGURA 3-28 ERRORES EN LA INTERPRETACIÓN DEL AUTÓMATA DEL INICIADOR.....	48

FIGURA 3-29 AUTÓMATA DEL INICIADOR.....	49
FIGURA 3-30 COMPORTAMIENTO DEL INICIADOR.....	49
FIGURA 3-31 CREACIÓN DE LOS GESTORES.....	52
FIGURA 3-32 CREACIÓN DE LOS COMPONENTES DE LA APLICACIÓN DE ACCESO.....	53
FIGURA 3-33 CREACIÓN DE UNA INSTANCIA DEL PATRÓN DE AGENTES REACTIVOS.....	54
FIGURA 3-34 AUTÓMATA DEL GESTOR DE ORGANIZACIÓN.....	54
FIGURA 3-35 ARRANQUE DE LA APLICACIÓN DE ACCESO.....	55
FIGURA 3-36 TERMINACIÓN DE LA APLICACIÓN DE ACCESO.....	56
FIGURA 5-37 DIAGRAMA DE COMPORTAMIENTO PARA LA PETICIÓN DE INFORMACIÓN DE ALTA	59
FIGURA 5-38 AUTÓMATA DEL AGENTE ACCESOALTA.....	60
FIGURA 5-39 FICHERO XML CON EL AUTÓMATA DEL COMPORTAMIENTO DE ALTA.....	60
FIGURA 5-40 INTERFAZ DE USO DEL RECURSO DE VISUALIZACIÓN DE ALTA.....	61
FIGURA 5-41 ESTRUCTURA DE CLASES DEL RECURSO.....	62
FIGURA 5-42 ESTRUCTURA DE UN AGENTE REACTIVO.....	64
FIGURA 5-43 CREACIÓN DE UNA INSTANCIA DEL PATRÓN DE AGENTE REACTIVO.....	65
FIGURA 5-44 PATRÓN DE RECURSO.....	66
FIGURA 5-45 VISUALIZADOR DE ACCESO.....	67
FIGURA 5-46 ENVÍO DE INFORMACIÓN POR PARTE DEL AGENTE DE ACCESO(I).....	68
FIGURA 5-47 ENVÍO DE INFORMACIÓN POR PARTE DEL AGENTE DE ACCESO(II).....	68
FIGURA 5-48 DESCRIPCIÓN DE LA ORGANIZACIÓN(I).....	69
FIGURA 5-49 DESCRIPCIÓN DE LA ORGANIZACIÓN(II).....	70
FIGURA 5-50 ESQUEMA DE ELEMENTOS DE LA DESCRIPCIÓN.....	71
FIGURA 5-51 RELACIONES DE HERENCIA ENTRE ELEMENTOS DE LA DESCRIPCIÓN.....	72
FIGURA 5-52 DIAGRAMA UML DEL RECURSO DE CONFIGURACIÓN.....	73
FIGURA 5-53 DIAGRAMA UML PARA LA IMPLEMENTACIÓN DE LA CONFIGURACIÓN.....	74

1 Introducción

ICARO es una infraestructura software que tiene como objetivo el desarrollo de aplicaciones con Organizaciones de Agentes. ICARO proporciona “patrones agente” a partir de los cuales pueden generarse ejemplares de agentes de aplicación ejecutables en los nodos de una red de procesadores. Las aplicaciones se modelan como organizaciones formadas por agentes y recursos. En ICARO tanto el concepto de agente como el de organización tienen un modelo de diseño bien definido y una implementación robusta, flexible y abierta que ha sido pensada para simplificar el trabajo de los desarrolladores.

La presente versión de la infraestructura tiene dos patrones agente: el agente reactivo y un patrón dirigido por objetivos, que en otras ocasiones hemos llamado cognitivo. El presente manual se centra en el desarrollo de aplicaciones con el patrón de agente reactivo. El patrón dirigido por objetivos se describe en otro manual separado. Se incluye también la distribución de los agentes y de los recursos en una red de procesadores.

Toda la infraestructura está implementada en Java, su utilización está orientada a ingenieros de software con conocimientos de ingeniería software, diseño con UML y Java.

La estructura del documento es la siguiente: En el capítulo 2 se detalla el contenido de la infraestructura, se presenta la arquitectura general, y se da una visión genérica de cada uno de sus componentes. En los capítulos siguientes se detalla el proceso de instalación y los pasos necesarios para el desarrollo de aplicaciones con la infraestructura.

Los componentes de ICARO han sido utilizados en el desarrollo de aplicaciones complejas con bastante éxito. Se han obtenido reducciones significativas en tiempos de desarrollo y en número de errores, por ello nos hemos decidido a publicarlos y a recomendar su uso en proyectos de todo tipo. Tanto los patrones de agentes como el modelo organizativo pueden ser ampliados y mejorados.

Se ha hecho un esfuerzo especial para elaborar la documentación sobre el uso de la infraestructura y la documentación de ingeniería, pero somos conscientes de sus deficiencias y esperamos mejorarla con vuestra ayuda. Enviarnos por e-mail las sugerencias que creáis oportunas

Este documento sigue siendo una versión inicial que debe ir perfeccionándose con vuestras aportaciones. Los comentarios, críticas y sugerencias de mejora son bienvenidos.

2 Organización y contenido y de la infraestructura

La infraestructura puede descargarse desde GitHub (https://github.com/fgarijo/ICARO-infraestructuraV0_2) Si se descarga en la opción Download ZIP , al descomprimir los ficheros se obtiene la siguiente estructura de directorios:

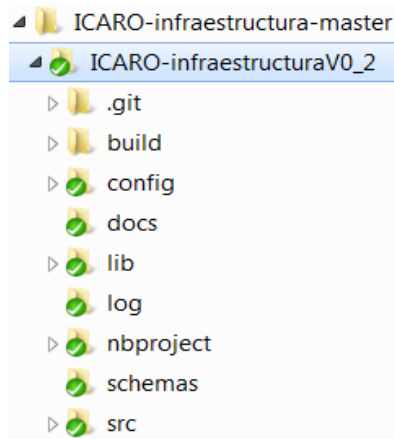


Figura 2-1 Estructura de directorios

- **config** contiene los ficheros XML que describen la organización que implementa las aplicaciones: el detalle de estos ficheros se describe más adelante.
- **docs** contiene el manual de usuario y otros documentos relativos al uso de Icaro
- **nbproject** contiene la información relevante para definir un proyecto Netbeans para trabajar con Icaro. En docs se proporciona una guía para definir un proyecto con Eclipse.
- **lib** contiene las librerías necesarias para el funcionamiento de las aplicaciones y de la infraestructura.
- **schemas** contiene esquemas XML para realizar la descripción de la estructura de la organización y otros elementos

La estructura de paquetes del código fuente contenido de la carpeta src se muestra desplegado en la Figura 2-2. Refleja el modelo de diseño, que está inspirado en una organización cuyo objetivo consiste en implementar la funcionalidad de la aplicación.

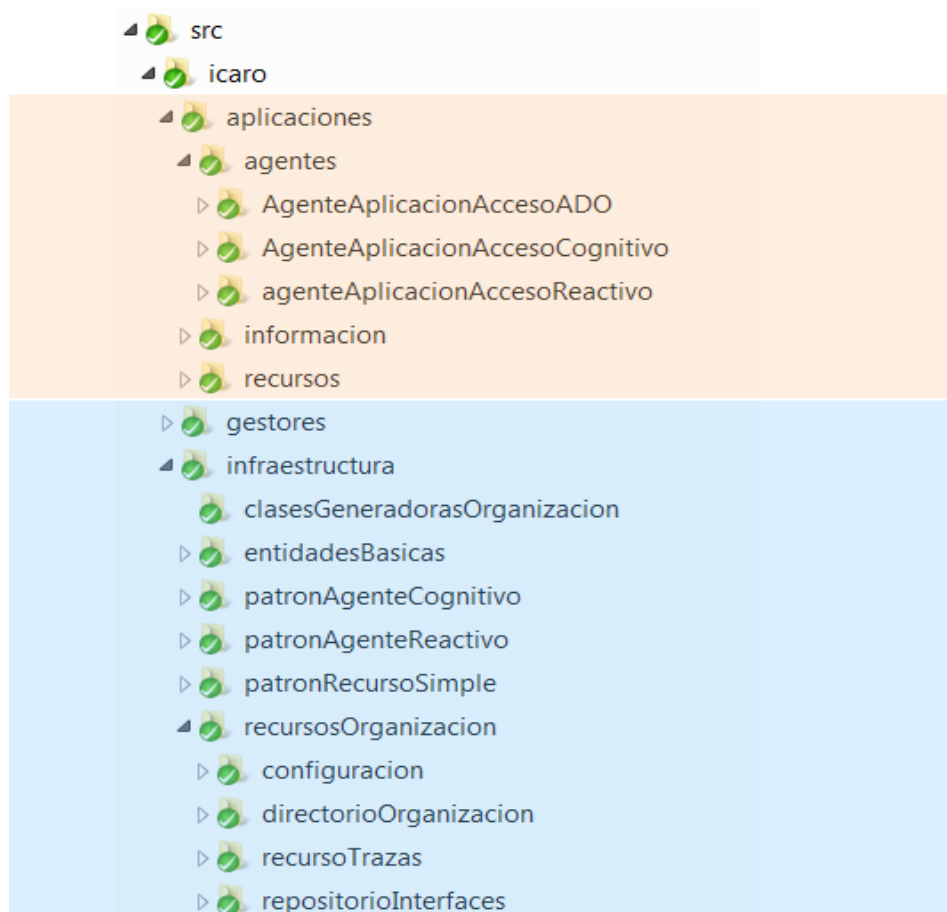


Figura 2-2 Estructura de paquetes del código fuente

El código está estructurado en dos capas:

- **La capa de aplicación.** Contiene los elementos –código y ficheros- que modelan las aplicaciones como organizaciones de agentes y recursos
- **La capa del marco de trabajo.** Contiene la infraestructura computacional necesaria para que el marco de trabajo funcione. Esta formada por **los gestores** - agentes con un comportamiento ya definido - y por **la Infraestructura** que contiene componentes y clases para la implementación de las organizaciones.

La Figura 2-3 representa las dos capas y las dependencias entre los componentes de cada capa:

La capa de aplicaciones contiene cuatro tipos de paquetes:

- **Agentes:** contienen la descripción del comportamiento de los agentes que pueden utilizarse en las aplicaciones. Un agente puede tener varios comportamientos definidos en los paquetes correspondientes.
- **Información:** contiene clases que modelan diferentes entidades de los dominios de aplicación. El paquete esta estructurado en tres paquetes para almacenar clases de dominio, objetivos y tareas
- **Recursos:** contienen el código de los recursos. Hay dos tipos de recursos comunes a gran parte de las aplicaciones: los recursos de visualización o de forma más genérica de interfaces gráficas y los recurso de persistencia. Pueden añadirse otros recursos según el tipo de aplicaciones.

La capa de aplicaciones contiene dos tipos de paquetes:

- **Gestores.** Contiene agentes predefinidos para gestionar los componentes que implementan las organizaciones o aplicaciones construidas por los usuarios. Se proporcionan tres agentes gestores: Gestor de Organización (GO) Gestor de Agentes (GA) y Gestor de Recursos (GR).

- **Infraestructura** de la organización, donde se encuentran los patrones que sirven para generar los agentes y los recursos de las aplicaciones, recursos de la organización y componentes básicos que hacen posible la implementación de los agentes y de los recursos.

La siguiente figura proporciona una visión más detallada de las dependencias entre los paquetes.

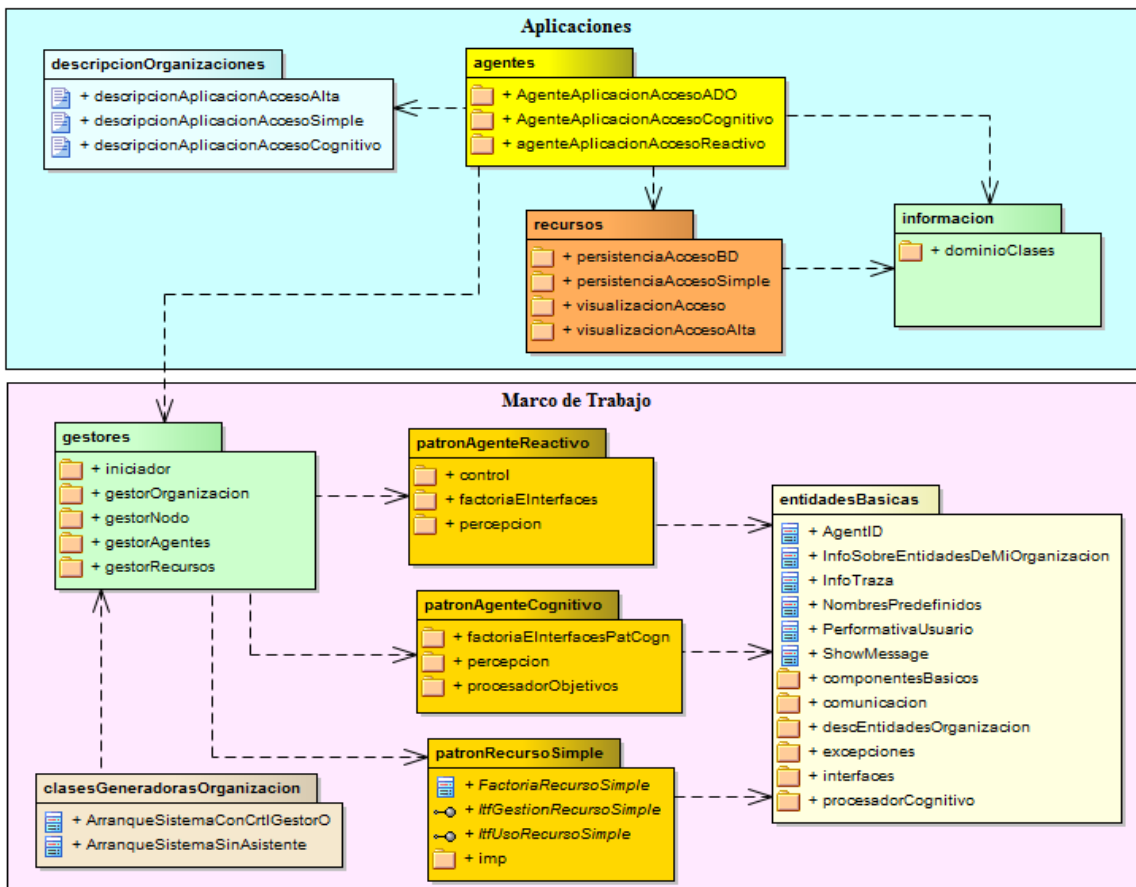


Figura 2-3 Relaciones de dependencia entre paquetes

3 Instalación y arranque de ICARO-T

3.1 Requisitos de instalación

La presente versión de ICARO funciona sobre un equipo con que tenga como mínimo las siguientes características:

Hardware: Procesador Intel Pentium IV con 512 Mb de Memoria RAM

Software:

- Sistema operativo Windows XP o superior
- Máquina virtual de Java (jre1.6.0_07) o superior

El sistema está implementado en java, puede por tanto funcionar en cualquier plataforma que soporte Java y en particular sistemas con SO Linux, pero la presente versión no ha sido probada en este entorno. Sugerimos a los usuarios que lo hagan y reporten sus experiencias.

3.2 Proceso de instalación en un PC con Windows

Para instalar ICARO seguir los siguientes pasos:

1. Descargar el fichero de código del portal : (https://github.com/fgarijo/ICARO-infraestructuraV0_2)
2. Descomprimir el fichero ICARO-xx.zip en un directorio cualquiera del PC. **Si se instala en un disco de red el script de arranque no funcionará.**
3. Comprobar que está declarada la variable de entorno JAVA_HOME y que contiene el directorio raíz correcto donde se encuentra la máquina virtual de Java. Para ello, podemos hacer lo siguiente:
 - Pulsamos sobre el icono “Mi Pc” con el botón derecho.
 - Pulsamos sobre “Propiedades”.
 - Una vez que salga la ventana, pulsamos sobre la pestaña “Opciones avanzadas”.
 - Pulsamos el botón “Variables de entorno” y buscamos la variable llamada JAVA_HOME.
 - Una configuración usual para esta variable es C:\Archivos de programa\Java\jdk1.6.0_07.
 - Si no se encuentra esta variable, hay que crearla. Para ello, pulsamos sobre el botón “Nueva”. Escribimos como nombre de variable “JAVA_HOME”, y en su valor escribimos la ruta absoluta del directorio que contiene la máquina virtual de Java (en nuestro caso, C:\Archivos de programa\Java\jdk1.6.0_07).

3.3 Prueba de funcionamiento de la infraestructura

La infraestructura contiene un ejemplo sencillo de aplicación – un sistema de acceso- que sirve como prueba de funcionamiento de la instalación y como ejemplo de desarrollo basado en una organización de agentes.

3.3.1 Ejecución de las aplicaciones de prueba con Netbeans

El proceso de arranque es el siguiente:

- Abrir Netbeans. En la pestaña projects seleccionar abrir proyecto y navegas hasta la carpeta: ICARO-infraestructura-master. Dentro de esta carpeta aparecerá el proyecto : ICARO-infraestructuraV0_2.

- Abrir el proyecto ICARO-infraestructuraV0_2
- Seleccionar el proyecto. En la barra superior de iconos hay una ventana donde debe aparecer el nombre “accesoSinBd”. Este es el nombre de la configuración del proyecto que ejecuta el sistema de acceso con el fichero de configuración: *descripcion.AplicacionAccesoSimple.xml*.
- Pulsar run. Deben aparecer la **ventana del sistema de acceso** y la **ventana de trazas**.

Ventana "Agente Acceso".

La ventana de acceso la presenta el recurso de Visualización de Acceso a partir de una orden del agente de acceso. Esto indica que el agente se mantiene activo y que los recursos necesarios para implementar la funcionalidad de acceso están listos para ser usados.

Figura 3-4 Ventana de acceso

Una vez que aparece la ventana se pueden introducir los datos de acceso que serán procesados por el agente de acceso.

El sistema termina después de intentar un acceso, haya sido válido o no.

Ventana del recurso de trazas

El recurso de trazas presenta esta ventana, en la que se pueden visualizar todas las trazas resultantes de la ejecución del ejemplo.

3.3.2 Ejecución de las aplicaciones de prueba con Eclipse

Para los que estén familiarizados con Eclipse no será difícil crear un proyecto a partir de los fuentes, las librerías y los ficheros de configuración proporcionados.

Para los que deseen información acerca de la creación del proyecto y la ejecución de sistema de acceso pueden seguir los pasos indicados en el fichero (CrearEnEclipse_ProyectoICARO.pdf) . El fichero que se debe pasar como configuración es el mismo que en NetBeans: descripcionAplicacionAccesoSimple.xml

3.3.3 Problemas en la ejecución del ejemplo.

Gran parte de los problemas con los sistemas nuevos es que **no arrancan** y cuando lo hacen producen errores extraños con mensajes incomprensibles, y a veces bloquean el PC provocando una rabia inmensa en el usuario.

El arranque de ICARO no esta exento de producir estos efectos (excepto bloquear el PC que no hemos detectado hasta la fecha). Una de las ventajas del desarrollo de aplicaciones con ICARO es que se producen menos errores, y cuando aparecen, se deben poder detectar y corregir con rapidez. No descartamos que la ejecución haya producido **uno de los fallos difíciles**, es decir, que la aplicación no haya generado ningún indicio o mensaje, o lo que es peor, que el PC haya comenzado a tener comportamientos extraños, o simplemente se queda colgado. **Si esto ocurre rogamos contactéis con el equipo de desarrollo.**

Si se crean los componentes de la organización y se producen más errores aparecerán también **en la ventana de trazas del sistema** donde es más sencillo leer e interpretar la información. Cada gestor se ocupa de la creación y de la monitorización de los elementos de la organización, con un doble clic sobre el nombre del agente o recurso que aparece en la parte derecha se abrirá la ventana con las trazas generadas durante su actividad.

Para poder interpretar el origen de los errores, sus causas y sobre todo la forma de corregirlos, es necesario conocer con un poco más detalle la arquitectura y el funcionamiento del ejemplo del sistema de acceso que describimos en el apartado siguiente.

4 Introducción al funcionamiento de los componentes de ICARO: especificación e implementación del sistema de acceso

El comportamiento del sistema de acceso puede expresarse en UML con casos de uso y con escenarios de comunicación usuario-sistema donde se definen :

- Los mensajes o la información que se intercambia entre el usuario y el sistema – por medio de ventanas-
- Las tareas internas que lleva a cabo el sistema y la información necesaria para que se realicen las tareas.

Este tipo de diagramas son típicos de la fase de análisis y sirven para definir de forma genérica la funcionalidad del sistema, al tiempo que se identifican las interfaces con el usuario y las tareas o actividades que el sistema debe realizar.

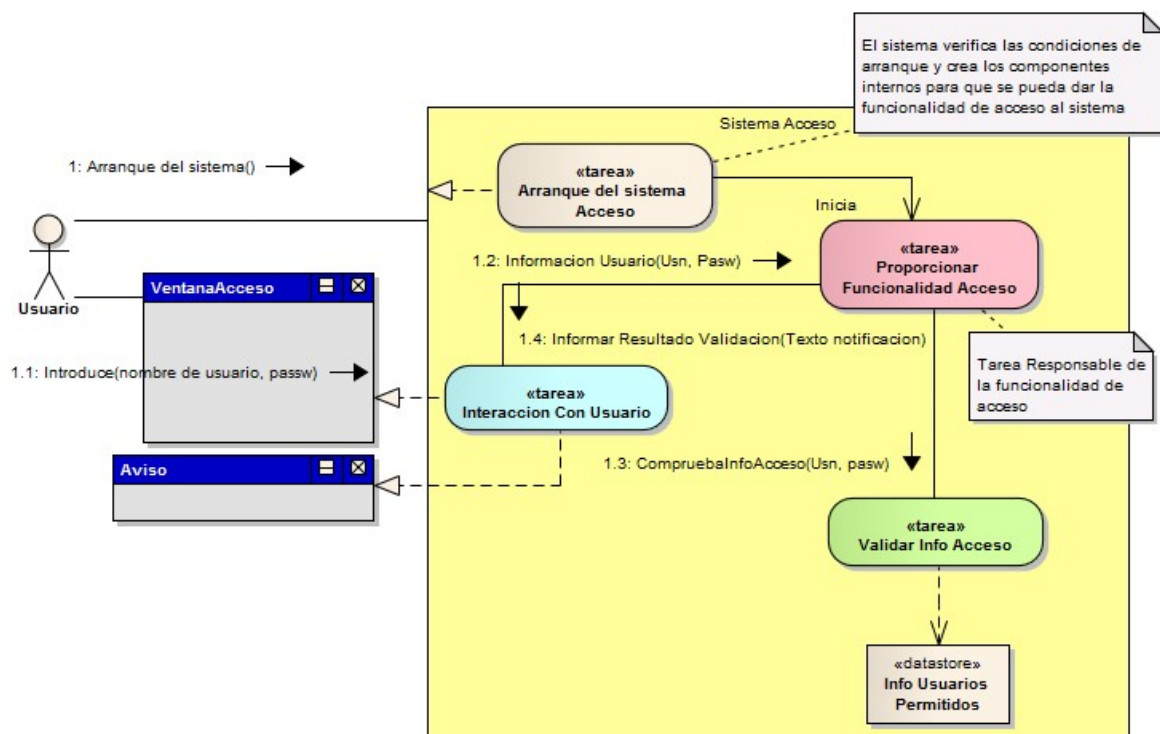


Figura 3-6 Escenario de Interacción Usuario-Sistema de Acceso

4.1 Implementación del sistema de Acceso con ICARO.

El código que implementa el sistema de acceso está contenido en la carpeta `\src\icaro\aplicaciones` dentro de los paquetes:

- `src\icaro\aplicaciones\agentes\agenteAplicacionAccesoReactivo\comportamiento`
- `src\icaro\aplicaciones\recursos\persistenciaAccesoSimple`
- `src\icaro\aplicaciones\recursos\visualizacionAcceso`

Esta estructura de paquetes sigue el modelo de diseño de la organización, por tanto vamos a introducir brevemente los pasos que se han seguido para producir el código del ejemplo.

El objetivo es construir una “organización específica” que implemente la funcionalidad del sistema de acceso utilizando la infraestructura de ICARO.

Para conseguirlo seguimos los siguientes pasos:

P1- Identificación de Agentes y Recursos de la aplicación.

- Partimos de los **casos de uso del sistema y de los requisitos** que se deben cumplir.
- **Asignamos las tareas del sistema a Agentes o a Recursos.** Se pueden asignar varias tareas a un agente o a un recurso. Los criterios de asignación dependen del tipo de tarea a realizar y de los criterios del arquitecto del sistema. Este paso da como resultado una arquitectura inicial del sistema basada en el patrón organizativo ICARO donde los componentes son Agentes y Recursos con interfaces de uso y de gestión.

P2- Definición del comportamiento Computacional de los agentes y recursos identificados. Para ello es necesario tener en cuenta las normas de construcción y el funcionamiento de los patrones proporcionados por ICARO (el objetivo del ejemplo es explicarlos)

P3- Definir la organización. En ICARO pueden coexistir distintos componentes que pueden ensamblarse para formar organizaciones. Una organización se define con un documento formal donde se especifican los agentes y los recursos que forman la aplicación, los roles de los agentes y las dependencias entre agentes y recursos. La organización es un documento que sirve para crear la organización identificando los agentes y los recursos que la forman.

En los apartados siguientes seguiremos con el ejemplo de acceso cada uno de los pasos

4.1.1 Identificación de Agentes y Recursos de la aplicación de acceso

La identificación consiste en determinar qué entidad o entidades serán **responsables de realizar** las tareas definidas en la fase de análisis.

Una posible **asignación de responsabilidades** es la siguiente:

Tarea Sistema	Tipo de Componente	Identificador Componente
Arranque del Sistema	Agente	GestorOrganización
Funcionalidad de Acceso	Agente	AgenteAcceso
Interfaz de Usuario	Recurso	RecursoVisualización
Validar Info Usuario	Recurso	Recurso de Persistencia

Las razones de esta asignación son las siguientes:

- El arranque del sistema con ICARO lo hace siempre el **GestorOrganización**, por ello esta tarea se la asignamos a este componente que ya está implementado.
- Asignamos la tarea de “**proporcionar funcionalidad de acceso**” a un agente – el **agente de acceso**-. Este agente **será el responsable de que el sistema cumpla la especificación de acceso**, es decir debe implementar el modelo de interacción con el usuario especificado en el escenario.
- Utilizamos **recursos** para implementar las tareas de **interfaz con el usuario** y **validación de los datos de acceso** por dos razones:
 - Las dos tareas representan una funcionalidad concreta y en parte bien conocida
 - En el caso de la **interfaz con el usuario** la tarea consiste en gestionar la interfaz gráfica del sistema (Vista), para que el usuario pueda intercambiar información. En el escenario se han utilizado ventanas, pero podría ser una interfaz puramente textual o incluso vocal. La tarea va a consistir en: **1) visualizar la información del sistema** de forma que el usuario pueda recibirla, y **2) adquirir la información que el usuario introduzca** para procesarla.
 - En el caso de la tarea de validación, la tarea va a consistir en comprobar si los datos introducidos por el usuario (usn y pasw) coinciden con los almacenados en la base de datos donde se guardan los datos de acceso válidos .

- Ambas tareas van a estar controladas por el agente de acceso y van a actuar como recursos suyos. El agente recibirá la información que le envíe la interfaz de usuario y utilizará la tarea de validación para conocer si permite o no el acceso.

Una vez identificados los agentes y los recursos se obtiene directamente la arquitectura inicial del sistema de acceso Figura 3-8.

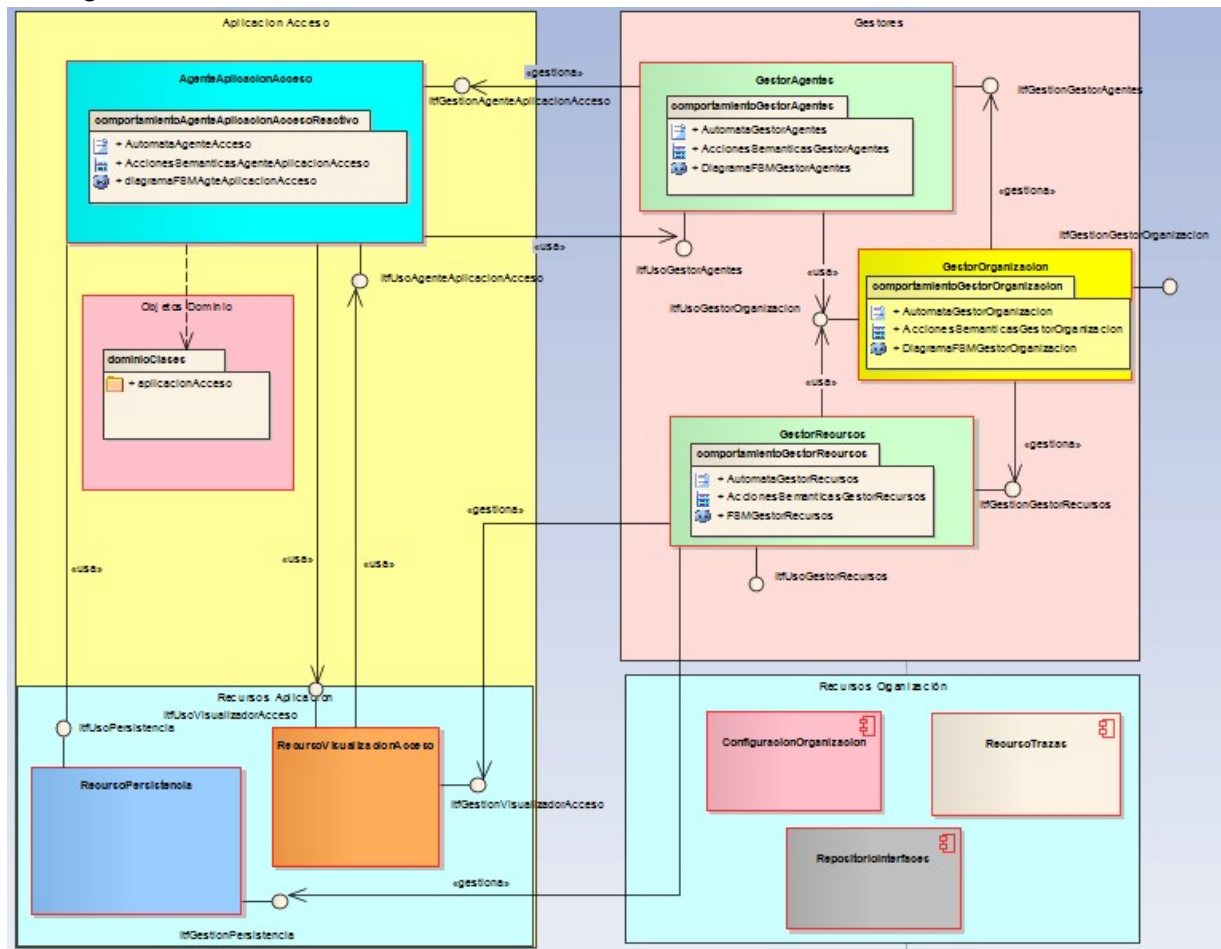


Figura 3-7 Componentes del sistema de Acceso

En la parte izquierda de la figura se encuentran los componentes que implementan la funcionalidad de la aplicación: El agente de Acceso, los recursos de visualización y de persistencia.

En la parte derecha se encuentran los componentes proporcionados por ICARO:

- **Los gestores** que se encargan de crear y monitorizar los componentes de la aplicación
- **Los recursos de la organización** que permiten la creación de la infraestructura inicial de la organización, incluidos los gestores, y que dan servicio tanto a los gestores como a los agentes y a los recursos de la aplicación.

La comunicación entre los componentes de la arquitectura se lleva a cabo a través de interfaces, de forma que la implementación interna queda oculta a los potenciales clientes de cada componente. En la figura cada componente tiene una **interfaz de uso y otra de gestión**.

A partir de la arquitectura general del sistema el paso siguiente consiste en *refinar* los componentes de la aplicación (agentes y recursos) para definir su comportamiento computacional de forma que puedan colaborar con el resto los componentes de ICARO.

4.1.2 Definición del comportamiento computacional de los Agentes y Recursos identificados.

El comportamiento puede expresarse con diagramas de comunicación UML donde se define la información intercambiada entre los componentes para implementar los escenarios correspondientes a los casos de uso. Un ejemplo concreto del comportamiento de los componentes esta en la Figura 3-9.

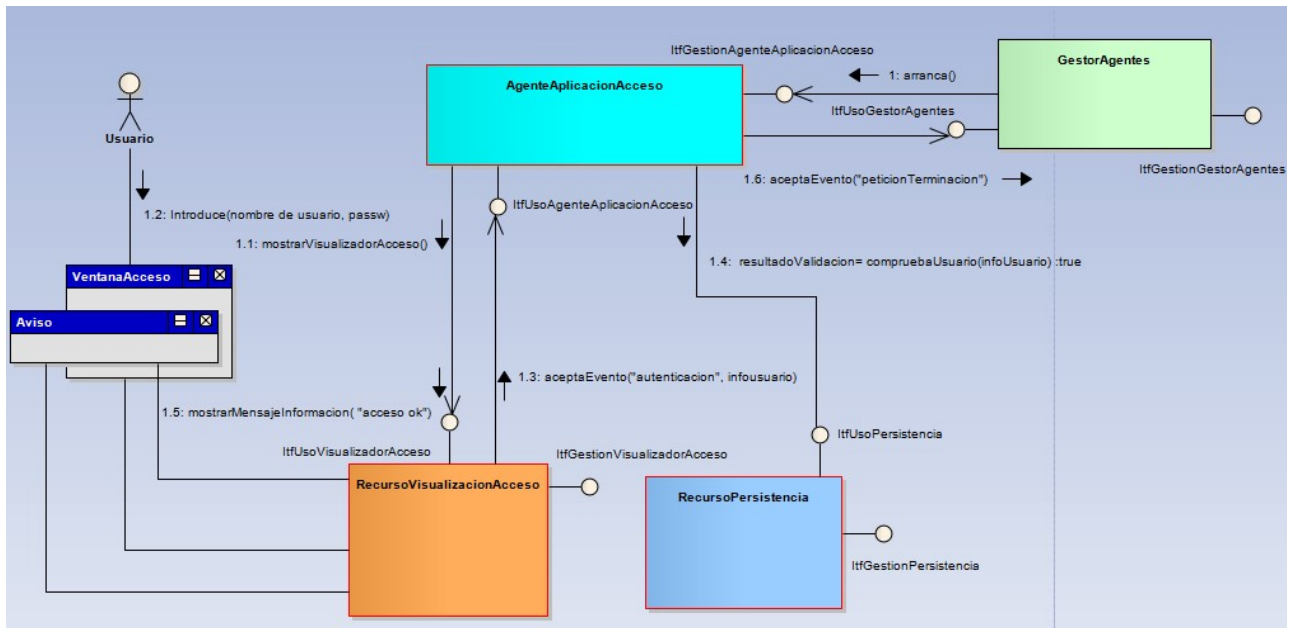


Figura 3-8 Comportamiento de los componentes del sistema de Acceso

Para su elaboración se han tenido en cuenta las siguientes ideas:

- Los gestores se encargan de la creación, arranque, parada y monitorización de los componentes de la aplicación. El gestor de organización crea y monitoriza a los gestores de agentes y de recursos, y estos crean y monitorizan los agentes y los recursos de la aplicación.
- Las interfaces de los agentes y recursos creados, se almacenan en el **repositorio de interfaces de la organización**. Agentes y recursos **pueden obtener las interfaces de otros integrantes de la organización a través del repositorio de interfaces**.
- Los **agentes son componentes controladores**, dan órdenes, reciben información, la procesan y la distribuyen a los recursos para obtener nueva información.
- El comportamiento del **Agente de Acceso** es el siguiente:
 - Controla el recurso de visualización ordenándole que abra las ventanas de interfaz con el usuario para que el usuario pueda interactuar con el sistema, y las cierre cuando el proceso de interacción haya terminado.
 - Utiliza el recurso de persistencia (BD) para validar la información introducida por el usuario
 - Espera a que el usuario introduzca los datos de acceso. Cuando le lleguen los datos introducidos por el usuario:
 - Valida los datos recibidos.
 - Comunica al usuario el resultado de la validación.
 - Espera nuevos datos si el acceso no es correcto
 - Termina cuando el usuario haya accedido al sistema.
- Los recursos ofrecen interfaces con las operaciones que necesitan los agentes. La interfaz de uso ofrece las operaciones propias de la funcionalidad del recurso. En la figura se han identificado dos operaciones del recurso de visualización de acceso (*mostrarVisualizador Acceso* y *mostrarMensajeInformacion*) y una operación del recurso de persistencia (*compruebaUsuario*)

- **La comunicación entre agentes y recursos se realiza a través de sus interfaces.** El agente o recurso obtiene la interfaz de la entidad con la que se quiere comunicar a través del Repositorio de Interfaces
- **La comunicación entre agentes y recursos** puede ser:
 - **Síncrona.** El agente da ordenes al recurso por medio de las operaciones de la interfaz de uso y recibe los resultados como retorno de la operación. Por ejemplo la operación *compruebaUsuario*.
 - **Asíncrona.** El agente da una orden por la interfaz de uso y recibe la respuesta por medio de un evento enviado por el recurso a través de la interfaz de uso del agente. Ejemplo: la información que introduce el usuario, la recoge el componente de visualización y la envía al agente.

4.1.2.1 Modelo de comportamiento del Agente de Acceso con un Autómata de Estados Finitos.

El comportamiento del agente de Acceso puede modelarse con un diagrama de estados como el de la Figura 3-10, donde los estados representan situaciones concretas del proceso de funcionamiento o del ciclo de vida del agente.

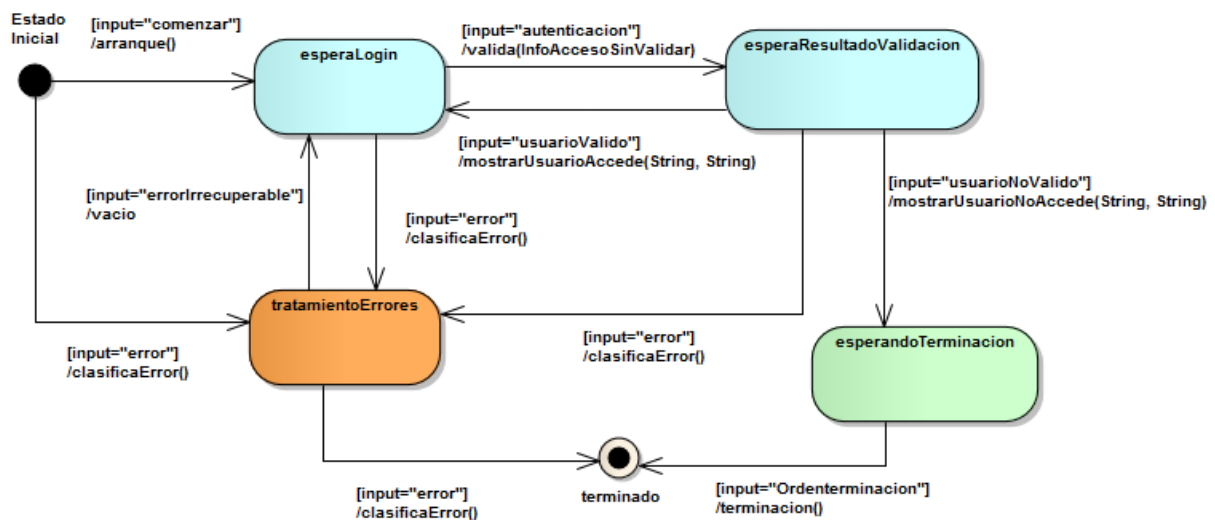


Figura 3-9 Autómata del agente de Acceso

En el diagrama se ha generalizado el comportamiento del agente de forma que sea capaz de implementar el caso de uso y otras capacidades como la detección de errores. Para ello se ha incluido un estado al que se debe transitar cuando la información que llega no es la esperada o cuando se producen errores en la ejecución de las acciones.

La interpretación del diagrama es la siguiente:

- El agente se encuentra en un estado inicial. Cuando le llega el input “comenzar”, ejecuta la acción “arranque” que consiste en obtener información acerca de recurso de visualización y darle una orden para que presente la ventana de acceso. Una vez ejecutada la acción satisfactoriamente, se espera hasta que el usuario introduzca un identificador y una contraseña.
- Cuando llega la información, se valida ejecutando una acción de consulta a la base de datos.
 - Si el usuario se encontraba registrado, se notifica al usuario de que puede acceder al sistema y finalizamos.
 - Si no, se notifica al usuario de que no es posible el acceso y se queda a la espera de nuevos datos.
 - Si se reciben errores, se transita a un estado donde se deben tratar.
 - Se pasa a un estado de terminación:
 - Cuando el usuario ha sido validado .
 - Cuando hay errores que no se pueden tratar.

Este tipo de diagramas pueden ser interpretados por una **Máquina de Estados Finitos Extendida** (MEFE) o (Extended Finite State Machine)

Una definición formal de las MEFE/EFSM puede encontrarse en:

http://en.wikipedia.org/wiki/Extended_finite_state_machine.

Existen distintos modelos computacionales para una MEFE. El patrón de agente que estamos describiendo proporciona un modelo que no es ni el mejor ni el más completo, pero que está probado y se ha revelado útil para el desarrollo de numerosas aplicaciones.

La idea básica es que **el comportamiento del agente reactivo se defina** mediante un **Autómata de Estados Finitos** (expresado con el diagrama de la figura), y que el agente sea capaz de interpretarlo. Para que la interpretación sea posible es necesario tener en cuenta lo siguiente:

- El formalismo gráfico debe ser expresado de forma textual mediante un fichero XML donde el autómata quede definido por medio de las transiciones.
- Las transiciones se modelan como relaciones entre pares de estados $T_i (S_j, S_k)$: Input/Acción. Como puede verse en el diagrama las transiciones corresponden a los arcos etiquetados con pares `<input> / <acción>` por ejemplo el arco (esperaLogin,EsperaResultadoValidación): `[input="autenticación"/valida(InfoAccesoSinValidar)]`
- Los inputs se extraen de los eventos que recibe el agente a través de la percepción.
- Las acciones se modelan como métodos de una clase denominada *AccionesSemanticasAgenteAplicacionAcceso*

El agente interpreta el autómata en la forma usual: cuando recibe un input comprueba si éste forma parte de alguna de las transiciones asociada al estado en que se encuentra, y si existe ejecuta la acción asociada y transita al estado correspondiente indicado por la transición.

En el siguiente apartado se detalla el proceso de definición del autómata para el ejemplo del sistema de acceso.

4.1.3 Implementación del comportamiento del Agente de Acceso en ICARO

La implementación del comportamiento del Agente de Acceso con ICARO requiere la definición de dos elementos:

- El fichero “*automata.xml*”, que contiene la codificación del autómata de estados finitos en xml — Figura 3-11
- La clase *AccionesSemanticasAgenteAplicacionAcceso*. Esta clase proporciona los métodos que implementan las acciones del autómata –Figura 3-12. Estas acciones constituyen el “**modelo de actuación del agente**” e implementan “**la reacción del agente ante los estímulos exteriores**”.

Estos elementos se encuentran definidos en la **carpeta comportamiento** situada en la ruta:

icarol\aplicaciones\agentes\agenteAplicacionAccesoReactivo.

Un agente de aplicación puede tener varios comportamientos definidos. Cada comportamiento se sitúa en la carpeta comportamiento. Para el *agenteAplicacionAccesoReactivo*, se han definido dos comportamientos – comportamiento y comportamientoAlta- cada uno de ellos contiene una clase *accionesSemanticasagenteAplicacionAcceso* y un fichero *automata.xml* con la tabla de estados.

4.1.3.1 Definición de la tabla de transiciones del Autómata

La figura siguiente contiene las transiciones del autómata correspondiente a uno de los comportamientos del agente de acceso

```

- <tablaEstados descripcionTabla="Tabla de estados en pruebas para el agente de acceso">
- <estadoInicial idInicial="estadoInicial">
  <transicion input="comenzar" accion="arranque" estadoSiguiente="esperaLogin" modoDeTransicion="bloqueante" />
</estadoInicial>
- <estado idIntermedio="esperaLogin">
  <transicion input="autenticacion" accion="valida" estadoSiguiente="esperaResultadoValidacion" modoDeTransicion="bloqueante" />
  <transicion input="peticion_terminacion_usuario" accion="pedirTerminacionGestorAgentes" estadoSiguiente="esperandoTerminacion"
    modoDeTransicion="bloqueante" />
</estado>
- <estado idIntermedio="esperandoTerminacion">
  <transicion input="termina" accion="terminacion" estadoSiguiente="terminado" modoDeTransicion="bloqueante" />
</estado>
- <estado idIntermedio="esperaResultadoValidacion">
  <transicion input="usuarioValido" accion="mostrarUsuarioAccede" estadoSiguiente="esperandoTerminacion" modoDeTransicion="bloqueante" />
  <transicion input="usuarioNoValido" accion="mostrarUsuarioNoAccede" estadoSiguiente="esperandoTerminacion" modoDeTransicion="bloqueante" />
</estado>
- <estado idIntermedio="tratamientoErrores">
  <transicion input="errorIrrecuperable" accion="terminacion" estadoSiguiente="terminado" modoDeTransicion="bloqueante" />
  <transicion input="errorRecuperable" accion="nula" estadoSiguiente="esperaLogin" 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 3-10 Ejemplo definición del autómatas del agente de Acceso en XML

Como puede observarse el contenido del fichero *automata.xml* es bastante legible y coincide (aproximadamente) con la descripción gráfica. Podemos destacar sin embargo las siguientes características.

- Hay tres tipos de estados: inicial, intermedio y final
- Se mencionan una modalidad de transiciones como bloqueante
- Hay dos tipos de transiciones: normales y universales
- En el estado “**esperaLogin**” se menciona un input “**peticion_terminacion_usuario**” y una acción “**pedirTerminacionGestorAgentes**” que no existen en el diagrama gráfico.

Las diferencias entre el diagrama gráfico y el fichero se deben a que el primero es un modelo de diseño y el segundo es un modelo interpretable por el agente. El hecho de tener una transición más se debe a que se partió de un autómatas consistente con el diagrama de diseño, pero se observó que el comportamiento no era el adecuado, ya que el agente debe terminar cuando lo pide el usuario, cuando lo ordena el gestor o cuando detecta un error irrecuperable. Esta situación no se contempla en el diagrama inicial. Para conseguirlo que termine cuando el usuario lo pide es necesario introducir esta nueva transición. Esto puede hacerse directamente – y el desarrollador lo ha hecho así- modificando únicamente el fichero xml.

Para poder trabajar con el fichero XML es necesario conocer

- El meta-modelo que define las reglas sintácticas del autómatas.
- Las semántica asociada a las construcciones sintácticas, en este caso el tratamiento de los inputs, la ejecución de las acciones, y de las transiciones.

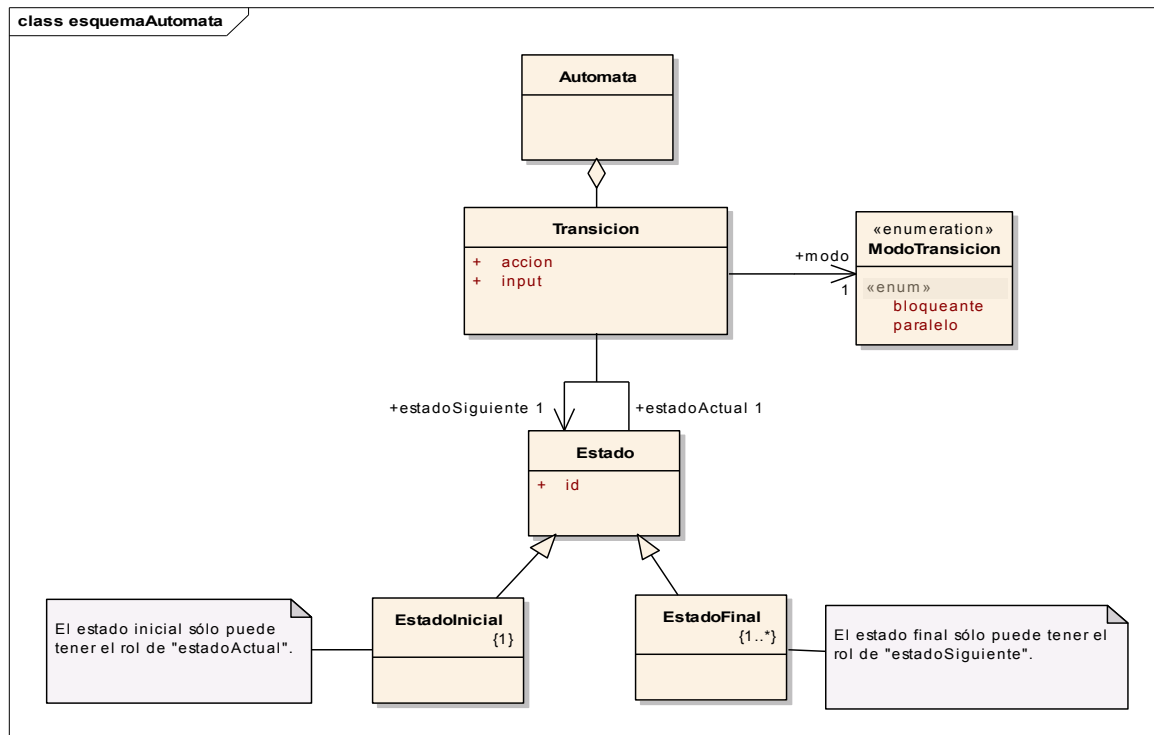


Figura 3-11 Esquema para modelar autómatas

El meta-modelo define las entidades que forman el autómata y las relaciones entre ellas. A partir del meta-modelo puede obtenerse un DTD (Data Type Definition) que se utiliza para validar sintácticamente los ficheros XML Figura 3-17

```

2 <!-- DTD para Tabla de estados. Aumentada con funcionalidad adicional-->
3 <!--Las transiciones universales son aquellas que, desde cualquier estado,
4 con el input especificado, van a un estado siguiente, ejecutando una acción
5 con un modo de ejecución. Son una forma de ahorrar líneas de definición de
6 transiciones, ya que habría que poner una transición con ese input en cada
7 estado para lograr el efecto que se consigue con las transiciones universales.-->
8 <!ELEMENT tablaEstados (estadoInicial,estado*,estadoFinal+,transicionUniversal*)>
9 <!ATTLIST tablaEstados
10     descripcionTabla CDATA #IMPLIED>
11 <!ELEMENT estadoInicial (transicion+)>
12 <!ATTLIST estadoInicial
13     idInicial ID #REQUIRED>
14 <!ELEMENT estado (transicion+)>
15 <!ATTLIST estado
16     idIntermedio ID #REQUIRED >
17 <!ELEMENT estadoFinal EMPTY>
18 <!ATTLIST estadoFinal
19     idFinal ID #REQUIRED>
20 <!ELEMENT transicionUniversal EMPTY>
21 <!ATTLIST transicionUniversal
22     input CDATA #REQUIRED
23     accion CDATA #REQUIRED
24     estadoSiguiente IDREF #REQUIRED
25     modoDeTransicion (bloqueante | paralelo) #REQUIRED>
26 <!ELEMENT transicion EMPTY>
27 <!ATTLIST transicion
28     input CDATA #REQUIRED
29     accion CDATA #REQUIRED
30     estadoSiguiente IDREF #REQUIRED
31     modoDeTransicion (bloqueante | paralelo) #REQUIRED >
32

```

Figura 312 Modelo del autómata con un DTD

En el DTD se expresa formalmente lo que el autómata debe tener .

- Un solo estado inicial, al menos un estado final y todos los estados, transiciones y transiciones universales que necesitemos.
- Las transiciones tienen asociada una modalidad que puede ser bloqueante o paralelo
- Hay transiciones universales y transiciones (normales).

La semántica de las construcciones sintácticas es la siguiente:

- **Tipos de transiciones.** 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. Vemos con un ejemplo este meta-modelo expresado por medio de un fichero DTD.
- **Modalidades de la transición.** Indican la forma en que se ejecutará la acción que forma la transición. Si es **bloqueante**, el interprete transita al estado indicado en el input termina la ejecución de la acción. Esto implica que si la acción no termina el autómata se quedará bloqueado. Esto puede ocurrir y da lugar a que el agente se quede inactivo, dado que no puede procesar más inputs. Para evitarlo –la solución consiste en ejecutar la acción en un hilo (thread) diferente - modalidad **no bloqueante** -, de esta forma si la acción se queda bloqueada el agente puede seguir funcionando. Las acciones que se ejecutan en paralelo llevan asociadas timeouts, con lo cual solo se quedan bloqueadas durante el tiempo que se especifique en el timeout. Cuando vence el timeout se genera un evento que puede ser procesado por el agente.

4.1.3.2 Definición de las acciones.

Las acciones que forman parte de las transiciones del autómata, se definen como métodos de una clase con los siguientes requisitos:

- **Nombrado:** el identificador de la clase debe ser: “*AccionesSemanticasAgenteAplicacion<NombreAgente>*”. En el sistema de acceso se llama “*AccionesSemanticasAgenteAplicacionAcceso*”
- **Herencia:** debe heredar de la clase “*AccionesSemanticasAgenteReactivo*” que se encuentra en el paquete : “*src\icaro\infraestructura\PatronAgenteReactivo\control\acciones*”
- **Localización:** debe situarse en el paquete:
“*src\icaro\aplicaciones\agentes\agenteAplicacionAccesoReactivo\comportamiento*”
- **Métodos:** El nombre de los métodos debe coincidir con el identificador de las acciones definidas en la tabla de estados del autómata. Por ejemplo si tenemos la transición:
<transicion input="comenzar" accion="arranque" estadoSiguiente="esperaLogin" modoDeTransicion="bloqueante"/>

En la clase *Acciones semánticasAgenteAplicacionAcceso* deberá existir un método *arranque*

Se ha subrayado el nombre del paquete *agenteAplicacionAccesoReactivo* porque este paquete tiene que tener un nombre conforme con la reglas de nombrado que detallaremos más tarde.

Tanto las reglas de nombrado como la organización de los paquetes tienen como objetivo facilitar la estructuración del código y su legibilidad. Esto simplifica el trabajo en equipo ayudando a la detección y a la corrección de errores.

La Figura 3-14 contiene los atributos y los métodos de las dos clases:

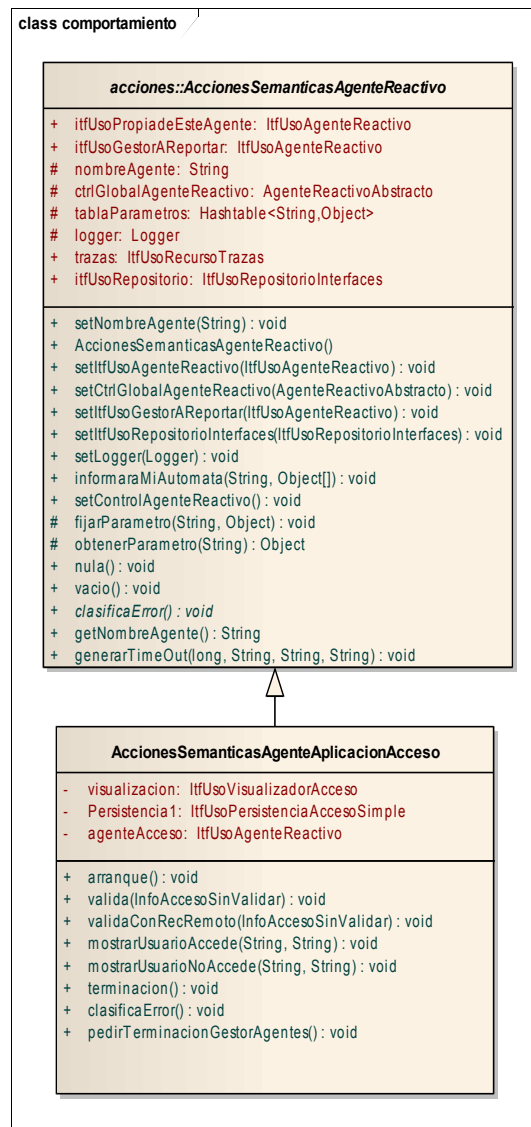


Figura 3-13 Acciones semánticas del agente reactivo de Acceso

La clase “*AccionesSemanticasAgenteReactivo*” proporciona métodos del patrón de agente reactivo comunes a todos los agentes reactivos y que se desarrollen a partir del patrón.

El desarrollador tiene libertad para la implementación de las acciones. La recomendación es que sean sencillas, es decir que **el agente ordene que se hagan las cosas y controle que se han hecho**, pero que **delegue en los recursos o en otros agentes la complejidad de ejecutar las ordenes**.

Las acciones del agente de acceso son las siguientes:

Arranque

- Entradas: ninguna
- Salida: ninguna
- Efecto: visualización de la ventana de acceso
- Proceso: se ordena al recurso “Visualizacion” a través de su interfaz de uso que visualice la ventana de acceso.

Valida

- Entradas: un usuario (String) y una contraseña (String)
- Salida: booleano con el resultado de la validación

- Efecto: resultado de la validación (válido, inválido, sin respuesta o error)
- Proceso: Se accede al recurso de persistencia a través de la interfaz de uso, quien comprueba si el nombre del usuario está en la base de datos y la contraseña es correcta. Si esto es así, se envía el Evento “usuarioValido”; si no, se envía el Evento “usuarioNoValido”. De esta forma provocará una transición de estado.

MostrarUsuarioAccede

- Entradas: un usuario (String) y una contraseña (String)
- Salida: ninguna
- Efecto: aparece un mensaje anunciando que se pudo acceder.
- Proceso: Se ordena al recurso de visualización que muestre un mensaje de información anunciando que el usuario ha podido acceder.

MostrarUsuarioNoAccede

- Entradas: un usuario (String) y una contraseña (String)
- Salida: ninguna
- Efecto: aparece un mensaje anunciando que no se pudo acceder.
- Proceso: Se ordena al recurso de visualización que muestre un mensaje de información anunciando que el usuario no ha podido acceder.

ClasificaError

- Entradas: ninguna
- Salida: ninguna
- Efecto: se envía el Evento “errorIrrecuperable” como política de tratamiento de errores.
- Proceso: se realiza el envío del evento a través de su interfaz de Uso.

Terminacion

- Entradas: ninguna
- Salida: ninguna
- Efecto: simplemente se anuncia por el logger que ha llegado una petición de terminación.
- Proceso: mensaje para el logger.

PedirTerminacionGestorAgentes

- Entradas: ninguna.
- Salida: se manda un Evento al gestor de agentes para indicarle que desea terminar.
- Efecto: el agente recibe un nuevo evento que provocará una transición.
- Proceso: se realiza el envío del evento al gestor de agentes través de su interfaz de Uso.

Puntos a tener en cuenta en las acciones del ejemplo:

- Las acciones que implementan **ordenes de visualización** se limitan a enviar la orden. El resultado de la orden no se considera relevante (salvo si se produce una excepción)
- La acción **Valida** se delega en el recurso de persistencia. Se obtiene el resultado de la validación como un valor de retorno en el uso del método de la interfaz. Es la propia acción quien genera un evento para que el agente se entere de que la validación ha sido correcta y transite.
- Las acciones pueden generar eventos que a su vez desencadenan acciones y transiciones
- Las acciones del ejemplo son todas **bloqueantes**, lo cual implica que si alguna de ellas no termina (se queda bloqueada), el agente se queda inactivo y la aplicación fuera de servicio.

4.1.4 Recursos del sistema de acceso

Los recursos que implementan el sistema de acceso están situados en el directorio:

“\src\icaro\aplicaciones\recursos”

Puede observarse que hay varias carpetas que contienen el código de distintos recursos.

El sistema de acceso utiliza dos recursos : *visualizacionAcceso* y *persistenciaAccesoSimple*.

El contenido de las carpetas tiene una estructura similar con dos elementos comunes:

- Una **clase que define la interfaz de uso del recurso**. Esta interfaz especifica las operaciones accesibles por los clientes potenciales del recurso
- Una carpeta donde se encuentra el código correspondiente a la implementación de las operaciones definidas en la interfaz.

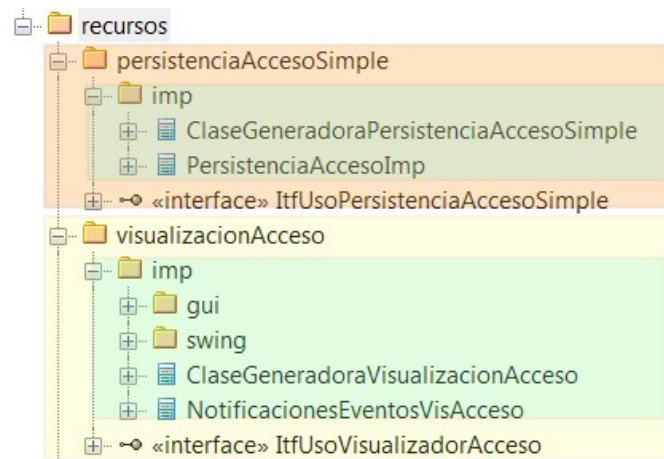


Figura 3-14 Estructura de carpetas de los recursos de las aplicaciones

En la carpeta *imp* se encuentra una **clase generadora** que se encarga de la creación del modelo computacional del recurso.

Los recursos en ICARO tienen las siguientes características:

- Los recursos son **componentes que deben ser creados y gestionados por el Gestor de Recursos**.
- Deben proporcionar **interfaces de uso y de gestión al resto de componentes de la organización**. En el modelo estático no aparecen las interfaces de gestión porque son comunes a todos los recursos, y por ello se crean a partir de un patrón.
- **El desarrollador debe definir e implementar las interfaces de uso** y responsabilizarse de que las interfaces cumplan los requisitos de diseño de forma independiente del resto de componentes. El recurso debe por tanto poderse activar independientemente del resto, probarse, desplegarse y utilizarse sin provocar efectos colaterales, en los componentes clientes.

ICARO facilita el proceso de conversión de un sistema legado en un recurso. Para ello es necesario recubrirlo de forma que sólo se vea su funcionalidad a través de las interfaces de uso y para que sea gestionable a partir de las interfaces de gestión. El proceso de construcción de un recurso, así como la conversión de un componente en un recurso se verá con más detalle en el apartado ().

La Figura 3-16. representa el modelo estático de los recursos *visualizacionAcceso* y *persistenciaAccesoSimple* con las operaciones concretas que proporcionan sus interfaces de uso.

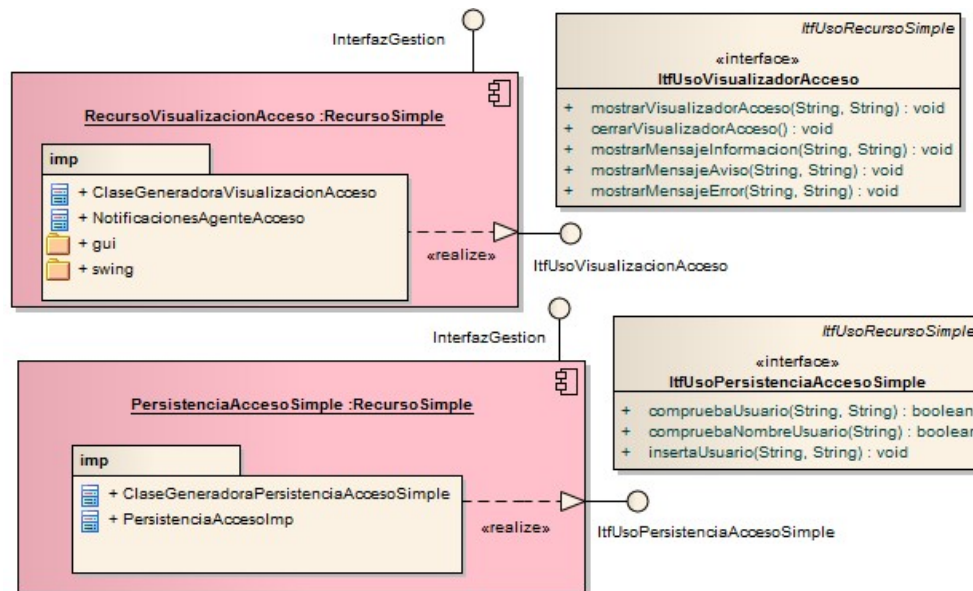


Figura 3-15 Vista estática de los recursos del sistema de acceso

Las operaciones que aparecen en la interfaz de uso han sido diseñadas **para que el agente trabaje lo menos posible** – es decir para que el modelo de control sea lo más sencillo posible –

- Es preferible **que los recursos proporcionen interfaces sencillas**, aunque la implementación sea compleja a tener interfaces con implementación sencilla **pero que obliguen a los agentes clientes a tener un modelo de control complejo** con muchos estados y operaciones .

Los detalles sobre las operaciones ofrecidas a los potenciales clientes se detallan a continuación

4.1.4.1 Recurso de Visualización de Acceso

Las operaciones del Recurso de Visualización de Acceso son bastante intuitivas y aparecen de forma natural cuando se hacen los casos de uso del sistema.

- **mostrarVisualizadorAcceso:**
 - Entradas: ninguna
 - Salida: se muestra la ventana de acceso por pantalla.
- **cerrarVisualizadorAcceso**
 - Entradas: ninguna
 - Salida: se cierra el visualizador, dejándolo de mostrar por pantalla.
- **mostrarMensajeAviso**
 - Entradas: titulo del mensaje (String) y contenido (String)
 - Salida: muestra por pantalla una ventana de aviso con el titulo y el mensaje dados.
- **mostrarMensajeError**
 - Entradas: titulo del mensaje (String) y contenido (String)
 - Salida: muestra por pantalla una ventana de error con el titulo y el mensaje dados.
- **mostrarMensajeInformacion**
 - Entradas: titulo del mensaje (String) y contenido (String)
 - Salida: muestra por pantalla una ventana de información con el titulo y el mensaje dados.

4.1.4.2 Recurso de Persistencia

El recurso de persistencia es el encargado almacenar los elementos computacionales de la aplicación. En este caso se ha utilizado como tecnología una librería que guarda los datos asociados a claves de acceso. Existe otro recurso de persistencia ***persistenciaAccesoBD*** cuya interfaz tiene las mismas operaciones pero basado

en una base de datos relacional implementada con MySQL. Para que funcione correctamente **MySQL** debe estar instalada en el nodo.

Las operaciones del recurso son las siguientes:

- **compruebaNombreUsuario:**
 - Entradas: un nombre de usuario (String)
 - Salida: true si el nombre de usuario pasado por parámetros se halla en la base de datos, y false en caso contrario.
- **compruebaUsuario**
 - Entradas: un nombre de usuario (String) y una contraseña (String)
 - Salida: true si existe el nombre de usuario con la contraseña especificada en la base de datos, y false en caso contrario.
- **insertaUsuario**
 - Entradas: un nombre de usuario (String) y una contraseña
 - Salida: ninguna. El efecto es que se añade un usuario a la base de datos con el nombre y contraseña indicados.

4.1.5 Comunicación asíncrona entre Agentes y Recursos

La comunicación entre Agentes y Recursos puede ser síncrona : cuando el agente utiliza la interfaz de uso de un recurso, o asíncrona cuando un recurso envía información a un agente por medio de eventos. La comunicación entre agentes es asíncrona por medio de mensajes.

Eventos y mensajes son **entidades contenedoras de información**. En ICARO se proporcionan las clases **EventoSimple** y **MensajeSimple** para que puedan ser extendidas según las necesidades de las aplicaciones. La idea es disponer una taxonomía reusable de eventos y de mensajes Figura 3-17.

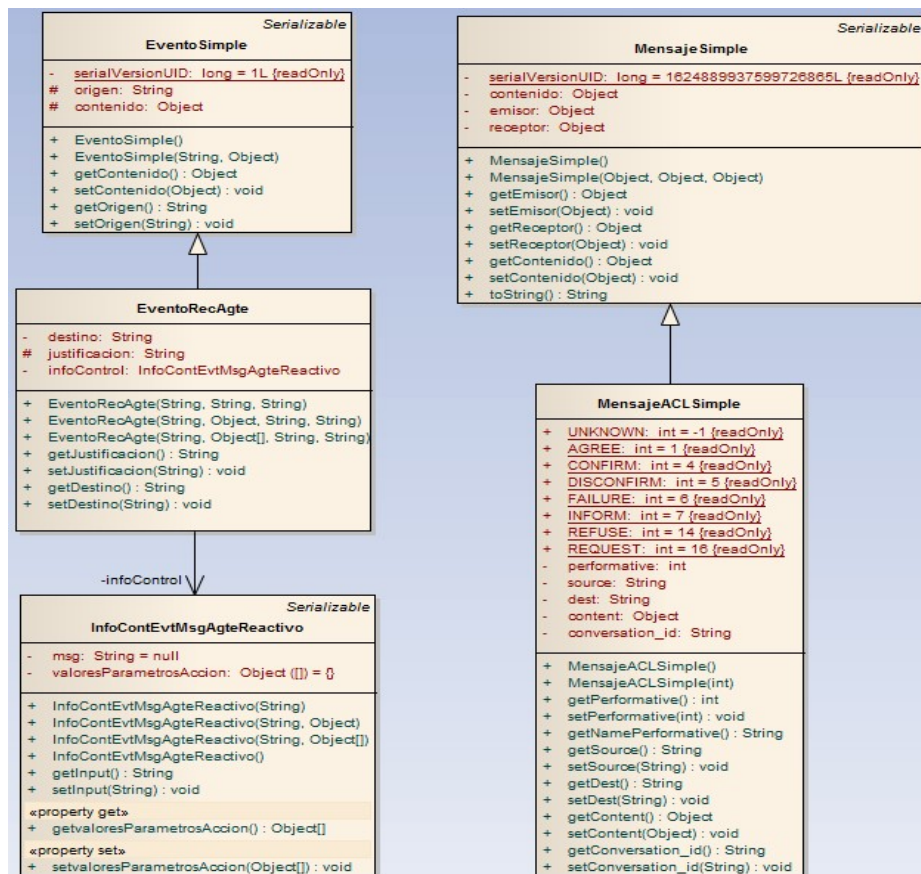


Figura 3-16 Taxonomía de eventos y mensajes

Estas clases están situadas en la carpeta : “src\icaro\infraestructura\entidadesBasicas\comunicacion”.

4.1.5.1 Ejemplos de comunicación asíncrona entre agentes y recursos en el sistema de Acceso

Como introducción a la descripción del modelo de eventos y de mensajes en ICARO veamos algunos ejemplos concretos de comunicación asíncrona entre agentes y recursos en el Sistema de acceso

Ejemplo 1: *El recurso de visualización de acceso comunica al Agente de acceso la información introducida por el usuario en la ventana de acceso*

La implementación de esta comunicación se encuentra en la clase : **NotificacionesEventosVisAcceso**

Que esta situada en la carpeta: **src\icaro\aplicaciones\recursos\visualizacionAcceso\imp**

El código java que implementa el envío de la información es el siguiente:

En el método: **peticionAutenticacion(String username, String password)**

Se crea un objeto de la clase: **InfoAccesoSinValidar**

```
InfoAutenticacion = new InfoAccesoSinValidar(username,password);
```

A continuación se utiliza el método:

```
public void enviarEventoOtroAgente(EventoRecAgte eventoAEnviar,String IdentAgenteReceptor)
```

que implementa el envío de la información al agente de acceso. En este método se realizan las siguientes acciones:

1. Obtención de la interfaz de uso del agente receptor de la información

```
itfUsoAgenteReceptor=(InterfazUsoAgente)  
NombresPredefinidos.REPOSITORIO_INTERFACES_OBJ.obtenerInterfaz  
(NombresPredefinidos.ITF_USO+IdentAgenteReceptor);
```

2. Envío del evento al agente receptor

```
itfUsoAgenteReceptor.aceptaEvento(eventoAEnviar);
```

Ejemplo 2: *El agente de Acceso envía un mensaje de petición de terminación al Gestor de agentes que a su vez la transmite al gestor de organización*

La implementación de esta comunicación se encuentra en la clase :

AccionesSemanticasAgenteAplicacionAcceso

Que esta situada en la carpeta:

src\icaro\aplicaciones\agentes\agenteAplicacionAccesoReactivo\comportamiento

En el método: **public void pedirTerminacionGestorAgentes()**

El código java que implementa el envío de la información es el siguiente:

```
this.itfUsoGestorAReportar.aceptaMensaje(new MensajeSimple (  
new InfoContEvtMsgAgteReactivo("peticion_terminar_todo"),  
this.nombreAgente,  
NombresPredefinidos.NOMBRE_GESTOR_AGENTES)  
);
```

En ambos casos la entidad emisora realiza los siguientes pasos:

1. Crea el evento o el mensaje que pretende enviar
2. Obtiene la interfaz de la entidad receptora del mensaje o evento utilizando el repositorio de interfaces
3. Envía el evento o mensaje utilizando la operación **aceptaEvento** o **aceptaMensaje** de la interfaz de uso del receptor.

A continuación se detalla el modelo de eventos y de mensajes.

4.1.5.2 Modelo de eventos y de mensajes en ICARO

Eventos y mensajes se caracterizan por una **envoltura y un contenido**. En la envoltura se proporciona información sobre **quien emite la información, cuando se generó, a quien va dirigida, etc.** y en el contenido se define **sobre qué se pretende informar**. Las clases *EventoSimple* y *MensajeSimple* modelan una **envoltura mínima y un contenido genérico**. Proporcionan atributos y operaciones para identificar:

- **La entidad emisora del evento o mensaje.** Esta información se modela como un identificador en el caso de los eventos y como un objeto en el caso de los mensajes
- **Información contenida en el evento o mensaje.** Esta información se modela como un objeto en ambos casos.

La clase mensaje incluye en su envoltura un atributo **receptor** para identificar al receptor del mensaje y operaciones para obtener y modificar su contenido.

En la misma carpeta se han definido las clases *EventoRecAgente* y *MensajeACLSimple* que heredan de *EventoSimple* y de *MensajeSimple*.

- *EventoRecAgente* añade a la envoltura del evento **atributos y operaciones para especificar: destino** - identificador de la entidad destinataria del evento -, y **justificación** para expresar anotaciones sobre el evento.
- *MensajeACLSimple* es una clase que simplifica el modelo de mensajes de FIPA. Añade a la envoltura atributos y operaciones para definir la performativa, y el *conversationId*.

El contenido de un mensaje o de un evento puede ser cualquier objeto, sin embargo **el objeto debe tener un valor semántico para el agente** es decir **debe poder ser interpretado por la percepción y/o por el control del agente**.

En el caso del agente reactivo la interpretación del contenido extraído de un evento o mensaje debe provocar que el **autómata que modela el control del agente**, cambie de estado y ejecute acciones.

Para facilitar el entendimiento entre los emisores y los agentes reactivos se proporciona la clase : *InfoContEvtMsgAgteReactivo*.

Esta clase permite a las entidades emisoras de eventos o mensajes generar contenidos que puedan ser interpretados por el **autómata de estados finitos que implementa el control del agente reactivo**. Para ello es necesario establecer una correspondencia entre el suceso que se pretende reportar y los inputs y acciones del **autómata de estados finitos que definen el comportamiento del agente**. La clase contiene dos atributos:

- **msg.** Permite identificar el suceso o información que transporta el evento o mensaje. Por ejemplo si el evento pretende informar sobre datos introducidos por el usuario se puede asociar a **msg** el valor “infoUsuario”, “userInput” o cualquier otra etiqueta que permita al agente receptor del evento interpretarlo.
- **msgElements.** Representa la información adicional del suceso identificado en el atributo **msg**. En el ejemplo anterior, (msg= infoUsuario), se asociaría a **msgElements** un objeto o una lista de objetos representando la información introducida por el usuario (por ejemplo un objeto con los valores de usn y pasw introducidos por el usuario).

En el ejemplo del sistema de acceso se utiliza la clase *InfoContEvtMsgAgteReactivo* de forma indirecta a través de la clase *EventoRecAgente*. Esta clase proporciona operaciones para crear directamente la información asociada al atributo msg y msgElements; en la implementación de las operaciones puede observarse que crea un objeto de la clase *InfoContEvtMsgAgteReactivo* que se incluye en el contenido del evento. La utilización de la clase *EventoRecAgente* permite al emisor simplificar la creación y el envío de eventos. Si no utilizara esta clase debería : 1) crear un objeto *InfoContEvtMsgAgteReactivo* con la información a enviar y 2) crear un *eventoSimple* que tenga como contenido el objeto creado.

La clase *MensajeACLSimple* añade atributos y operaciones para implementar mensajes FIPA.

4.1.5.3 Generación e interpretación de eventos y mensajes.

El modelo de comunicación implementado en ICARO esta representado en la Figura 3-18

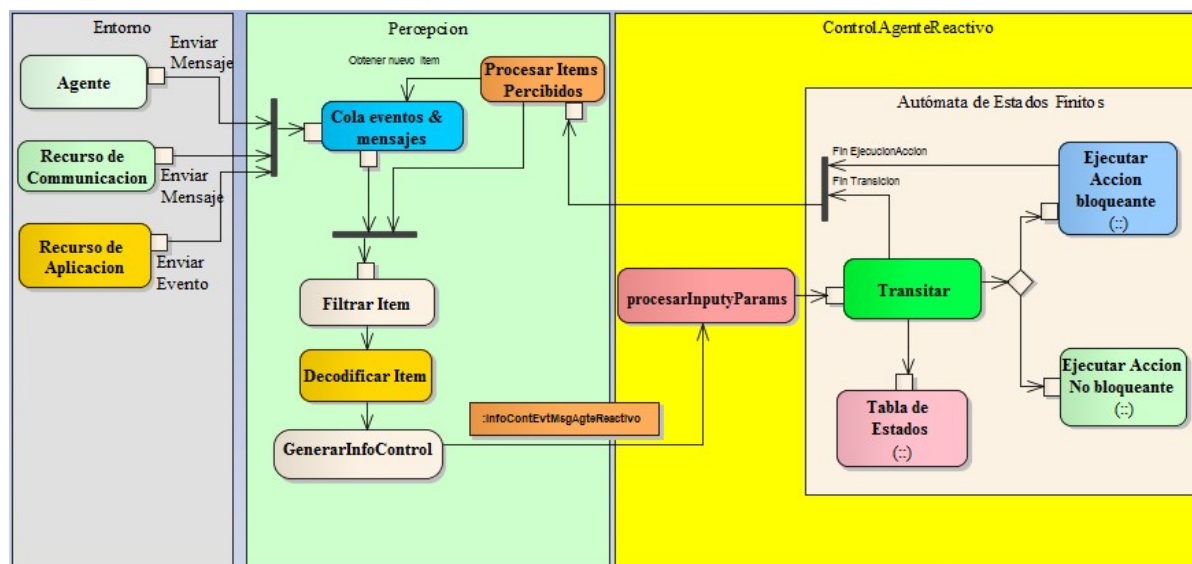


Figura 3-17 Comunicación entre emisores y receptores de eventos

Los agentes reactivos tienen capacidad para recibir, emitir y procesar eventos y mensajes.

Su ciclo de funcionamiento es el siguiente Figura 3-18:

- Reciben eventos o mensajes concurrentemente de distintos componentes a través de la interfaz de uso. El item recibido se envía a la percepción.
- La percepción
 - Proporciona interfaces para almacenar y extraer los eventos y mensajes recibidos
 - Clasifica los eventos o mensajes recibidos por el agente. Como resultado el item puede ser:
 - Eliminado si no se satisfacen determinados requisitos, por ejemplo en el caso de los mensajes que el identificador del receptor del mensaje no coincida con el identificador de agente.
 - Almacenado, para ser tratado posteriormente de acuerdo con el estado interno del agente.
 - Activa un thread (EnvioItemsThread) para procesar los eventos y mensajes almacenados. El procesamiento consiste en los siguientes pasos:
 - Selección de los mensajes de la cola
 - Filtrado del item obtenido. Actualmente no se realiza ningún filtro, pero podría introducirse filtros contextuales o temporales - por ejemplo eliminar los o eventos mensajes del mismo tipo que se produzcan en un determinado intervalos temporal.
 - Extracción y verificación del contenido del evento o mensaje. La verificación consiste en comprobar que el evento o mensaje contienen un objeto de la clase InfoContEvtMsgAgteReactivo. Esta clase se encuentra en la carpeta **comunicación** -la misma donde se encuentran los eventos y mensajes-; proporciona los métodos necesarios para que el control pueda funcionar adecuadamente. Si no se puede obtener un objeto de esta clase el item es eliminado. En el caso de que se obtenga:
 - Se envía el objeto extraído al control del agente por medio de su interfaz **ItfControlAgteReactivo**.
- El control realiza el siguiente ciclo:
 - Cuando recibe a través de su interfaz una petición para procesar la información obtenida por la percepción (*procesarInfoControlAgteReactivo*)
 - Extrae del objeto *InfoControlAgteReactivo* la información correspondiente al input y a la lista de objetos que van a ser utilizados como parámetros de la acción en el autómata
 - Envía la información al autómata para que transite

- Cuando la transición termina se devuelve el control al procesador de items de la percepción que puede continuar su ciclo procesando – en caso en que exista - un nuevo item.
- **El autómata implementa la máquina de Estados Finitos Extendida:**
 - Interpreta la tabla de estados en XML y las acciones definidas en el comportamiento del agente
 - Recibe como entrada la información extraída de eventos o mensajes : (un input y una lista de objetos)
 - Comprueba si con la información obtenida y en el estado en que se encuentra es posible ejecutar alguna de las transiciones definidas en la **tabla de estados**
 - Se consulta la tabla de estados para obtener una transición correspondiente al input recibido y al estado actual del autómata (T(estadoActual, Input recibido)).
 - Si existe la transición
 - Se obtiene la acción a ejecutar que debe corresponderse con un método de la clase Acciones semánticas del agente.
 - Se construye la invocación al método de la clase Acciones semánticas utilizando como parámetros la lista de objetos extraída del contenido del mensaje o evento.
 - Se cambia el estado del autómata al estado definido en la transición
 - **Se ejecuta el método con los parámetros**
 - Cuando la ejecución de la acción termina el thread de la percepción que procesa los items almacenados en la cola recupera el control y continúa procesando nuevos items.

Del funcionamiento del agente reactivo puede destacarse:

- Para que **el agente reaccione ante los eventos o mensajes enviados** es necesario **que contengan información interpretable por el procesador de items** (implementado por la clase **ProcesadorItemsPercepReactivo**). **El procesador debe poder extraer del objeto que se le envía los inputs** para que el autómata ejecute alguna acción y cambie de estado, los **parámetros** para poder ejecutar las acciones.
- En la fase de diseño es necesario establecer una correspondencia ente **la información enviada en los eventos** con los **estados y acciones del autómata que define el comportamiento del agente**. La clase **InfoContEvtMsgAgteReactivo** ha sido concebida con este propósito.
- Para enviar información a otros agentes se requiere:
 - Crear un evento o un mensaje **con información interpretable** por los posibles receptores
 - **Enviar el evento o mensaje al receptor por medio de su interfaz de uso.**

4.1.5.4 Sincronización del comportamiento de agentes y recursos: el modelo de información común.

Vamos a seguir con un ejemplo del sistema de acceso la sincronización entre el comportamiento de los emisores y de los agentes receptores. En el diagrama de comunicación de la Figura 3-19 se han identificado los siguientes mensajes:

Emisor	Mensaje	Receptor
RecursoVisualizacionAcceso	1.3:aceptaEvento("autenticacion",infoUsuario)	AgenteAplicacionAcceso
AgenteAplicacionAcceso	1.6: aceptaEvento("peticionTerminacion")	GestorAgentes

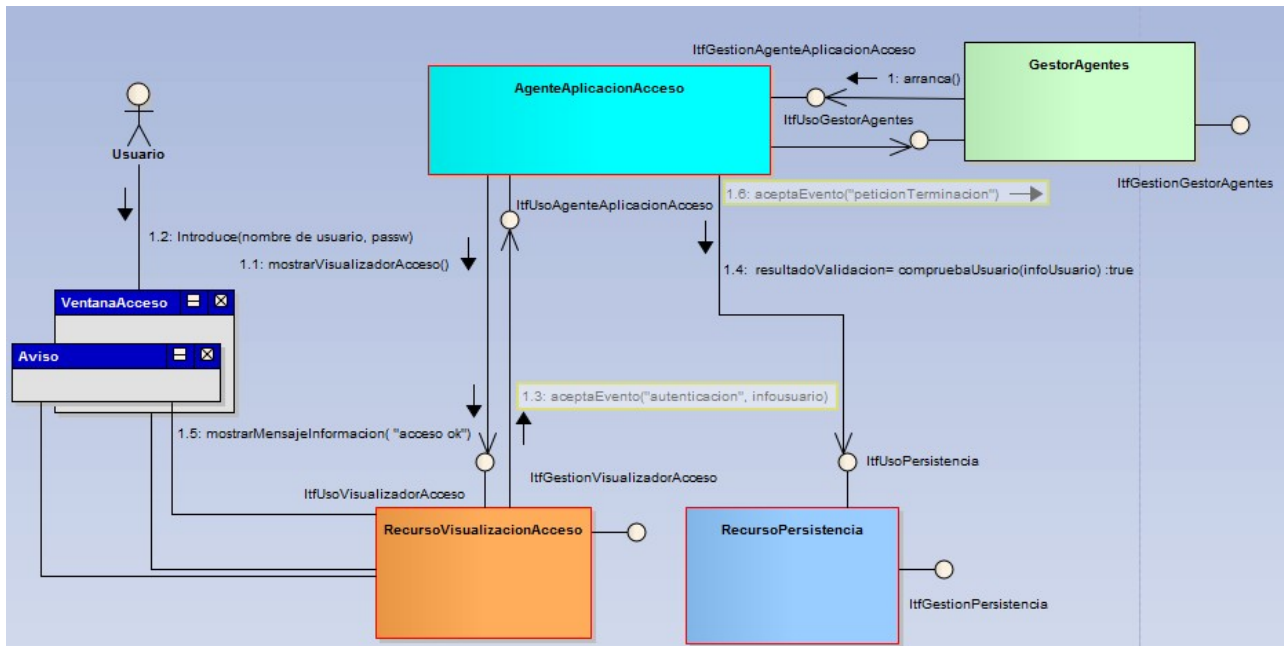


Figura 3-18 Envío de eventos entre entidades

La notación utilizada en el diagrama indica que el **RecursoVisualizacionAcceso**:

- Debe crear un evento cuyo contenido tenga la siguiente información:
 - **msg** = “autenticación”
 - **msgElements** = *infoUsuario* (donde *infoUsuario* es un objeto que contiene el *usrn* y el *pasw* introducidos por el usuario).
- Utiliza la interfaz de uso del agente para enviarle el evento creado

La implementación en Java del envío del mensaje es prácticamente directa utilizando la clase **EventoRecAgente**:

```
// Creación del evento para enviarlo al agente. La informacion del evento es la siguiente
// msg= "autenticacion"; msgElement= InfoAutenticacion ;
// origen = "VisualizacionAccesoI" ( Nombre del recurso de visualizacion) y destino = nombreAgenteAcceso
itfUsoAgente.aceptaEvento(new EventoRecAgte("autenticacion", InfoAutenticacion, nombredeEsteRecurso, nombreAgenteAcceso));
```

Esta instrucción esta:

en el método: **public void peticionAutenticacion(String username, String password)**
 de la clase **NotificacionesAgenteAcceso**
 en el paquete **icaro.aplicaciones.reursos.visualizacionAcceso.imp;**

Siguiendo el diagrama de comunicación podemos conocer el estado en que se encuentra el agente cuando recibe el evento del Recurso de Visualización.

El gestor de Agentes crea el Agente de acceso y le da la orden de arranque a través de su interfaz de gestión. Como resultado de la orden el agente ejecuta la acción “**arranque**” y transita al estado **esperaLogin**.

El código Java de la acción **arranque** se encuentra en el método **arranque** de la clase **AccionesSemanticasAgenteReactivoAcceso**.

La implementación de la orden que le da el agente al visualizador para que saque la ventana de acceso, esta en la instrucción:

```

public void arranque() {
    try {
        visualizacion = (ItfUsoVisualizadorAcceso) itfUsoRepositorio.obtenerInterfaz
        (NombresPredefinidos.ITF_USO+"VisualizacionAcceso1");
        visualizacion.mostrarVisualizadorAcceso(this.nombreAgente, NombresPredefinidos.TIPO_REACTIVO);
        trazas.aceptaNuevaTraza(new InfoTraza(this.nombreAgente,"Se acaba de mostrar el visualizador", InfoTraza.NivelTraza.debug));
    }
}

```

visualizacion.mostrarVisualizadorAcceso(this.nombreAgente,NombresPredefinidos.TIPO_REACTIVO);

Se pasan como parámetros al visualizador **el nombre y el tipo del agente que da la orden**. De esta forma el visualizador conoce a quien le tiene que mandar los eventos con la información generada por el usuario.

La clase **AccionesSemanticasAgenteReactivoAcceso** junto con el autómata define el comportamiento del Agente Reactivo de acceso, por tanto **se encuentra en el paquete : *icaro.aplicaciones.agentes.comportamiento***.

La figura siguiente ilustra el procesamiento del evento por el agente y la ejecución de la **acción valida** que tiene como consecuencia la petición al recurso de persistencia de la validación de la información introducida por el usuario.

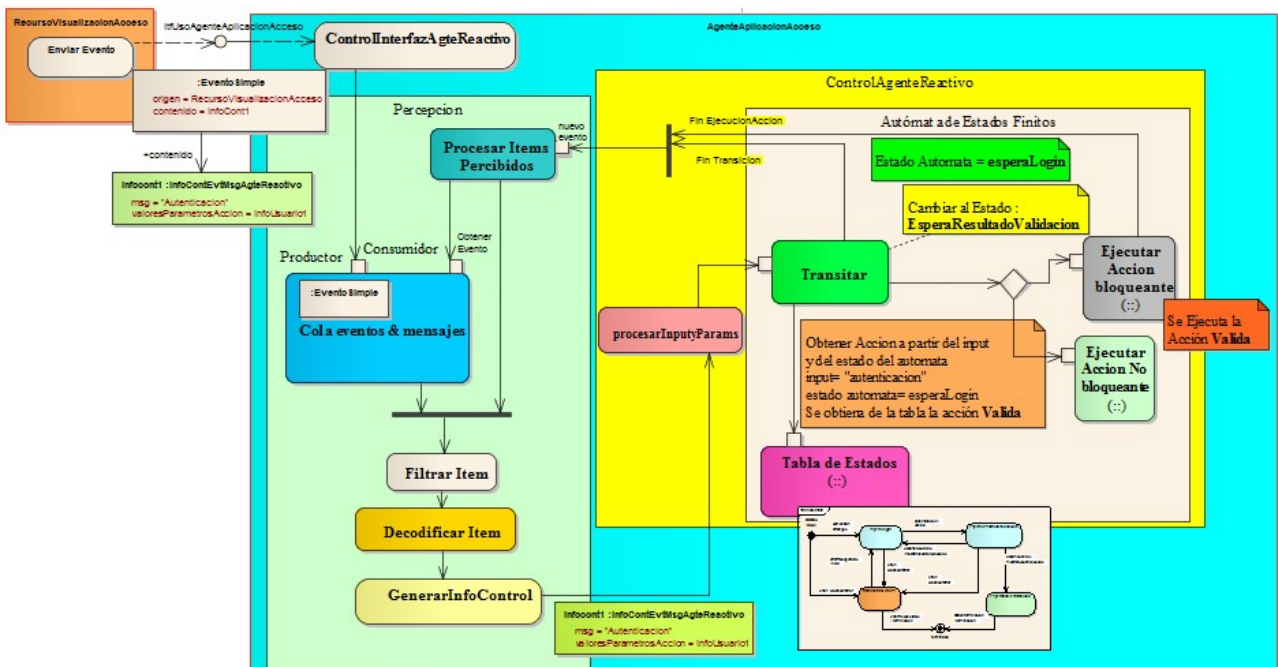


Figura 3-19 Envío y procesamiento de un evento por el agente reactivo

La **acción valida** se encuentra implementada en el método *valida* de la clase **AccionesSemanticasAgenteReactivoAcceso**.

```

public void valida(InfoAccesoSinValidar infoUsuario) {
    boolean ok = false;
    try {
        Persistencial = (ItfUsoPersistenciaAccesoSimple) itfUsoRepositorio.obtenerInterfaz
        (NombresPredefinidos.ITF_USO+"Persistencial");
        ok = Persistencial.compruebaUsuario(infoUsuario.tomaUsuario(), infoUsuario.tomaPassword());
    }
}

```

El control del agente transforma las acciones de la tabla de estados en invocaciones a los métodos de la clase que implementa las acciones semánticas. Esta transformación no tiene problemas cuando los métodos no tienen parámetros, pero cuando los tiene como es el caso de la acción valida, esta información se pasa en el atributo **msgElements** por medio de una lista de objetos.

En el ejemplo de la figura lo que se necesita validar es la información introducida por el usuario (usn y pasw).

- El *RecursoVisualizacionAcceso* **obtiene la información**, crea un objeto de la **clase InfoAccesoSinValidar** con los valores del usn y el pasw , y se lo envía al agente asociándolo al atributo **msgElements** del evento.
- El agente extrae este objeto del atributo **msgElements** del evento y lo utiliza para construir la invocación al **método valida** de su clase de acciones semánticas.
- En este método se usa el contenido del objeto *infoUsuario* para invocar la operación *compruebaUsuario* de la interfaz de uso del recurso de persistencia, que es quien va a realizar la validación propiamente dicha utilizando la información almacenada en la base de datos. Esto se hace en la instrucción siguiente que devuelve un booleano indicando si son o no válidos los datos:

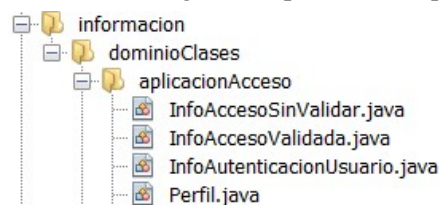
```
ok = PersistenciaI.compruebaUsuario(infoUsuario.tomaUsuario(),infoUsuario.tomaPassword());
```

Agentes y recursos son capaces de intercambiar información por medio de eventos, pero para que haya “**entendimiento**” entre ambos es necesario **que tengan un modelo común de información** (en el ejemplo la clase **InfoAccesoSinValidar**). **Esto permite, (aunque no garantiza) que:**

- los emisores generen eventos con información interpretable por posibles receptores
- los receptores sean capaces a su vez de extraer, interpretar y utilizar la información recibida.

El modelo común de información esta formado por clases que representan las entidades computacionales de la aplicación y son utilizadas por agentes y recursos como base común de procesamiento y de comunicación de información.

En ICARO estas clases se agrupan en el directorio *icaro\aplicaciones\informacion\dominioClases* que admite una clasificación en subdirectorios según las aplicaciones que se desarrollen.



Las clases de dominio utilizadas para el ejemplo se encuentran en la carpeta *aplicacionAcceso*. De estas clases sólo se utiliza en el ejemplo *InfoAccesoSinValidar*.

La segunda clase puede ser útil para distinguir la información validada de la que no lo es, y la tercera procede de una situación común a muchas aplicaciones consistente en obtener un perfil del usuario a partir de los datos de acceso. En este caso el agente puede crear **un objeto perfil** a partir de los datos obtenidos del recurso de persistencia, o puede recibir el perfil del usuario como respuesta a la petición de autenticación. El agente interpreta la información del perfil del usuario utilizándola para:

- a) delegar el proceso de interacción con el usuario en otro agente.
- b) seleccionar un visualizador adecuado al perfil del usuario.
- c) ordenar a un visualizador genérico el tipo de interfaz que tiene que presentar al usuario.

4.1.6 Resumen de Puntos a tener en cuenta para la implementación de la comunicación entre agentes y recursos.

- La comunicación entre agentes es asíncrona por medio de mensajes
- La comunicación entre agentes y recursos puede ser síncrona (operacional) o asíncrona (por medio de comandos donde el resultado del comando se recibe de forma asíncrona)
- La comunicación entre recursos y agentes es asíncrona por medio de eventos

ICARO proporciona un modelo de eventos y de mensajes que puede ser extendido para el desarrollo de nuevas aplicaciones o procesos de comunicación.

La entidad emisora de la información (agente o recurso) debe realizar las siguientes acciones:

- **Obtener la interfaz de uso de la entidad a la que debe mandar la información** (En el ejemplo el recurso de visualización debe obtener la interfaz de uso del agente de Acceso para mandarle los eventos con la información introducida por el usuario)
- **Crear un evento o un mensaje con la información que debe enviar.** En el ejemplo precedente se utiliza la clase EventoRecAgente donde se proporcionan atributos y operaciones para definir :
 - **La información que el control del agente interpretará como un input de su autómata.** (atributo `msg` en la clase EventoRecAgente) .
 - **La lista de objetos** que el control del agente pueda convertir en parámetros para ejecutar las acciones definidas en la tabla de estados del autómata (atributo `msgElements` de la clase EventoRecAgente) .
 - **Los objetos que se pasan en el atributo msgElements pertenecen a clases que deben estar definidas en el modelo de información común.**
- **Enviar la información utilizando el método `aceptaEvento`** de la interfaz de uso (*`ifUsoAgente.aceptaEvento`*).

La entidad receptora de los eventos o mensajes (agentes).

- Puede recibir eventos o mensajes concurrentemente de distintos emisores.
- Cuando recibe un evento o mensaje se envía a la percepción que lo almacena en una cola para su tratamiento .
- El procesamiento de los eventos almacenados es independiente de su almacenamiento.
- Cuando se decide procesar un item de la cola , se ejecutan las siguientes acciones:
 - Se extrae el contenido del evento o mensaje y de dicho contenido se extrae un input y una colección de objetos
 - Se pasa la información obtenida al autómata para que transite. La ejecución de una transición dependerá del estado del agente, del input obtenido y de la información de la tabla de estados. Si se encuentra en la tabla una transición que corresponda al input obtenido en el estado actual del agente se obtiene la acción de la tabla y se procede a ejecutarla.
 - La ejecución de una acción obtenida de la tabla de estados consiste en **ejecutar el método de la clase `AccionesSemnaticasAgenteReactivo<identificadorAgente>` que tiene el mismo nombre que la acción definida en la tabla de estados**, utilizando como parámetros los objetos obtenidos en el atributo `msgElements`.

La comunicación síncrona entre agentes y recursos se realiza por medio de las operaciones definidas en las interfaces de uso de las entidades.

- Los agentes son clientes de las interfaces de uso de los recursos
- Utilizan las operaciones ofrecidas por los recursos para dar ordenes y para hacer peticiones de procesamiento de información. Las respuestas pueden recibirlas de forma síncrona (en el ejemplo la operación valida del recurso de persistencia) o asíncrona a través de un evento.
- No se recomienda que los recursos sean clientes de otros recursos. Un recurso se concibe como una entidad que lleva a cabo sus tareas sin depender de otros elementos de la organización. Esto facilita su distribución en distintos nodos y su evolución siempre que conserve sus interfaces.

4.1.6.1 Generación de eventos internos por parte de los agentes reactivos

En los ejemplos precedentes se ha visto con detalle el proceso de creación , envío, recepción e interpretación de los eventos. Los agentes ejecutan acciones como resultados del procesamiento de los eventos que reciben del entorno, pero pueden además generar eventos propios que les hagan cambiar de estado y ejecutar nuevas acciones.

Esta situación se presenta cuando el agente utiliza los servicios de un recurso a través de su interfaz de uso. Un ejemplo de esta situación lo tenemos en el sistema de acceso cuando el agente recibe los datos de acceso y ejecuta la acción **valida** y transita al estado **esperaResultadoValidacion**.

La **acción valida** consiste en **pedir la validación al recurso de persistencia** mediante la operación *`compruebaUsuario`* de su interfaz de uso.

```
( instrucción : ok =
Persistencia1.compruebaUsuario(infoUsuario.tomaUsuario(),infoUsuario.tomaPassword()); del método
valida )
```

La respuesta del recurso de persistencia esta codificada como un booleano que indica si los datos del usuario son válidos o no. Este resultado se obtiene dentro del método **valida**, pero es necesario transmitirlo al control del agente (al autómata) para que se ejecute la acción y el cambio de estado correspondiente a haber obtenido el resultados de la validación. Para ello se debe crear un **eventoRecAgente** y enviárselo a sí mismo utilizando su propia interfaz de uso. El código java es el siguiente:

```
Object[] datosEnvio = new Object[] {infoUsuario.tomaUsuario(), infoUsuario.tomaPassword()};
if(ok){agenteAcceso.aceptaEvento(new
    EventoRecAgte("usuarioValido",datosEnvio,this.nombreAgente,NombresPredefinidos.NOMBRE_AGENTE_APLICACION+"Acceso"));}
else agenteAcceso.aceptaEvento(new EventoRecAgte("usuarioNoValido", datosEnvio,this.nombreAgente,this.nombreAgente));
```

Si el método valida no genera este evento, el agente se quedará esperándolo en el estado **esperaResultadoValidacion**.

La clase base AccionesSemanticasAgente reactivo proporciona un método para que el agente se envíe eventos a sí mismo :

```
public void informaraMiAutomata(String input, Object[] infoComplementaria)
```

Este método también esta disponible en la clase *AccionesSemanticasAgenteAplicacionAcceso*. Por tanto una implementación equivalente es :

```
if (ok) {informaraMiAutomata("usuarioValido",datosEnvio);}
else informaraMiAutomata("usuarioValido",datosEnvio);
```

Otra opción también válida es que el recurso de persistencia genere ese evento como resultado de la petición de validación. De esta forma la acción semántica se encarga sólo de dar la orden y se delega en el recurso el envío asíncrono del resultado de la validación. Esta modificación puede hacerse como ejercicio.

4.2 Definición de la organización que implementa el sistema de acceso

La organización que implementa una aplicación se define con un documento formal –expresado en XML– donde se especifican las propiedades globales y los componentes que van a implementar la funcionalidad. Para cada componente se definen, los roles, atributos, características y nodos donde deben ser desplegados. **Este documento se utiliza para verificar las características definidas y para crear los componentes computacionales que deben llevar a cabo la funcionalidad del sistema.**

La descripción de la organización del ejemplo de acceso se encuentra en el fichero: *descripcionAplicacionAcceso.xml*, situado en: “\config\icaro\aplicaciones\descripcionOrganizaciones”

La Figura 3-21 presenta el documento que describe la organización del sistema de acceso.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<icar:DescOrganizacion xmlns:icar="urn:icar:aplicaciones:descripcionOrganizaciones"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:icar:aplicaciones:descripcionOrganizaciones ../../schemas/DescripcionOrganizacionSchema.xsd">
  <!--*****
  Propiedades globales de la organización
  *****-->
  <icar:PropiedadesGlobales>
    <icar:intervaloMonitorizacionGestores>50000</icar:intervaloMonitorizacionGestores>
    <icar:activarPanelTrazasDebug>true</icar:activarPanelTrazasDebug>
  </icar:PropiedadesGlobales>
  <icar:DescripcionComponentes>
    <icar:DescComportamientoAgentes>
      <!--*****
      Descripción del comportamiento de los gestores
      *****-->
      <icar:DescComportamientoGestores>
        <icar:DescComportamientoAgente
          nombreComportamiento="GestorOrganizacion" rol="Gestor" tipo="Reactivo" />
        <icar:DescComportamientoAgente
          nombreComportamiento="GestorAgentes" rol="Gestor" tipo="Reactivo" />
        <icar:DescComportamientoAgente
          nombreComportamiento="GestorRecursos" rol="Gestor" tipo="Reactivo" />
      </icar:DescComportamientoGestores>
      <!--*****
      Descripción del comportamiento de los agentes de aplicación
      *****-->
      <icar:DescComportamientoAgentesAplicacion>
        <icar:DescComportamientoAgente
          nombreComportamiento="AgenteAplicacionAcceso" rol="AgenteAplicacion"
          localizacionComportamiento="icar.aplicaciones.agentes.agenteAplicacionAccesoReactivo.comportamiento"
          tipo="Reactivo" />
        </icar:DescComportamientoAgentesAplicacion>
      </icar:DescComportamientoAgentes>
      <!--*****
      Descripción de los recursos de aplicación
      *****-->
      <icar:DescRecursosAplicacion>
        <icar:DescRecursoAplicacion nombre="PersistenciaAccesoSimple"
          localizacionClaseGeneradora="icar.aplicaciones.recursos.persistenciaAccesoSimple.imp.ClaseGeneradoraPersistenciaAccesoSimple"/>
        <icar:DescRecursoAplicacion nombre="VisualizacionAcceso"/>
      </icar:DescRecursosAplicacion>
    </icar:DescripcionComponentes>
  </icar:DescInstancias>

```

Figura 3-20 Descripción de los componentes de la Organización

En síntesis lo que expresa es lo siguiente:

- La organización tiene una serie de **propiedades computacionales globales** aplicables al conjunto de sus componentes. Las que se especifican en el documento se refieren **al intervalo de monitorización de los gestores**, a la **activación o no de las trazas** para comprobar el comportamiento de los componentes. Pueden añadirse nuevas propiedades siguiendo el formato atributo valor.
- A continuación se describen **los Componentes de la organización**. Se distinguen tres tipos de componente: Gestores de la Organización. Agentes de aplicación y recursos de aplicación.
 - **La organización tiene tres gestores:** El Gestor de la Organización, el Gestor de Agentes y el Gestor de Recursos. Los gestores son agentes cuyo rol en la organización esta predefinido, es decir deben dedicarse a gestionar otras entidades.
 - Para cada agente es necesario definir su modelo de comportamiento, que en el caso de los gestores se especifica con tres parámetros:
 - El **nombre del comportamiento**.
 - El **rol del agente**. Indicará que es un gestor
 - El **tipo de agente**. Indica el patrón para generar el agente en este caso es un agente reactivo

En el documento se define el gestor de organización de la forma:

```

<icar:DescComportamientoGestores>
  <icar:DescComportamientoAgente
    NombreComportamiento="GestorOrganizacion" rol="Gestor" tipo="Reactivo" />
  </icar:DescComportamientoAgente>

```

Los agentes de aplicación se describen de forma similar a los agentes gestores. ICARO permite asociar varios comportamientos al mismo agente de manera que el desarrollador pueda experimentar con cada comportamiento del agente y elegir el más adecuado. La definición del comportamiento que se utilizará en la organización se hace indicando la ruta (path) donde se encuentran las clases y ficheros que permitirá

generar la instancia del agente a partir del patrón: En el caso de un agente reactivo es la ruta donde se encuentra la tabla de estados y las acciones semánticas por ejemplo:

```
<icaro:DescComportamientoAgentesAplicacion>
```

```
<icaro:DescComportamientoAgente
```

```
  nombreComportamiento="AgenteAplicacionAcceso"rol="AgenteAplicacion" tipo="Reactivo"
```

```
  localizacionComportamiento="icaro.aplicaciones.agentes.agenteAplicacionAccesoReactivo.comportamiento1"/>
```

```
<icaro:DescComportamientoAgente
```

Si no se especifica la localización del comportamiento se asume por defecto que se encontrará en la ruta:

"icaro.aplicaciones.agentes.agenteAplicacion<IdentificadorAgente>Reactivo.

Los recursos se describen especificando el identificador del recurso y la localización de la clase que permite la creación del recurso. Si no se especifica la localización se buscará en la ruta por defecto

"icaro.aplicaciones.agentes.recursosAplicacion"

Ejemplo:

```
<icaro:DescRecursosAplicacion>
```

```
  nombre="Persistencia" localizacionClaseGeneradora="icaro.aplicaciones.recursos.persistencia.imp.PersistenciaImp"/>
```

```
  nombre="VisualizacionAcceso"/>
```

```
<icaro:DescRecursosAplicacion>
```

Una vez definidos los componentes se especifican los - ejemplares de agentes y recursos - que implementarán la funcionalidad de la aplicación.

4.2.1 Especificación del modelo dinámico de la organización.

La segunda parte del documento define los componentes dinámicos Figura 3-22. Debemos tener en cuenta que a partir de un componente estático pueden generarse múltiples ejemplares distribuidos en diferentes nodos. Por ejemplo a partir del *agenteAplicacionAcceso* del que hemos especificado su comportamiento, pueden generarse múltiples ejemplares (o instancias clónicas) en un determinado nodo (PC) o en nodos diferentes.

```

<!--*****
*****      Instancias de gestores      *****
*****-->
<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="Persistencial" tipoComponente="RecursoAplicacion"/>
      <icaro:componenteGestionado refId="VisualizacionAcceso1" tipoComponente="RecursoAplicacion"/>
    </icaro:componentesGestionados>
  </icaro:InstanciaGestor>
</icaro:Gestores>
<!--*****
*****      Instancias de Agentes de aplicación      *****
*****-->
<icaro:AgentesAplicacion>
  <icaro:Instancia id="AgenteAplicacionAcceso1" refDescripcion="AgenteAplicacionAcceso">
    <icaro:listaPropiedades>
      <icaro:propiedad atributo=" " valor=""/>
    </icaro:listaPropiedades>
  </icaro:Instancia>
</icaro:AgentesAplicacion>
<!--*****
*****      Instancias de Recursos de aplicación      *****
*****-->
<icaro:RecursosAplicacion>
  <icaro:Instancia id="Persistencial" refDescripcion="PersistenciaAccesoSimple" xsi:type="icaro:Instancia">
    <icaro:listaPropiedades>
      <icaro:propiedad atributo="MYSQL_NAME_BD" valor="bbddejemplo" />
      <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/" />
    </icaro:listaPropiedades>
  </icaro:Instancia>
  <icaro:Instancia id="VisualizacionAcceso1" refDescripcion="VisualizacionAcceso" xsi:type="icaro:Instancia"/>
</icaro:RecursosAplicacion>
<!--*****
*****      Descripción común del nodo en el que se despliegan las instancias      *****
*****-->
<icaro:nodoComun>
  <icaro:nombreUso>NodoPrincipal</icaro:nombreUso>
  <icaro:nombreCompletoHost>localhost</icaro:nombreCompletoHost>
</icaro:nodoComun>
</icaro:DescInstancias>

```

Figura 3-21 Descripción del modelo dinámico de la Organización

Definición de ejemplares (instancias) de Gestores

En el documento se especifican en primer lugar los ejemplares de gestores que deben ser generados al crear la organización y para cada gestor los ejemplares de entidades gestionadas. Ejemplo:

```

<icaro:InstanciaGestor id="GestorOrganizacion" refDescripcion="GestorOrganizacion">
  <icaro:componentesGestionados>
    <icaro:componenteGestionado refId="GestorAgentes" tipoComponente="Gestor"/>
    <icaro:componenteGestionado refId="GestorRecursos" tipoComponente="Gestor" />
  </icaro:componentesGestionados>

```

En estas líneas se expresa que cuando se cree la organización, deberá generarse un ejemplar de *GestorOrganizacion* cuya descripción se llama también *GestorOrganizacion*. Este gestor debe gestionar (es decir crear, monitorizar y controlar) los siguientes componentes: un *GestorAgentes* que es de tipo *Gestor* y un *GestorRecursos* que también es un gestor.

La especificación de las instancias correspondientes al gestor de agentes y al gestor de recursos se hace de forma similar:

El gestor de agentes tiene que crear y gestionar los agentes de la organización; en esta organización deberá crear un ejemplar de agente que se llamará *AgenteAplicacionAcceso1* a partir de la descripción de comportamiento identificada como *AgenteAplicacionAcceso* que se hizo en la primera parte del documento.

```
<icar:InstanciaGestor id="GestorAgentes" refDescripcion="GestorAgentes">
    <icar:componentesGestionados>
        <icar:componenteGestionado refId="AgenteAplicacionAcceso1"
            tipoComponente="AgenteAplicacion"/>
    </icar:componentesGestionados>
```

Definición de ejemplares (instancias) de Agentes de Aplicación.

La instancia *AgenteAplicacionAcceso1* se define en el apartado de definición de instancias de agentes de aplicación.

```
<icar:AgentesAplicacion>
    <icar:Instancia id="AgenteAplicacionAcceso1" refDescripcion="AgenteAplicacionAcceso">
        <icar:listaPropiedades>
            <icar:propiedad atributo="" valor="" />
        </icar:listaPropiedades>
    </icar:Instancia>
</icar:AgentesAplicacion>
```

Definición de ejemplares (instancias) de Recursos

Para la definición de las instancias de recursos se sigue el mismo proceso utilizando la lista de propiedades para definir las características específicas de cada recurso – por ejemplo con el recurso de persistencia-

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

Definición de los nodos donde deben ejecutarse los agentes y los recursos

En el caso del sistema de acceso el despliegue es en un solo nodo.

```
<icar:nodoComun>
    <icar:nombreUso>NodoPrincipal</icar:nombreUso>
    <icar:nombreCompletoHost>localhost</icar:nombreCompletoHost>
</icar:nodoComun>
</icar:Instancias>
```

En el apartado siguiente se detalla un ejemplo donde los agentes y recursos se ejecutan en nodos diferentes.

4.3 Distribución de Agentes y Recursos en una red de procesadores

ICARO ha sido concebido para facilitar el desarrollo de aplicaciones donde los componentes estén físicamente distribuidos en diferentes nodos procesadores.

- La especificación de los nodos donde se deben ejecutar las instancias de Agentes y Recursos se hace de forma declarativa en el documento de definición de la organización.
- La creación y la activación de los agentes y recursos en los nodos la realizan los gestores de acuerdo con las especificaciones del documento de definición de la organización.
- La comunicación entre agentes y recursos es transparente a su localización física. Esto implica que la distribución no complica el proceso de implementación de los componentes.

Para ilustrar el proceso de distribución de componentes, consideremos de nuevo el sistema de acceso con los siguientes requisitos de ejecución (modelo de despliegue):

- Los componentes se van a distribuir en dos procesadores (PCs) conectados en una red local. Los identificadores de los procesadores en la red (hostId) son: **PROCESADOR_1** y **PROCESADOR_2**
- En el **PROCESADOR_1** se van a ejecutar los gestores y en el **PROCESADOR_2** se ejecutarán el agente de acceso el recurso de visualización y el recurso de persistencia.

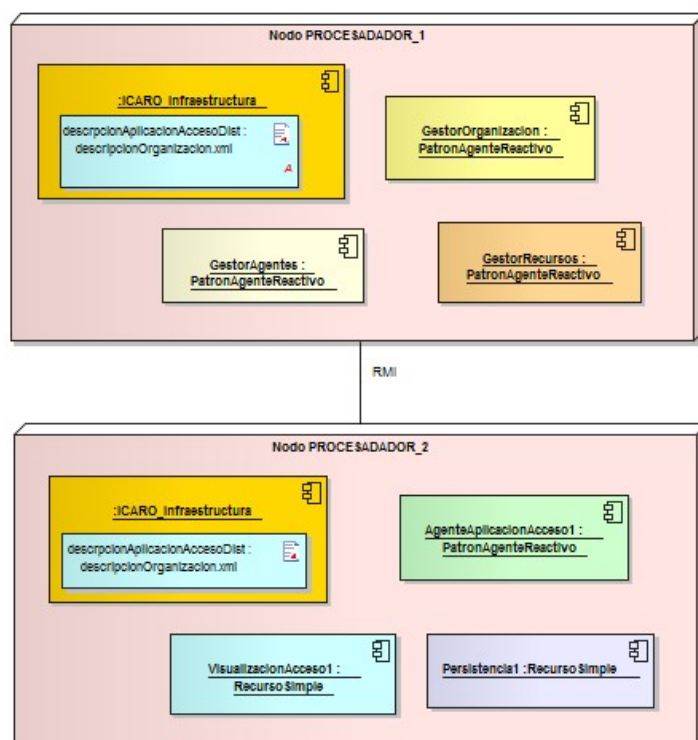


Figura 3-22 Diagrama de despliegue de la Aplicación de Acceso

Para que los agentes y los recursos se creen en diferentes nodos son necesarios los siguientes pasos:

1. Crear un nuevo fichero de descripción de la organización donde se definan los nodos donde deben ser creados los componentes.
2. Verificar que existe conectividad entre los nodos (Realizar un ping desde un nodo hacia el otro y viceversa)
3. Instalar ICARO en cada nodo
4. Ejecutar en cada nodo el script de arranque pasándole como parámetro el fichero de descripción de la organización que se debe crear

NO es necesario cambiar los componentes. Se utilizan las facilidades que ofrece Java con RMI para hacer transparente el proceso de comunicación entre entidades distribuidas. El nuevo fichero de descripción de la organización es el siguiente:

- En el apartado de definición de instancias de los componentes es necesario especificar los identificadores de los nodos donde van a ser creadas y ejecutadas las instancias

```

***** Instancias de Agentes de aplicación *****
*****-->
<icar:AgentesAplicacion>
  <icar:Instancia id="AgenteAplicacionAcceso1" refDescripcion="AgenteAplicacionAcceso">
    <icar:nodoEspecifico>
      <icar:nombreUso>PROCESADOR_2</icar:nombreUso>
      <icar:nombreCompletoHost>172.16.0.5</icar:nombreCompletoHost>
    </icar:nodoEspecifico>
  </icar:Instancia>
</icar:AgentesAplicacion>
<!--*****
***** Instancias de Recursos de aplicación *****
*****-->
<icar:RecursosAplicacion>
  <icar:Instancia id="Persistencial" refDescripcion="PersistenciaAccesoSimple" xsi:type="icar:Instancia">
    <icar:nodoEspecifico>
      <icar:nombreUso>PROCESADOR_2</icar:nombreUso>
      <icar:nombreCompletoHost>172.16.0.5</icar:nombreCompletoHost>
    </icar:nodoEspecifico>
  </icar:Instancia>
  <icar:Instancia id="VisualizacionAcceso1" refDescripcion="VisualizacionAcceso" xsi:type="icar:Instancia"/>
    <icar:nodoEspecifico>
      <icar:nombreUso>PROCESADOR_2</icar:nombreUso>
      <icar:nombreCompletoHost>172.16.0.5</icar:nombreCompletoHost>
    </icar:nodoEspecifico>
</icar:RecursosAplicacion>

```

Figura 3-23 Definición del fichero de la nueva descripción de la organización-1

Puede observarse que en la descripción del nodo se incluye además del hostId y la dirección IP del nodo. Esta dirección IP sirve como identificador alternativo de nodo.

La especificación del nodo donde se ejecutan los gestores se puede indicar de forma específica en la definición de la instancias de los gestores o de forma genérica indicando que será el nodo común y que su nombre es PROCESADOR_1 Figura 3-25

```

***** Descripción común del nodo en el que se despliegan las instancias *****
*****-->
<icar:nodoComun>
  <icar:nombreUso>PROCESADOR_1</icar:nombreUso>
  <icar:nombreCompletoHost>localhost</icar:nombreCompletoHost>
</icar:nodoComun>
</icar:DescInstancias>
</icar:DescOrganizacion>

```

Figura 3-24 Definición del fichero de la nueva descripción de la organización-2

Una vez definido el nuevo fichero de descripción de la organización. La ejecución del script de arranque en cada uno de los nodos debe dar como resultado la aparición de la ventana de trazas en cada nodo con las entidades creadas. La ventana de acceso debe aparecer en el nodo: PROCESADOR_2 que es donde se debe haber creado el agente de acceso, el recurso de visualización y el recurso de persistencia.

Las incidencias que se produzcan en el funcionamiento serán reportadas por el sistema de trazas que se describe a continuación.

Para poder interpretar y corregir los posibles errores es necesario explicar con más detalle los procesos de creación y de funcionamiento de los componentes de ICARO que se describen en el capítulo siguiente.

4.4 Sistema de trazas de la organización.

El recurso de trazas de la organización se creó como soporte para el desarrollo. Permite realizar un seguimiento de trazas a lo largo de la ejecución de las aplicaciones. La estructura interna y la interfaz de uso del recurso puede verse en la Figura 3-23.

La interfaz del recurso de trazas proporciona **diferentes operaciones para los distintos tipos de entidad emisora de la traza y según sea el tipo de traza que se pretenda realizar.**

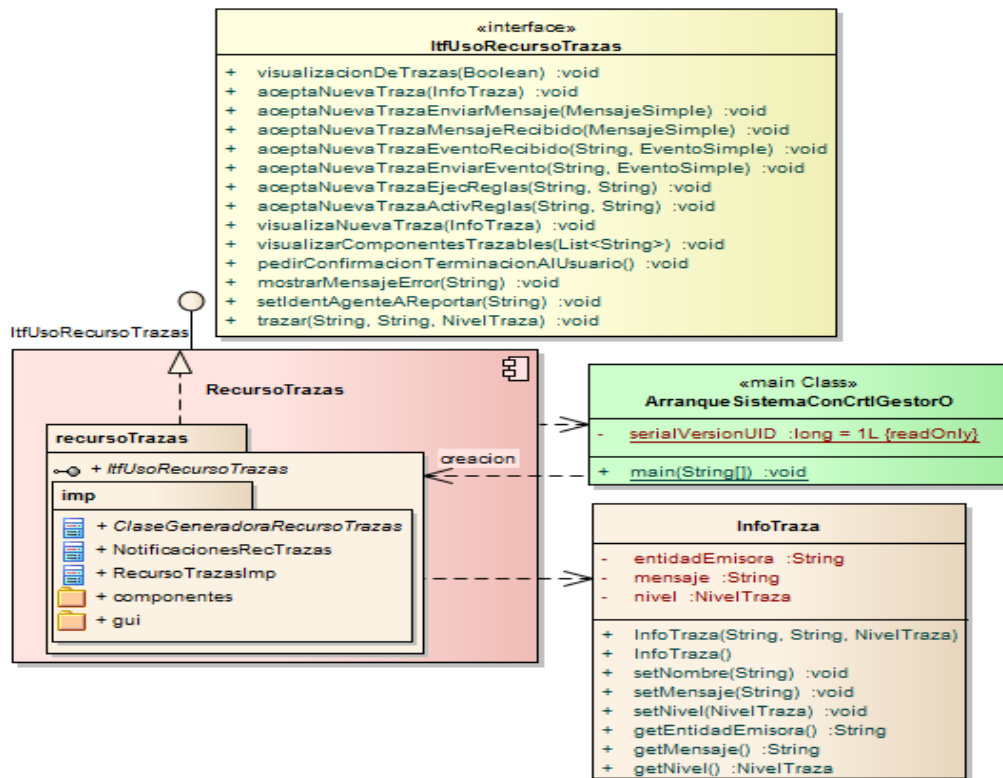


Figura 3-25 Recurso de trazas y la clase InfoTraza

La clase **InfoTraza** proporciona una entidad genérica para definir la información que se pretende trazar utilizando el recurso : entidad emisora de la traza, mensaje de la traza, el tipo de traza (debug, info, error) . Un ejemplo concreto de su utilización se describe a continuación.

4.4.1 Sistema de trazas del ejemplo

Al ejecutar la aplicación con el agente de acceso, se muestran las siguientes ventanas:

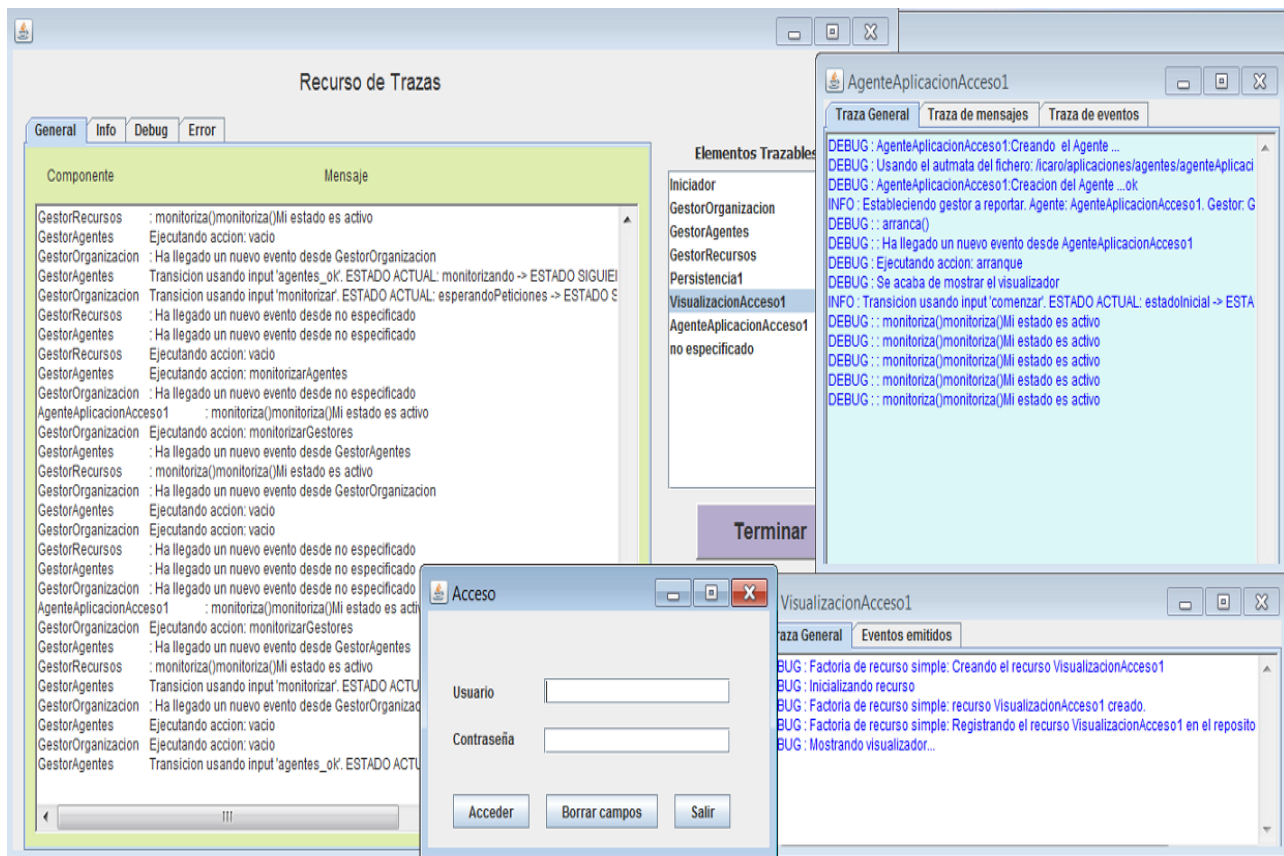


Figura 3-26 Ventanas mostradas al ejecutar la aplicación de acceso

La ventana del recurso de trazas permite visualizar en las pestañas de la ventana principal los tipos de trazas especificados en el código para realizar un seguimiento de la ejecución. En la pestaña ‘General’ aparecen todas las trazas de la ejecución, mientras que en las otras tres aparecen clasificadas según su nivel de traza: info (información), debug (depuración), y error (errores).

En la ventana de Elementos trazables se encuentra la lista de los elementos del sistema que se trazan. Si se pincha sobre cada elemento aparece una ventana específica con sus trazas.

Las ventanas de traza de agentes y recursos tienen un panel general donde se puede mostrar información concreta sobre el comportamiento de la entidad, y paneles para mostrar información específica. En el caso de un agente reactivo hay dos paneles para mostrar, los mensajes emitidos y recibidos y los eventos recibidos. En el caso de los recursos se proporciona un panel para trazar los eventos emitidos.

Las ventanas de agentes y de recursos pueden abrirse, moverse o cerrarse.

Existe también la posibilidad de guardar las trazas en ficheros de log, aunque esta opción está deshabilitada.

4.4.2 Utilización del recurso de trazas

La activación o desactivación de las trazas puede definirse en la descripción de las propiedades de la organización.

```
<icaro:PropiedadesGlobales>
```

```
<icaro:activarPanelTrazasDebug>true</icaro:activarPanelTrazasDebug>
```

En el código la utilización del recurso de trazas puede hacerse desde cualquier objeto o componente, para ello se requiere:

- Obtener la interfaz del recurso de trazas que se encuentra almacenada en el repositorio de interfaces. Esto se hace mediante la siguiente instrucción:

```
try{
```

```
    ItfUsoRecursoTrazas trazas = (ItfUsoRecursoTrazas)itfUsoRepositorio.obtenerInterfaz(
```

```

        NombresPredefinidos.ITF_USO+NombresPredefinidos.RECURSO_TRAZAS);
    }catch (Exception e) {
        System.out.println("No se pudo usar el recurso de trazas");
    }
}

```

La interfaz del recurso de trazas proporciona **diferentes operaciones el tipo de entidad emisora de la traza y el tipo de traza que se pretenda realizar.**

- Una vez obtenida la interfaz de uso recuperada del repositorio de interfaces, para utilizarla es necesario crear un objeto **InfoTraza** especificando:
 - La entidad que genera la traza : **emisor**
 - La información que se pretende trazar : **contenido**
 - El **nivel** de traza : info, debug, error, fatal

```

trazas.aceptaNuevaTraza(new InfoTraza("AgenteAplicacionAcceso",
    "Ha habido un problema en la persistencia al comprobar el usuario y el password",
    InfoTraza.NivelTraza.error));

```

4.5 Problemas y soluciones en la utilización de ICARO

En informática suele ser raro que las cosas funcionen a la primera, sobre todo en sistemas de código abierto desarrollados bajo la iniciativa y el esfuerzo de poca gente que no dispone de mucho tiempo para la documentación del sistema. Los problemas con los sistemas nuevos suelen comenzar en el arranque.

En ICARO el arranque de una aplicación ha sido pensado para poder trazar la creación de los componentes responsables de la funcionalidad de la aplicación y para detectar los posibles fallos, el contexto en el que se produce y sus posibles causas.

El proceso de arranque de una aplicación consta de dos partes:

1. **Creación de la infraestructura de la organización.** Se crean **agentes gestores y recursos comunes a todas las aplicaciones**. Los gestores que tienen la responsabilidad de la creación y de la monitorización del funcionamiento de los agentes y recursos de la aplicación.
2. **Creación comprobación y monitorización de los agentes y recursos propios de la aplicación.**

La figura detalla el proceso de arranque.

4.5.1 Arranque del sistema

Una vez conocido el modelo de la organización – agentes con sus roles, recursos y dependencias- podemos explicar con más detalle el proceso de funcionamiento que comenzará con el arranque de la aplicación. El proceso de arranque crea los componentes de la aplicación de forma progresiva y supervisada a partir de las descripciones del modelo de la organización.

La siguiente figura describe con más detalle el proceso de creación de los componentes de la organización.

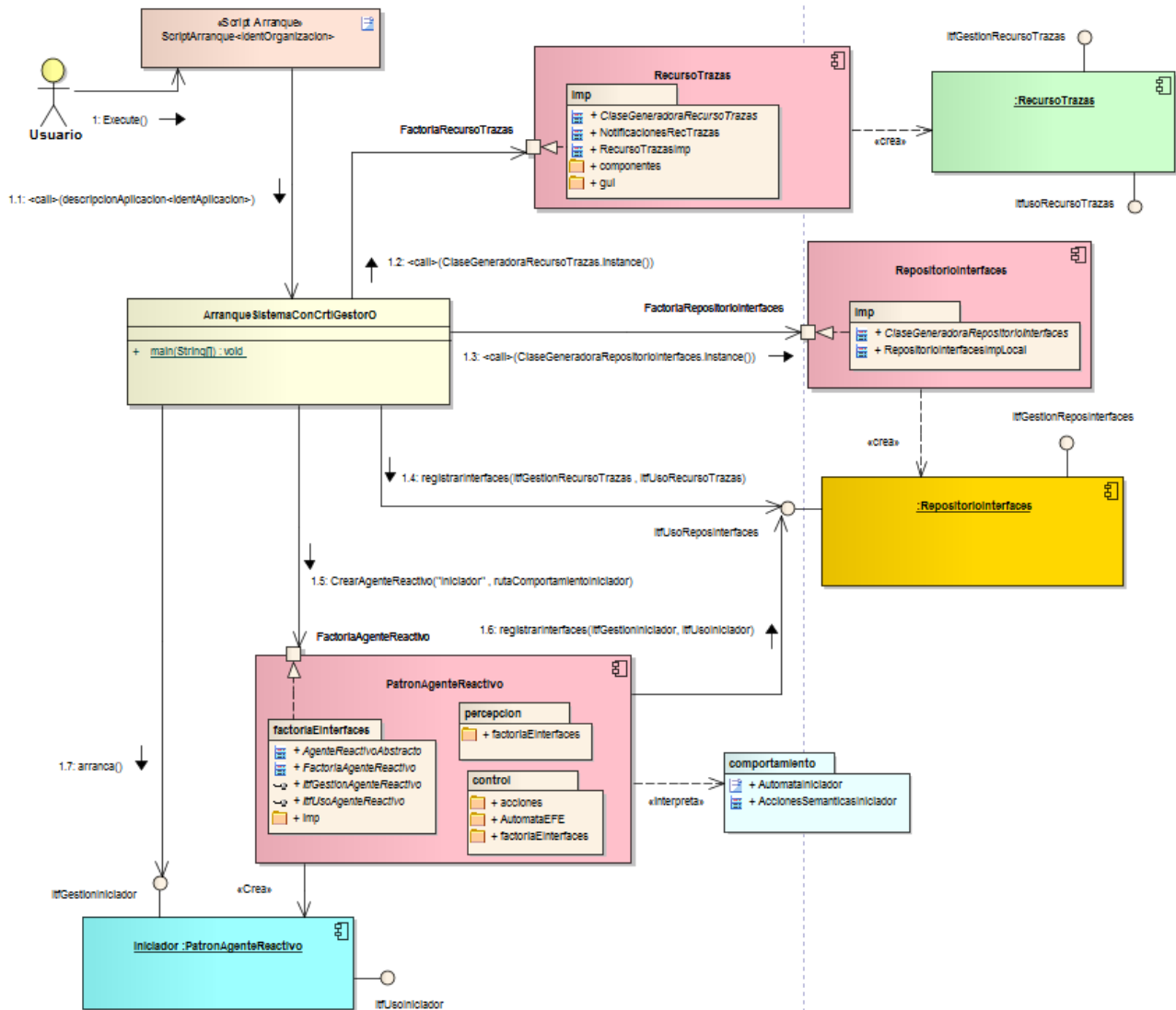


Figura 3-27 Arranque de las aplicaciones

El script de arranque realiza una invocación al main de la clase java ArranqueSistemaConCrtiGestorO, que inicia el proceso de arranque creando los **recursos de la organización**. Se crean en primer lugar instancias del RepositorioInterfaces y del RecursoTrazas. (mensajes 1.2 y 1.3). Estos componentes son necesarios para continuar el proceso de creación del resto de los componentes.

A continuación se crea una instancia de agente iniciador que sera el responsable de validar la descripción de la organización y de crear los agentes y recursos de la aplicación incluidos los gestores. Este agente iniciador captura los errores los visualiza e intenta corregirlos. Cuando termina su cometido desaparece de la organización, dejando paso al Gestor de la Organización.

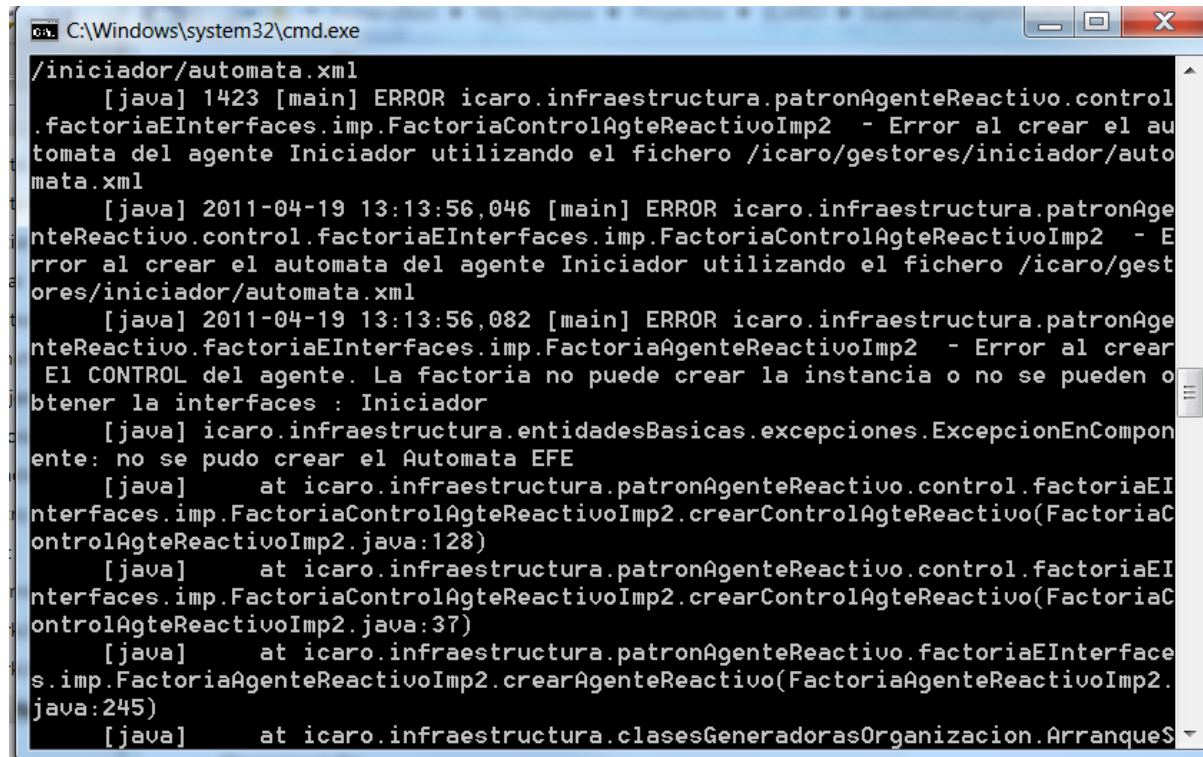
Es poco probable que se produzcan errores en la creación de los recursos de la organización y en el iniciador, (acciones 1, 1.1, 1.2, 1.3, 1.4 y 1.5), pero si se producen es **necesario revisar el contenido de la infraestructura**.

Si se producen fallos en la creación del iniciador, lo más probable es que :

1. La ruta donde se encuentre el comportamiento no se pase adecuadamente, o su contenido haya sido modificado.
2. El patrón no funcione adecuadamente

En ambos casos los errores aparecerán en la pantalla cmd.exe

Un ejemplo de este tipo de error es que el fichero que describe le autómata del iniciador tenga errores sintácticos. En este caso no se visualiza la ventana de trazas o se visualiza de forma inestable. Los errores aparecen en la ventana cmd.exe

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays several lines of Java error messages. The first line is "/iniciador/automata.xml". The second line is "[java] 1423 [main] ERROR icaro.infraestructura.patronAgenteReactivo.control.factoriaEInterfaces.imp.FactoriaControlAgteReactivoImp2 - Error al crear el automata del agente Iniciador utilizando el fichero /icaro/gestores/iniciador/automata.xml". The third line is "[java] 2011-04-19 13:13:56,046 [main] ERROR icaro.infraestructura.patronAgenteReactivo.control.factoriaEInterfaces.imp.FactoriaControlAgteReactivoImp2 - Error al crear el automata del agente Iniciador utilizando el fichero /icaro/gestores/iniciador/automata.xml". The fourth line is "[java] 2011-04-19 13:13:56,082 [main] ERROR icaro.infraestructura.patronAgenteReactivo.control.factoriaEInterfaces.imp.FactoriaControlAgteReactivoImp2 - Error al crear el CONTROL del agente. La factoria no puede crear la instancia o no se pueden obtener la interfaces : Iniciador". The fifth line is "[java] icaro.infraestructura.entidadesBasicas.excepciones.ExcepcionEnComponente: no se pudo crear el Automata EFE". The sixth line is "[java] at icaro.infraestructura.patronAgenteReactivo.control.factoriaEInterfaces.imp.FactoriaControlAgteReactivoImp2.crearControlAgteReactivo(FactoriaControlAgteReactivoImp2.java:128)". The seventh line is "[java] at icaro.infraestructura.patronAgenteReactivo.control.factoriaEInterfaces.imp.FactoriaControlAgteReactivoImp2.crearControlAgteReactivo(FactoriaControlAgteReactivoImp2.java:37)". The eighth line is "[java] at icaro.infraestructura.patronAgenteReactivo.factoriaEInterfaces.imp.FactoriaAgenteReactivoImp2.crearAgenteReactivo(FactoriaAgenteReactivoImp2.java:245)". The ninth line is "[java] at icaro.infraestructura.clasesGeneradorasOrganizacion.ArranqueS".

```
C:\Windows\system32\cmd.exe
/iniciador/automata.xml
[java] 1423 [main] ERROR icaro.infraestructura.patronAgenteReactivo.control
.factoriaEInterfaces.imp.FactoriaControlAgteReactivoImp2 - Error al crear el au
tomata del agente Iniciador utilizando el fichero /icaro/gestores/iniciador/auto
mata.xml
[java] 2011-04-19 13:13:56,046 [main] ERROR icaro.infraestructura.patronAge
nteReactivo.control.factoriaEInterfaces.imp.FactoriaControlAgteReactivoImp2 - E
rror al crear el automata del agente Iniciador utilizando el fichero /icaro/gest
ores/iniciador/automata.xml
[java] 2011-04-19 13:13:56,082 [main] ERROR icaro.infraestructura.patronAge
nteReactivo.factoriaEInterfaces.imp.FactoriaControlAgteReactivoImp2 - Error al crear
El CONTROL del agente. La factoria no puede crear la instancia o no se pueden o
btener la interfaces : Iniciador
[java] icaro.infraestructura.entidadesBasicas.excepciones.ExcepcionEnCompon
ente: no se pudo crear el Automata EFE
[java] at icaro.infraestructura.patronAgenteReactivo.control.factoriaEI
nterfaces.imp.FactoriaControlAgteReactivoImp2.crearControlAgteReactivo(FactoriaC
ontrolAgteReactivoImp2.java:128)
[java] at icaro.infraestructura.patronAgenteReactivo.control.factoriaEI
nterfaces.imp.FactoriaControlAgteReactivoImp2.crearControlAgteReactivo(FactoriaC
ontrolAgteReactivoImp2.java:37)
[java] at icaro.infraestructura.patronAgenteReactivo.factoriaEInterface
s.imp.FactoriaAgenteReactivoImp2.crearAgenteReactivo(FactoriaAgenteReactivoImp2.
java:245)
[java] at icaro.infraestructura.clasesGeneradorasOrganizacion.ArranqueS
```

Figura 3-28 Errores en la interpretación del autómata del iniciador

Los errores más comunes aparecen cuando el iniciador empieza a crear los componentes de la organización que implementa la aplicación definida por el usuario. Las actividades que realiza el iniciador se detallan en la Figura 3-30 y en la Figura 3-31

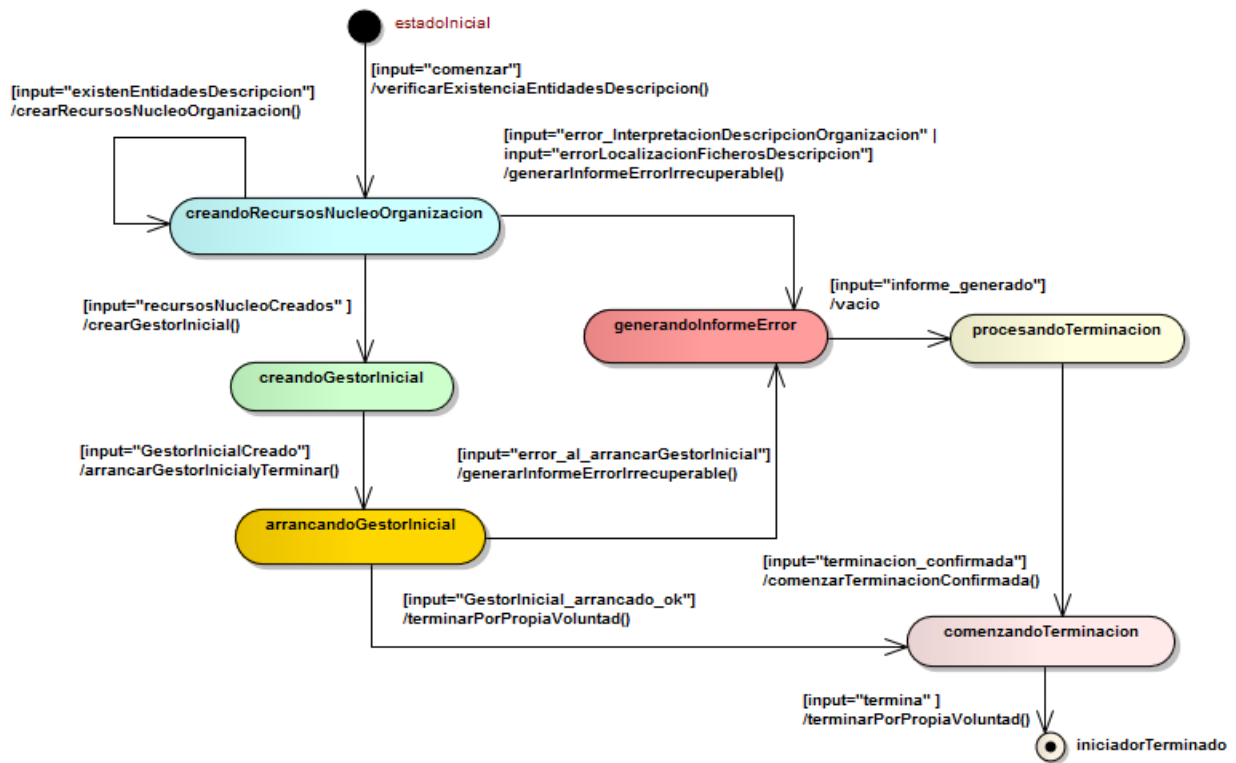


Figura 3-29 Autómata del Iniciador

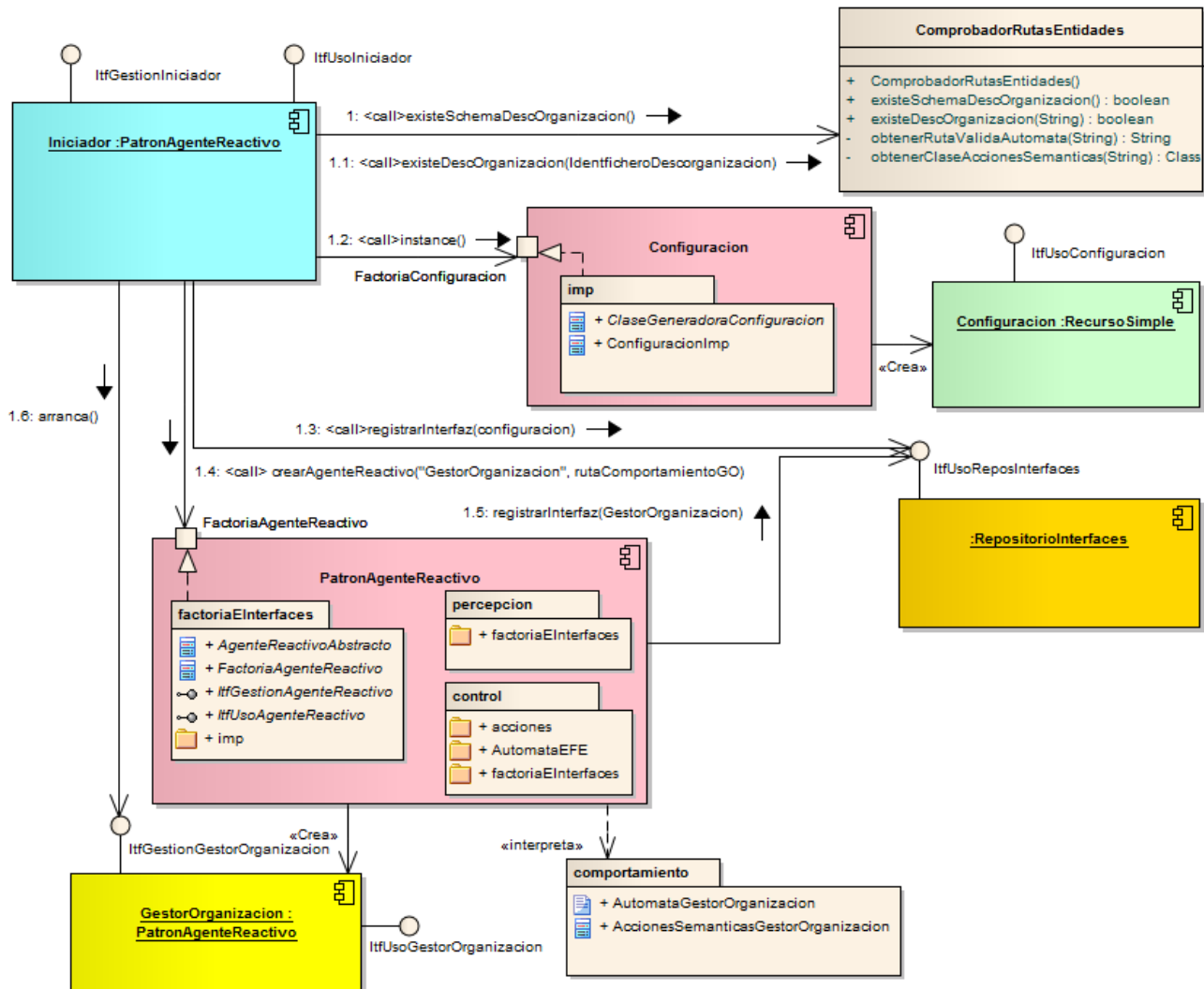
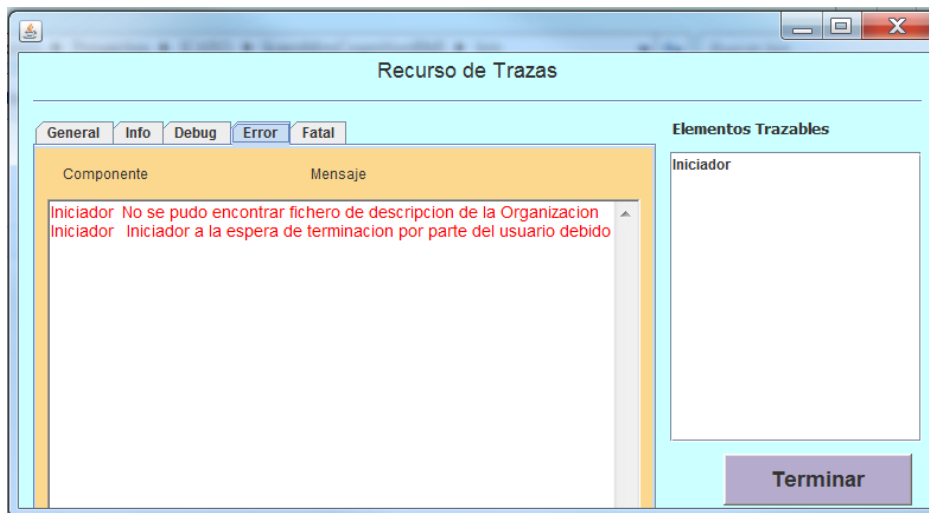


Figura 3-30 Comportamiento del Iniciador

Pueden ocurrir los siguientes fallos que aparecerán en la ventana de trazas.

- En el paso 1 (invocación al método **existeSchemaDescOrganizacion**) el resultado puede ser : que **No se encuentra el fichero de descripción de la organización**. Bien porque no existe o porque el nombre no es el adecuado.
- En el paso 1.1 (invocación al método **existeDescOrganizacion**) el resultado puede ser : que **No se encuentra el fichero XML que contiene la descripción de la organización**

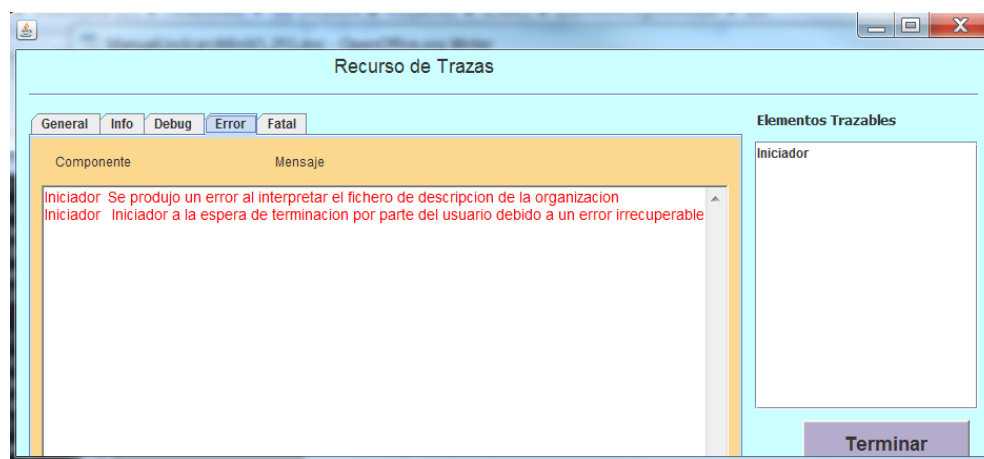
En ambos casos los errores aparecerán en la ventana de trazas.



Solución del error: revisar y corregir el nombre del fichero y/o la ruta donde se encuentra almacenado.

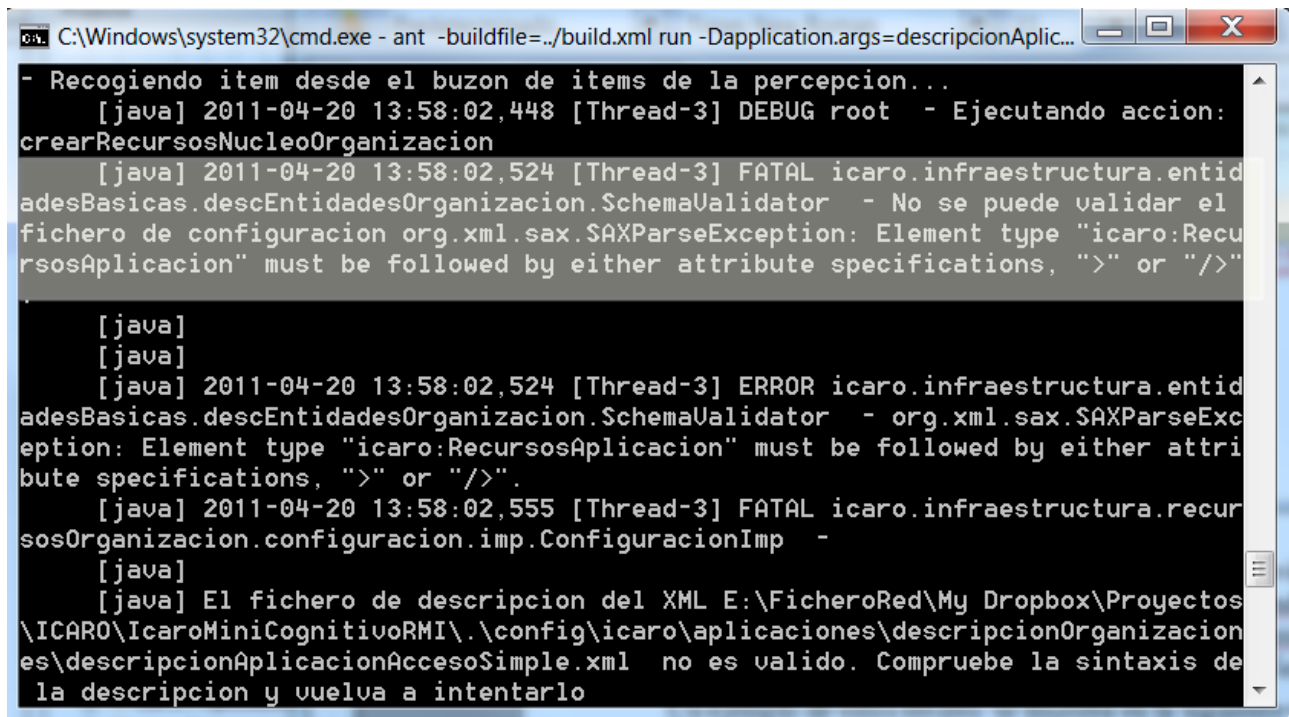
- **En el paso 1.2** Se ordena la creación del recurso de configuración. La creación de este recurso tiene un triple resultado:
 - Interpretar la descripción de la organización y producir una representación interna de su contenido
 - Crear una instancia del recurso de configuración.
 - **Reportar los errores que se produzcan en la Interpretación del fichero de descripción de la organización y en la creación del recurso.**

Todas estas situaciones aparecerán en la ventana de trazas, pero es en la ventana **cmd.exe**. Donde es necesario buscar las causas concretas de los errores



La solución a este tipo de errores consiste en revisar los mensajes específicos del procesador XML (SchemaValidator) que aparecerán en la ventana **cmd.exe** y corregirlos. Es necesario buscar los mensajes de error *ERROR icaro.infraestructura.entidadesBasicas.descEntidadesOrganizacion.SchemaValidator - org.xml.sax.SAXParseException que indican los detalles de los errores*

Un ejemplo de estos errores se muestra en la siguiente ventana:



```
C:\Windows\system32\cmd.exe - ant -buildfile=../build.xml run -Dapplication.args=descripcionAplic...
- Recogiendo item desde el buzón de items de la percepcion...
[java] 2011-04-20 13:58:02,448 [Thread-3] DEBUG root - Ejecutando accion:
crearRecursosNucleoOrganizacion
[java] 2011-04-20 13:58:02,524 [Thread-3] FATAL icaro.infraestructura.entidadesBasicas.descEntidadesOrganizacion.SchemaValidator - No se puede validar el
fichero de configuracion org.xml.sax.SAXParseException: Element type "icaro:RecursosAplicacion" must be followed by either attribute specifications, ">" or "/>"
[java]
[java]
[java] 2011-04-20 13:58:02,524 [Thread-3] ERROR icaro.infraestructura.entidadesBasicas.descEntidadesOrganizacion.SchemaValidator - org.xml.sax.SAXParseException: Element type "icaro:RecursosAplicacion" must be followed by either attribute specifications, ">" or "/>".
[java] 2011-04-20 13:58:02,555 [Thread-3] FATAL icaro.infraestructura.recursosOrganizacion.configuracion.imp.ConfiguracionImp -
[java]
[java] El fichero de descripcion del XML E:\FicheroRed\My Dropbox\Proyectos\ICARO\IcaroMiniCognitivoRMI\.\config\icaro\aplicaciones\descripcionOrganizaciones\descripcionAplicacionAccesoSimple.xml no es valido. Compruebe la sintaxis de la descripcion y vuelva a intentarlo
```

- En el **paso 1.3** se registran las interfaces de uso y de gestión del recurso para que puedan ser utilizables por los clientes (Siempre que no se han producido errores graves en los pasos anteriores).
- En el **paso 1.4** se ordena al **Patrón de Agente reactivo** que cree el **Gestor de la Organización**. El proceso de creación del gestor puede fallar por distintas causas:
 - La ruta del comportamiento que se pasa como parámetro para crear el agente, es errónea
 - La carpeta que indica la ruta no contiene la clase de acciones semánticas y/o no contiene el fichero con el autómata
 - Los nombres de los ficheros son incorrectos
 - El contenido de los ficheros dan errores al ser interpretados

Estos errores aparecerán en la ventana de trazas del Iniciador y en la ventana cmd.exe.

- Si el patrón ha creado sin problemas una instancia de gestor, registra sus interfaces en el repositorio de interfaces: **paso 1.5**
- Una vez creado con éxito el Gestor de Organización, El iniciador recupera del repositorio sus interfaces y le da la orden de arranque utilizando la interfaz de gestión : **paso 1.6**.
- **Al recibir la orden de arranque el Gestor de Organización (GO) comienza a funcionar y el Iniciador desaparece dejando el control de la aplicación al G.O.**

El comportamiento del GO es configurable y esta definido en el fichero de *descripción de la organización*. El comportamiento predefinido esta basado en un agente reactivo. Su autómata y acciones semánticas se encuentran en la carpeta: **src\icaro\gestores\gestorOrganizacion**

Las actividades que realiza el GO se detallan en la siguiente diagrama. Como en el caso del iniciador pueden ocurrir errores en cada actividad, reportándose en la ventana de trazas y en la ventana cmd.exe.

Superada con éxito la creación del Gestor de Organización ahora es él quien toma el control de la creación del resto de componentes de la organización. En primer lugar obtiene la descripción de los agentes bajo su y ordena al patrón que cree un ejemplar con los datos contenidos en la descripción. Una vez que ha sido creado se ordena que arranque, es decir que se ponga a funcionar de acuerdo con su modelo de comportamiento (autómata y acciones semánticas).

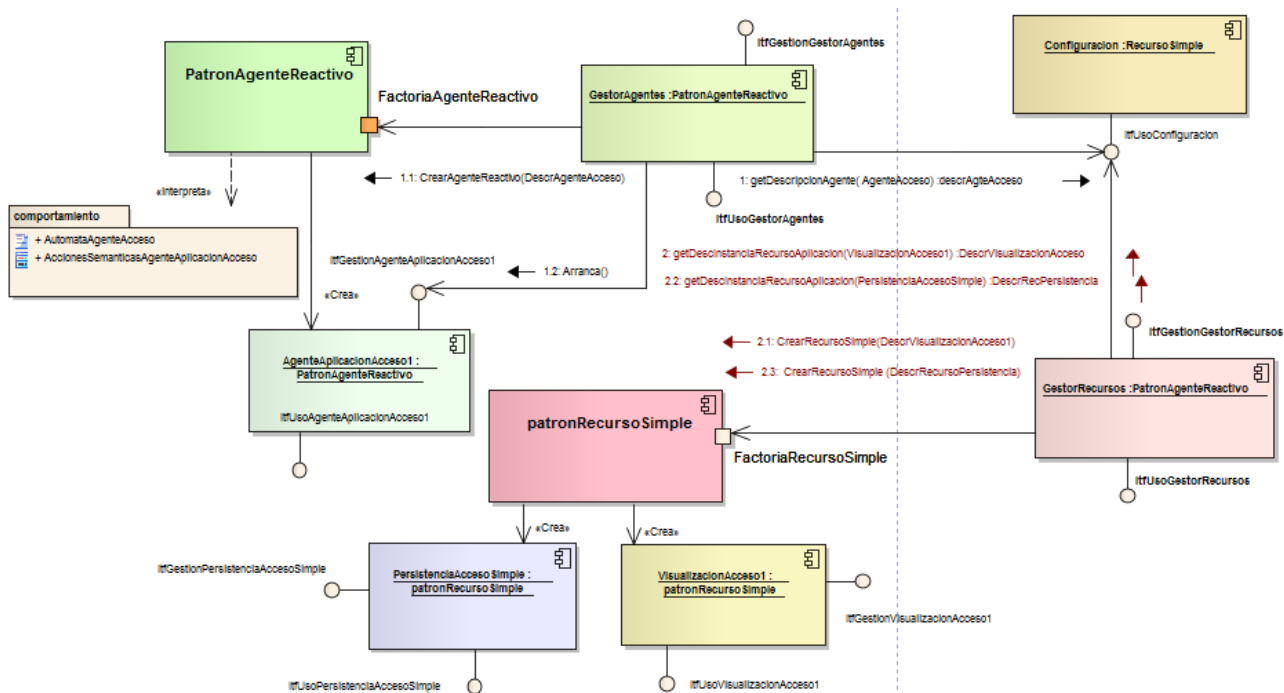


Figura 3-32 Creación de los componentes de la aplicación de acceso

4.6 El ciclo de vida de los gestores

Los gestores son agentes reactivos. Una vez creados y en funcionamiento su comportamiento está definido por su autómata y sus acciones semánticas. El flujo normal de acciones que definen su ciclo de funcionamiento se describe en la Figura 3-34

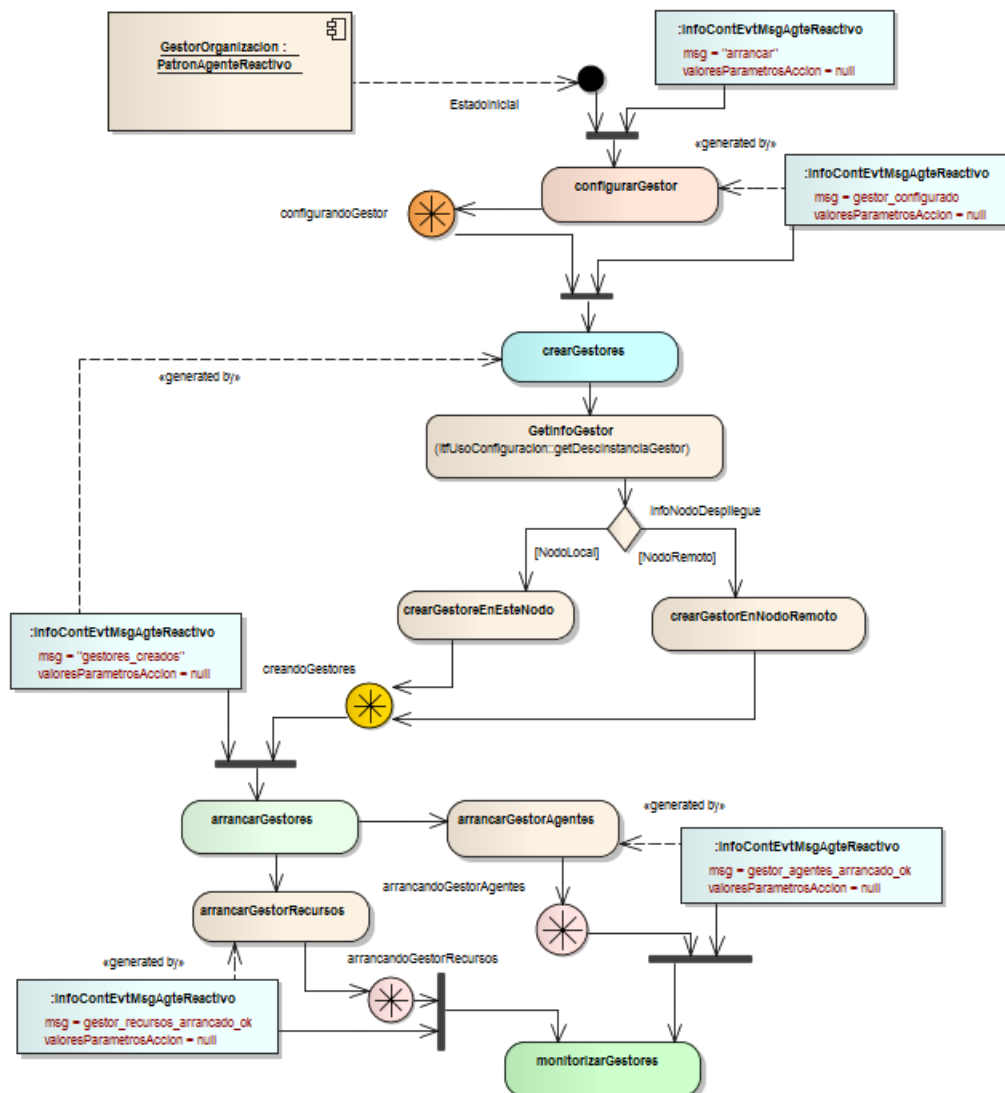


Figura 3-33 Creación de una instancia del patrón de agentes reactivos

En la figura los estados coinciden con los identificadores de los estados en la tabla de estados del autómata y los identificadores de las acciones con los nombres de los métodos de la clase de acciones semánticas.

Cuando el GO arranca:

- Se ejecuta la acción configurarGestor: Se inicializan los parámetros de funcionamiento , por ejemplo el intervalo de monitorización, la dirección del repositorio de interfaces y la referencia a la interfaz del recurso de configuración
- Se obtienen del recurso de configuración la información acerca de las entidades que se deben gestionar
- Se crean los ejemplares gestionados por medio del patrón
- Se arrancan los agentes creados para que comiencen a funcionar.
- Se inicia un ciclo de monitorización para comprobar si existe alguna anomalía o algún informe de error que necesite tratamiento.
- Terminado el ciclo de monitorización se pasa a un estado donde se pueden atender los mensajes enviados por los agentes gestionados

La figura siguiente muestra el autómata que representa el control del Gestor de Organización:

4.7 Implementación de los Casos de uso de la aplicación

Toda aplicación informática debe contemplar al menos tres tipos de casos de uso:

1. Arranque de la aplicación
2. Funcionamiento de la aplicación
3. Terminación

En ICARO el arranque y la terminación la realizan los gestores que tambien se ocupan de la supervisión del buen funcionamiento de los componentes de la aplicación.

4.7.1 Arranque de la aplicación:Activación del agente de Acceso

La aplicación de acceso comienza cuando el agente de acceso inicia su ciclo de vida. Esto ocurre cuando el Gestor de Agentes le da la orden de arrancar. El diagrama de la figura describe el escenario de arranque :

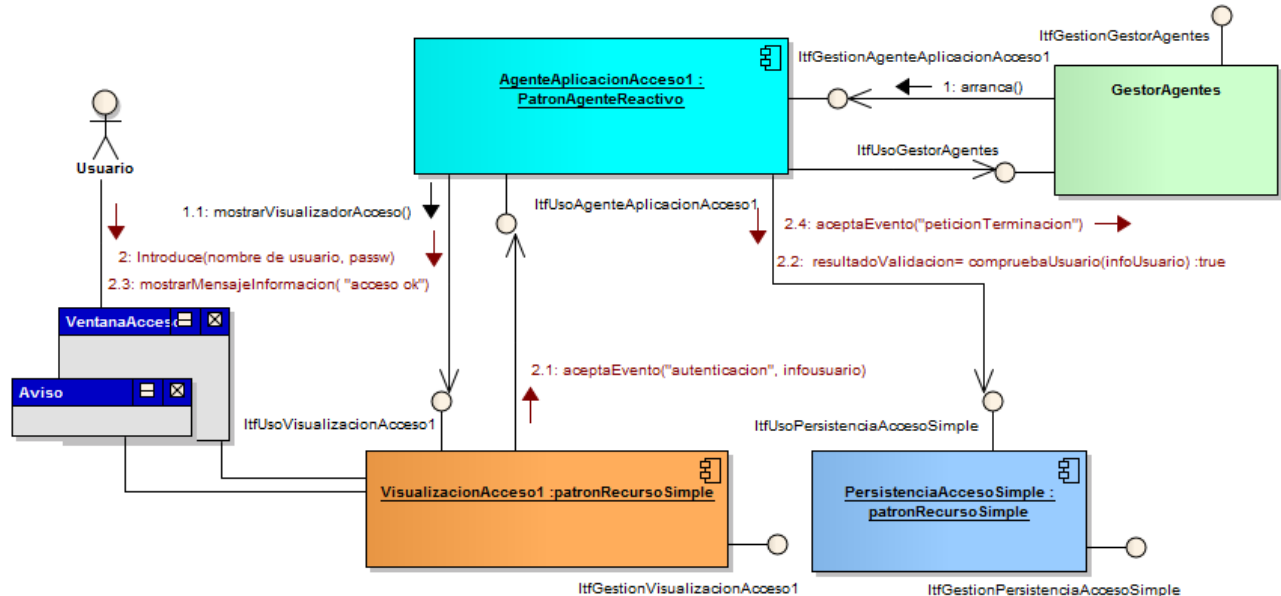


Figura 3-35 Arranque de la aplicación de acceso

La aparición de la ventana de acceso tiene lugar después de que el gestor de agentes cree al agente de aplicación de Acceso y lo arranque (mensaje 1.) En ese momento el autómata del agente de Acceso recibe el input “comenzar”, y ejecuta la acción semántica arranque que utiliza la interfaz de uso de recurso de visualización (mensaje 1.1) para ordenarle que se muestre la ventana de usuario.

La secuencia de mensajes 2, 2.1, 2.2 . 2.3 y 2.4 muestran el flujo de mensajes entre componentes cuando el usuario introduce datos de acceso válidos.

4.7.2 Terminación de la aplicación

Pueden considerarse dos tipos de terminación:

- Terminación explícita por **peticiones de terminación** originadas por los usuarios del sistema.
- Terminaciones extemporáneas debidas a fallos de los componentes del sistema.

En ICARO el principal requisito sobre la terminación del sistema es evitar las terminaciones extemporáneas. El sistema debe **terminar de forma controlada** y debe ser la consecuencia de una **decisión explícita y trazable del Gestor de Organización**. Los demás componentes hacen **peticiones de terminación** que pueden ser analizadas y resueltas por los gestores hasta llegar al Gestor de Organización. Cualquier **terminación no controlada** y no decidida por el GO será considerada como **un fallo del sistema**.

La principal misión de los gestores es detectar anomalías en el funcionamiento de los componentes de la aplicación y actuar en consecuencia, bien intentado corregir los fallos de los componentes monitorizados, o decidiendo su interrupción para evitar males mayores. En esta último caso se puede decidir la terminación de la aplicación, siempre trazando las causas de los fallos e indicando al usuario – cuando sea posible – las causas por las que se producen la terminación.

Las terminaciones explícitas en ICARO se interpretan como peticiones que deben llegar al GO a través de los gestores. Un ejemplo podemos verlo en el sistema de acceso.

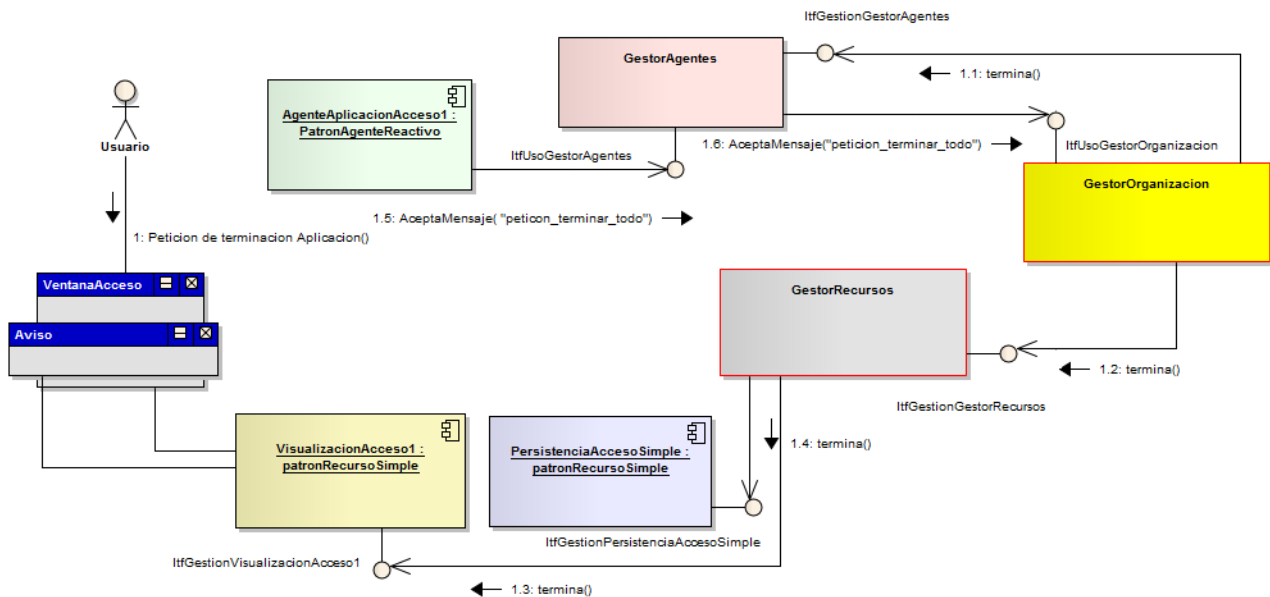


Figura 3-36 Terminación de la aplicación de Acceso

El agente de acceso recibe la petición y decide si aceptarla y darle curso, o pedirle al usuario una confirmación de su petición. Cuando el usuario confirma envía la petición a su gestor y este a su vez la reencamina hacia el Gestor de Organización que es quien decide dar ordenes sus gestores para que terminen. Estos a su vez dan ordenes a los elementos gestionados para que terminen. Los gestores verifican la ejecución de sus ordenes y cuando estan seguros de su cumplimiento terminan.

5 Desarrollo de aplicaciones con ICARO-T

Los componentes de ICARO permiten desarrollar aplicaciones utilizando un modelo organizativo basado en componentes. Los componentes de la aplicación están siempre gestionados y la funcionalidad está implementada por los elementos de la organización; es decir, agentes, recursos o clases del modelo de información. La organización proporciona agentes gestores y recursos que están pensados para facilitar el proceso de desarrollo.

5.1 Ideas básicas

ICARO tiene como objetivos principales facilitar el desarrollo incremental de las aplicaciones y la reusabilidad. Tanto los elementos de la infraestructura -patrones de agentes y patrón de recursos y recursos de la organización, como los agentes y recursos de aplicación, y las clases de dominio son elementos reusables. ICARO promueve una evolución progresiva donde tanto la infraestructura como los componentes de las aplicaciones se enriquezcan con las aportaciones de los desarrolladores.

En el caso de desarrollo de aplicaciones la utilización de agentes y recursos facilita la realización de prototipado rápido que pueden evolucionar sin tener que tirar todo lo hecho anteriormente.

En particular se pueden hacer que evolucione la funcionalidad de la aplicación añadiendo nuevos agentes o recursos y/o cambiando el comportamiento de los componentes iniciales sin dificultades mayores.

- Para cambiar o extender el comportamiento de los agente reactivos basta con:
 - Modificar su autómata añadiendo o modificando sus transiciones
 - Modificar o extender la clase de acciones semánticas
- También es posible:
 - Modificar la política de filtrado y tratamiento de los eventos.
- Para cambiar o extender el comportamiento de los extender los recursos es necesario :
 - adaptar su factoría y sus interfaces, y las clases que las implementan.
 - Modificar la información de los eventos, cuando la comunicación es asíncrona.
- Los componentes o las versiones de los mismos que intervienen en una aplicación pueden activarse y desactivarse de forma declarativa gracias al documento de definición de la organización.

En el ejemplo siguiente se detalla un caso de incremento de funcionalidad que ilustra las facilidades ofrecidas por el sistema para hacer un desarrollo incremental.

5.2 Normas de nombrado

Para asegurar el buen funcionamiento de las aplicaciones desarrolladas en la organización, es necesario seguir ciertos convenios de notación, tanto para crear agentes como para crear recursos.

5.2.1 Nombrado para crear agentes.

Definición de agentes y comportamientos . Dado que un agente puede tener asociado distintos comportamientos que pueden usarse para generar un ejemplar concreto de agente, es necesario respetar el patrón de ruta donde se van a almacenar los comportamientos. Este patrón es el siguiente:

icaro\aplicaciones\agentes\agenteAplicacion<nombreAgente>Reactivo.

En el ejemplo del sistema de acceso el nombreAgente es Acceso y la ruta:

icaro\aplicaciones\agentes\agenteAplicacionAccesoReactivo

Acciones semánticas: Las acciones semánticas correspondientes al agente de aplicación <NombreAgente> deberán llamarse obligatoriamente **“AccionesSemánticasAgenteAplicacion<NombreAgente>”**, y deberá situarse en un paquete de comportamiento situado en la ruta:

icaro\aplicaciones\agentes\agenteAplicacion<nombreAgente>Reactivo\comportamientoX

Utilización de la interfaz de uso del agente: si desde cualquier acción semántica se desea enviar un evento a un agente concreto, debemos utilizar su interfaz de uso. Para ello, debemos recuperarla previamente del repositorio de interfaces donde está registrada, de la siguiente manera (nótese en el siguiente ejemplo que se usa obligatoriamente como nombre clave el que se ha utilizado al registrar el agente en la configuración):

```
ItfUsoAgenteReactivo agenteAcceso = (ItfUsoAgenteReactivo) itfUsoRepositorio.obtenerInterfaz  
(NombresPredefinidos.ITF_USO+"AgenteAplicacionAcceso");
```

5.2.2 Nombrado para crear recursos.

La clase que implementa las interfaces de gestión y de uso para un recurso (clase generadora) debe estar situada de manera recomendable en la ruta =*"icaro.aplicaciones.recursos"*

Por ejemplo, el recurso “Persistencia” posee su clase principal, que implementa las dos interfaces necesarias en la ruta.

"icaro.aplicaciones.recursos.persistencia.imp.PersistenciaImp"/>

Utilización de la interfaz de uso del recurso: si se desea enviar un evento a un recurso concreto, debemos utilizar su interfaz de uso. Para ello, debemos recuperarla previamente del repositorio de interfaces donde está registrada, de la siguiente manera (nótese en el siguiente ejemplo que se usa obligatoriamente como nombre clave el que se ha utilizado al registrar el recurso en la configuración):

```
ItfUsoVisualizadorAccesovisualizacion = (ItfUsoVisualizadorAccesoOrig)  
itfUsoRepositorio.obtenerInterfaz (NombresPredefinidos.ITF_USO+"Visualizacion");
```

5.3 Ejemplos

El comportamiento del sistema de acceso es bastante trivial y no contempla, por ejemplo, el control del número de intentos o la notificación al usuario de los errores que se están produciendo. Consideremos dos posibles mejoras del comportamiento del sistema.

5.3.1 Ejemplo 1: Extensión de la funcionalidad del agente de acceso.

Definimos un nuevo requisito (informal) a la funcionalidad del sistema de acceso:

“Si el usuario realiza un acceso no válido, se le pedirán los datos de acceso –usn y pasw- para darle de alta en el sistema incluyéndolo en la base de datos”.

Los pasos necesarios para la implementación de la nueva funcionalidad del sistema son los siguientes:

4. **Elaborar escenarios de comunicación para concretar el comportamiento del sistema.** Esto debe servir para discutirlo con el cliente del sistema o con los usuarios del mismo y **acordar la funcionalidad que se va a implementar.**
5. **Definición del comportamiento del agente de acceso.** Es necesario definir un nuevo autómata y una nueva clase de acciones semánticas
6. **Adaptación de los recursos a la nueva funcionalidad.** Esto implica el rediseño de los recursos existentes y la implementación de nuevos recursos cuando sea necesario.

7. **Definición de la organización que implementa el nuevo sistema.** Se deben adaptar los gestores para que generen las nuevas versiones de los agentes y de los recursos.

Detallamos en los siguientes apartados cada uno de los pasos.

5.3.1.1 Escenario de comunicación para concretar la funcionalidad del sistema

Vamos a re-utilizar el escenario de comunicación del sistema de acceso y las mismas ventanas. Cuando el sistema detecta que los datos de acceso no son válidos utilizará una nueva ventana para pedirle el usn y el pasw con el que se le va a dar de alta. Figura ().

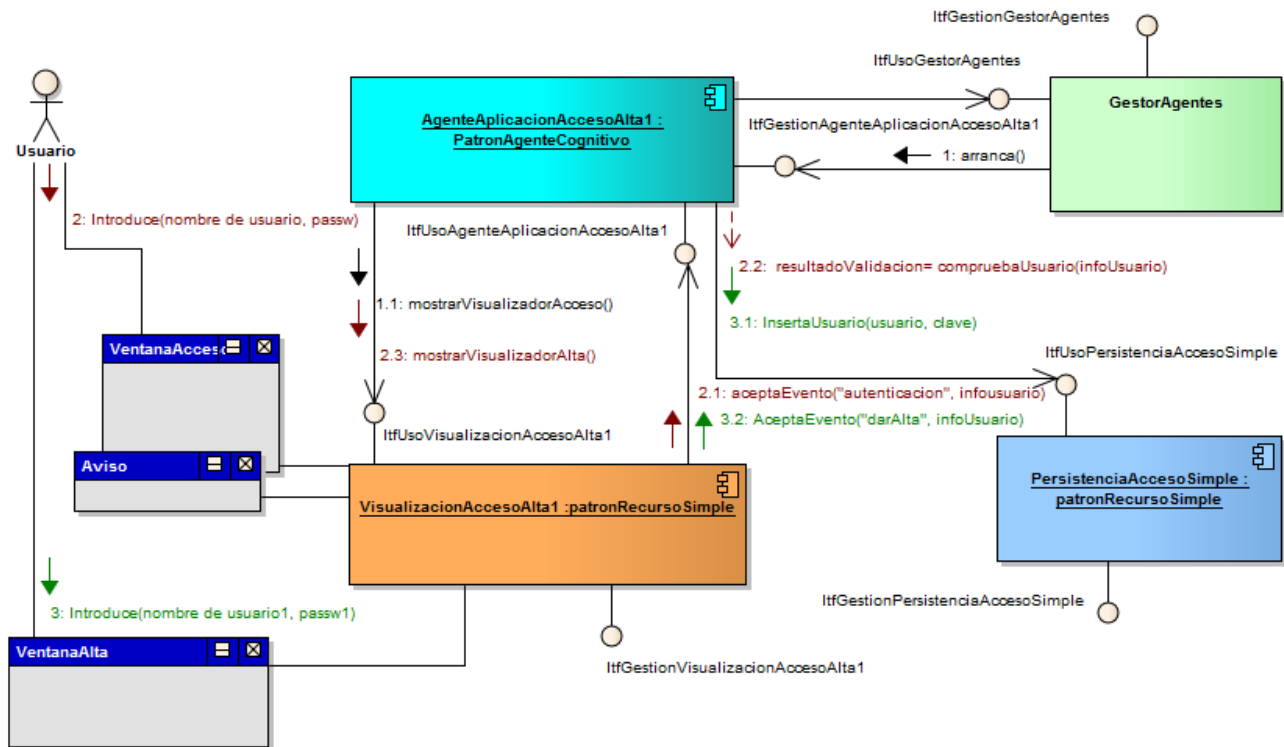


Figura 5-37 Diagrama de comportamiento para la petición de información de Alta

En el diagrama se optó por definir un nuevo recurso de visualización que es una extensión del utilizado en el sistema de acceso pero incorpora la operación: `mostrarVisualizadorAlta`. Esta operación debe implementar la interfaz de usuario para pedirle los datos de alta. Supondremos que estos datos se obtienen a partir de la ventana Alta, donde se le pide al usuario que introduzca su usn, su pasw y que repita el pasw para que se pueda comprobar que coinciden. En caso de que no coincidan se informa al usuario para que introduzca otros datos nuevos. Este comportamiento se deja que lo implemente el propio recurso. Cuando se consigue la información se crea un evento y se le envía al agente para que proceda a su almacenamiento en la base de datos.

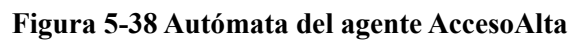
El agente recibe el evento y utiliza el recurso de persistencia para almacenar la información. Si todo va bien se informa al usuario del resultado. Si se producen excepciones en el recurso de persistencia se debería también informar al usuario.

5.3.1.2 Definición del comportamiento del agente de acceso

La definición del comportamiento consiste en definir un nuevo autómata y una nueva clase de acciones semánticas. Los pasos para obtener la implementación del comportamiento son los siguientes.

P1 – Definimos un nuevo paquete *comportamientoAlta* que situamos en la ruta *icaro\aplicaciones\agentes\agenteAplicacionAccesoReactivo*. Este paquete contendrá el autómata y las acciones semánticas del nuevo comportamiento *comportamientoAlta*

Extendemos autómata del sistema acceso con nuevos estados y transiciones. Ejemplo en la figura siguiente.



Crear el fichero y situarlo en el paquete *comportamientoAlta*

```
<?xml version="1.0"?>
<!DOCTYPE tablaEstados SYSTEM "TablaEstados.dtd">
<tablaEstados descripcionTabla="Tabla de estados en pruebas para el agente de acceso">
  <estadoInicial idInicial="estadoInicial">
    <transicion input="comenzar" accion="arranque" estadoSiguiente="esperaLogin" modoDeTransicion="bloqueante"/>
  </estadoInicial>
  <estado idIntermedio="esperaLogin">
    <transicion input="autenticacion" accion="valida" estadoSiguiente="esperaResultadoValidacion" modoDeTransicion="bloqueante"/>
    <transicion input="peticion_terminacion_usuario" accion="pedirTerminacionGestorAgentes" estadoSiguiente="esperandoTerminacion" modoDeTransicion="bloqueante"/>
  </estado>
  <estado idIntermedio="esperandoTerminacion">
    <transicion input="termina" accion="terminacion" estadoSiguiente="terminado" modoDeTransicion="bloqueante"/>
  </estado>
  <estado idIntermedio="esperaResultadoValidacion">
    <transicion input="usuarioValido" accion="mostrarUsuarioAccede" estadoSiguiente="esperandoTerminacion" modoDeTransicion="bloqueante"/>
    <transicion input="usuarioNoValido" accion="mostrarAltaUsuario" estadoSiguiente="esperaAlta" modoDeTransicion="bloqueante"/>
  </estado>
  <estado idIntermedio="esperaAlta">
    <transicion input="alta" accion="darAlta" estadoSiguiente="esperaResultadoAlta" modoDeTransicion="bloqueante"/>
  </estado>
  <estado idIntermedio="esperaResultadoAlta">
    <transicion input="correcto" accion="mostrarNuevoUsuario" estadoSiguiente="esperandoTerminacion" modoDeTransicion="bloqueante"/>
    <transicion input="incorrecto" accion="mostrarAltaUsuario" estadoSiguiente="esperaAlta" modoDeTransicion="bloqueante"/>
  </estado>
  <estado idIntermedio="tratamientoErrores">
    <transicion input="errorIrrecuperable" accion="terminacion" estadoSiguiente="terminado" modoDeTransicion="bloqueante"/>
    <transicion input="errorRecuperable" accion="nula" estadoSiguiente="esperaLogin" 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>
```

P3- Definir las nuevas acciones semánticas del agente.

Las nuevas acciones semánticas (métodos de la clase) son las siguientes:

- Arranque: misma acción semántica que en el AgenteAcceso
- Valida: misma acción semántica que en el AgenteAcceso
- mostrarUsuarioAccede: misma acción semántica que en el AgenteAcceso
- terminacion : misma acción semántica que en el AgenteAcceso
- pedirTerminacionGestorAgentes: misma acción semántica que en el AgenteAcceso
- clasificaError : misma acción semántica que en el AgenteAcceso
- mostrarAltaUsuario
- Entradas: ninguna
- Salida: saca por pantalla una ventana con la que podremos dar de alta a un usuario (nuevo recurso de visualización).
 - darAlta
- Entradas: un usuario (String) y una contraseña (String)
- Salida: se da de alta al usuario con su contraseña en la base de datos (usando el recurso de persistencia).
 - mostrarNuevoUsuario
- Entradas: un usuario (String) y una contraseña (String)
- Salida: se muestra un mensaje indicando que el usuario ha sido dado de alta correctamente. Se utiliza una operación del nuevo recurso de visualización para ejecutar esta acción semántica.

5.3.1.3 Adaptación de los recursos a la nueva funcionalidad

Para este ejemplo, se ha optado por crear un nuevo **Recurso de visualización de alta**. Este recurso, ofrece una interfaz de uso con las siguientes operaciones:

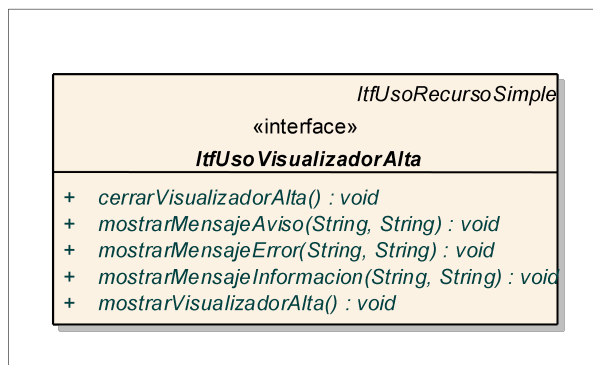


Figura 5-40 Interfaz de uso del recurso de visualización de alta

- mostrarVisualizadorAlta:
 - Entradas: ninguna.
 - Salida: se muestra la ventana de alta por pantalla.
 - cerrarVisualizadorAcceso:
 - Entradas: ninguna
 - Salida: se cierra el visualizador, dejándolo de mostrar por pantalla.
 - mostrarMensajeAviso:
 - Entradas: titulo del mensaje (String) y contenido (String)
 - Salida: muestra por pantalla una ventana de aviso con el titulo y el mensaje dados.

- mostrarMensajeError:
 - Entradas: titulo del mensaje (String) y contenido (String)
 - Salida: muestra por pantalla una ventana de error con el titulo y el mensaje dados.
- mostrarMensajeInformación:
 - Entradas: titulo del mensaje (String) y contenido (String)
 - Salida: muestra por pantalla una ventana de información con el titulo y el mensaje dados.

A partir de la definición de las operaciones de la interfaz se deben seguir los siguientes pasos para su implementación.

P1- Construir la clase que implementa las operaciones de la interfaz. Esta clase se llamará *ClaseGeneradoraVisualizacionAccesoAlta*. En esta clase se delega la creación de los elementos computacionales del recurso de forma que queden disponibles las operaciones de la interfaz de uso para sus clientes.

En el caso de que se produzcan excepciones en el proceso de creación de los componentes internos del recurso, se debe utilizar las operaciones de gestión **para cambiar el estado interno del recurso a un estado de error**, de forma que los gestores puedan aperebirse del estado y actuar en consecuencia.

P2- Esta clase debe heredar de la clase ImplRecursoSimpleImp que es quien implementa las operaciones de gestión.

P3- La clase ItfUsoVisualizadorAccesoAlta también debe heredar de ItfUsoRecursoSimple

El siguiente diagrama de clases describe el modelo final del recurso :

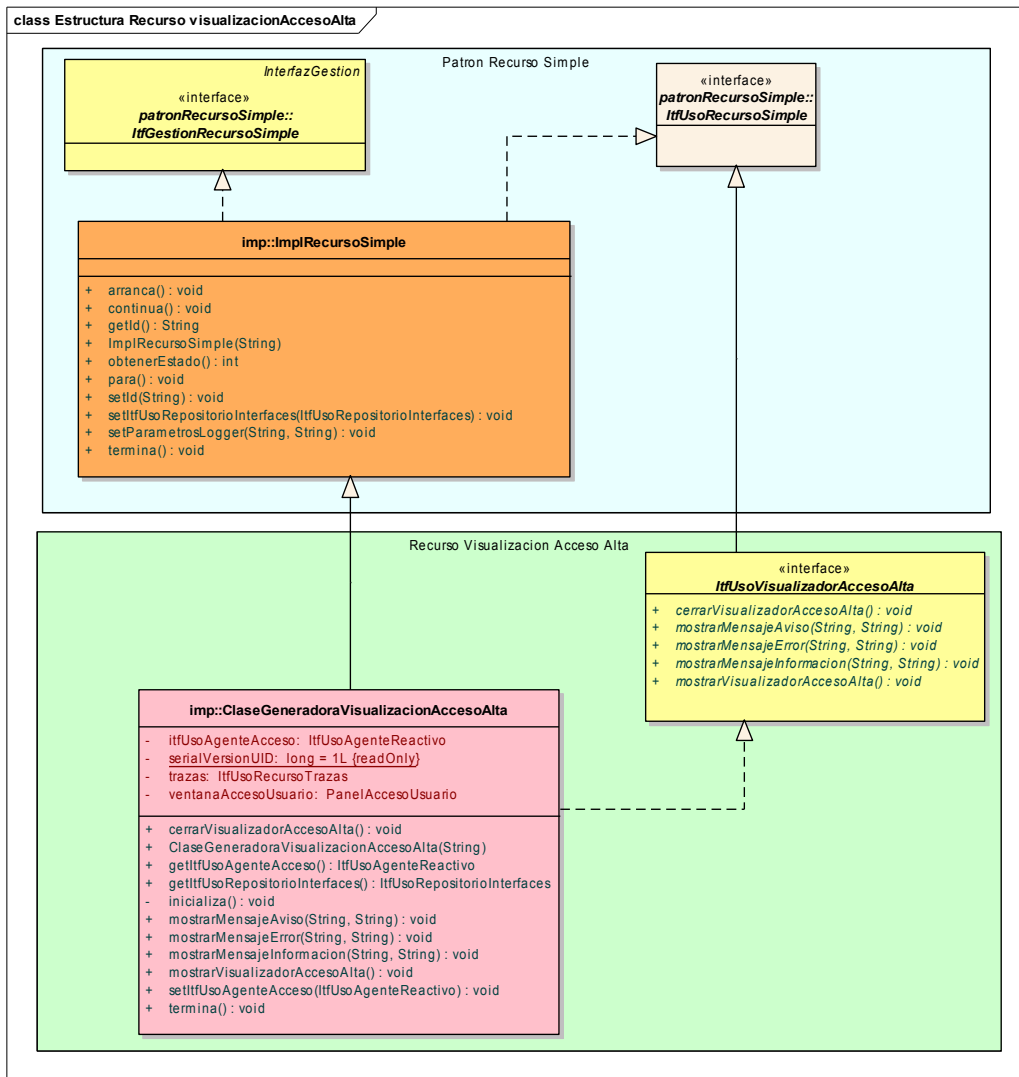


Figura 5-41 Estructura de clases del Recurso

5.3.1.4 Definición de la organización que implementa el nuevo sistema.

Se crea un fichero XML distinto llamado, por ejemplo, “descripcionAplicacionAlta.xml”, a partir de la configuración para el agente de Acceso, situándolo en el mismo directorio :

`\config\icaro\aplicaciones\descripcionOrganizaciones`

Se deben introducir los siguientes cambios:

Definir el comportamiento del agente de acceso que se va a utilizar

```

<icaro:DescComportamientoAgentesAplicacion>
  <icaro:DescComportamientoAgente
    nombreComportamiento="AgenteAplicacionAcceso"rol="AgenteAplicacion" tipo="Reactivo"
    localizacionComportamiento="icaro.aplicaciones.agentes.agenteAplicacionAccesoReactivo.comportamientoAlta
  "/>
  <icaro:DescComportamientoAgente

```

Definición de los recursos. Cambiamos el recurso de visualizacionAcceso por el Recurso de Visualizacion Alta. Este recurso deberá estar situado en "icaro.aplicaciones.recursos.Aplicacion"

No indicamos la ruta de la clase generadora que deberá estar situada en la ruta por defecto "icaro.aplicaciones.recursos. **VisualizacionAccesoAlta.imp**

Ejemplo:

```
<icaro:DescRecursosAplicacion>
  nombre="Persistencia"
  localizacionClaseGeneradora="icaro.aplicaciones.recursos.persistencia.imp.PersistenciaImp"/>
  nombre="VisualizacionAccesoAlta"/>
</icaro:DescRecursosAplicacion>
```

Definición de los - ejemplares de agentes y recursos - que implementarán la funcionalidad de la aplicación.

Se deben cambiar:

Las instancias que deben crear el gestor de recursos, ya que hemos introducido un nuevo recurso.

```
<icaro:InstanciaGestor id="GestorRecursos" refDescripcion="GestorRecursos" >
  <icaro:componentesGestionados>
    <icaro:componenteGestionado refId="Persistencia1" tipoComponente="RecursoAplicacion"/>
    <icaro:componenteGestionado refId="VisualizacionAccesoAlta1" tipoComponente="RecursoAplicacion"/>
  </icaro:componentesGestionados>
</icaro:InstanciaGestor>
```

Las instancias de recursos haciéndola coincidir con la instancia que debe generar el gestor

```
<icaro:Instancia id="VisualizacionAccesoAlta1" refDescripcion="VisualizacionAcceso" xsi:type="icaro:Instancia"/>
</icaro:RecursosAplicacion>
```

El fichero completo puede verse editando **descripcionAplicacionAlta.xml** situado en
`\config\icaro\aplicaciones\descripcionOrganizaciones`

5.4 Un poco más de detalle sobre los patrones

5.4.1 El patrón de agente reactivo

El patrón tiene como cometido crear agentes reactivos computacionales a su imagen y semejanza; es decir, con su misma estructura, pero con los objetos y hebras necesarias para que el modelo de agente reactivo procese información.

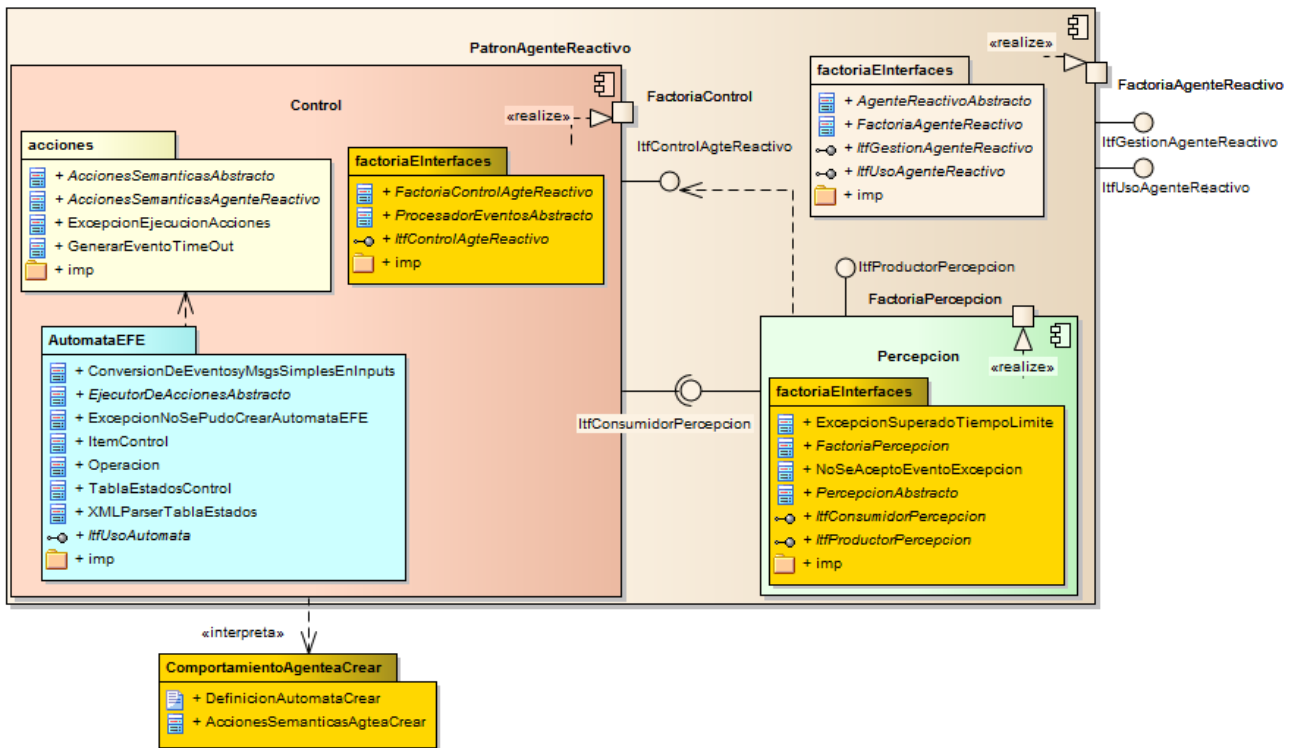


Figura 5-42 Estructura de un agente reactivo

Esta figura contiene la arquitectura del patrón de agente reactivo. Como puede observarse tiene los siguientes componentes:

- **Factoría.** Proporciona los métodos necesarios para crear ejemplares a partir de peticiones de los clientes. Estas peticiones llegan mediante la operación: *crearAgenteReactivo(nombreAgente)*
- **Control:** Es un componente que implementa el modelo de control del agente, que está basado en una máquina de estados finitos. Tiene dos paquetes internos que implementan el autómata y las acciones semánticas.
- **Percepción.** Es un componentes con las clases necesarias para implementar la recepción de eventos, encolarlos y consumirlos.

La creación de ejemplares de agente a partir del patrón se lleva a cabo mediante la operación `crearAgenteReactivo(descripcionAgente)`, donde `descripcionAgente` es un objeto que describe la información necesaria para crear un ejemplar de agentes.

En resumen, para que un cliente pueda crear un agente reactivo, tiene que conocer la factoría del patrón y la información necesaria para crearlo. Esta información consiste en un objeto descripción donde se definen las características del agente a crear. Entre estas características se encuentra la ruta donde el patrón puede encontrar el comportamiento del agente a crear, el identificador de la instancia y otros parámetros adicionales.

La siguiente figura resume el proceso de creación de forma abstracta:

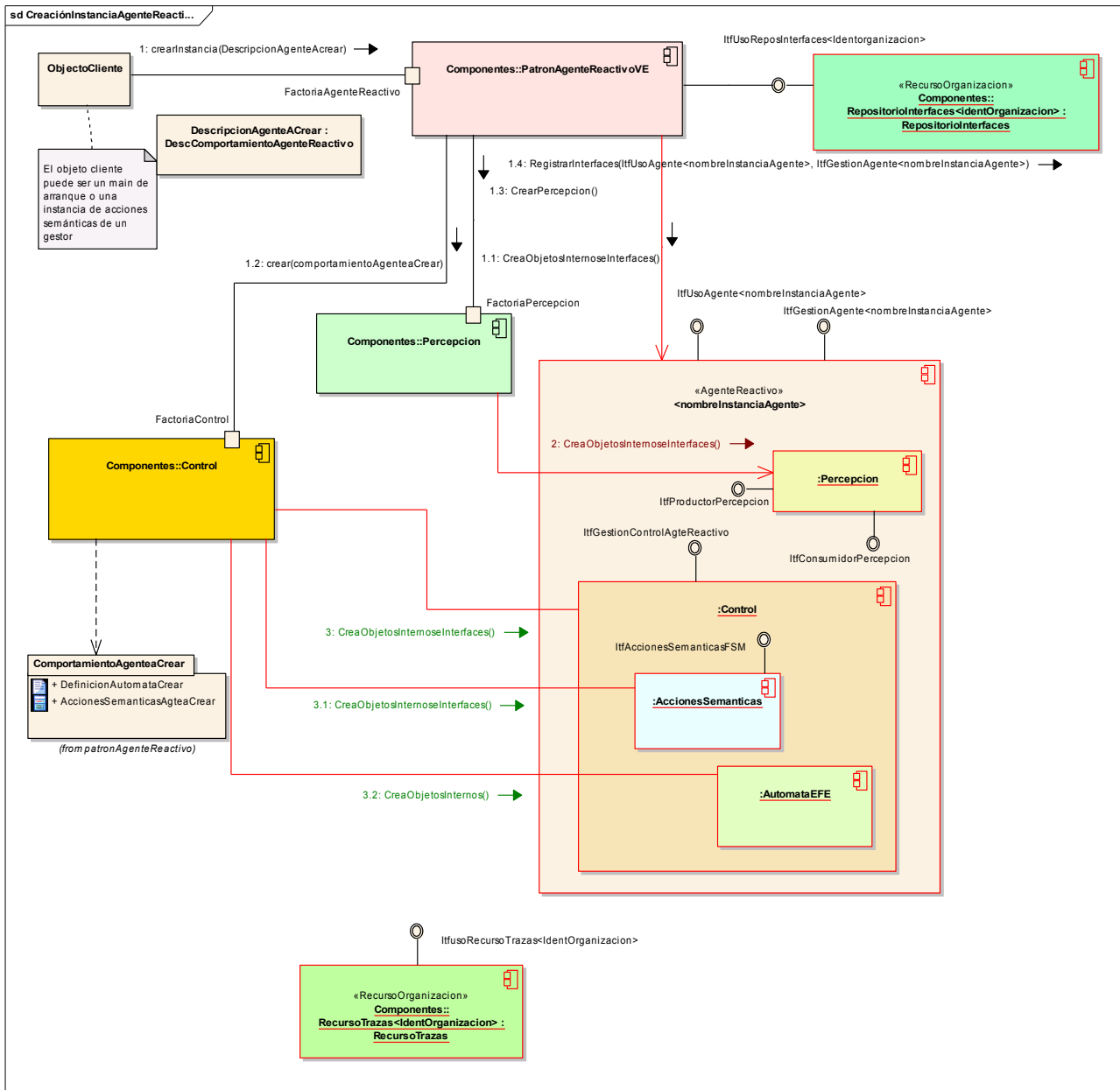


Figura 5-43 Creación de una instancia del patrón de agente reactivo

El proceso de creación se reporta al recurso de trazas.

5.4.2 El patrón de recurso

El patrón de recurso tiene como objetivo proporcionar una visión uniforme de los recursos a través de interfaces similares a las de los agentes.

El patrón permite recubrir un recurso de forma que ofrezca interfaces de uso e interfaces de gestión.

La interfaz de gestión contiene operaciones similares a las de los agentes, mientras que la interfaz de uso debe proporcionar las operaciones propias de la funcionalidad del recurso.

El diagrama siguiente describe la estructura externa del componente con las clases internas del patrón:

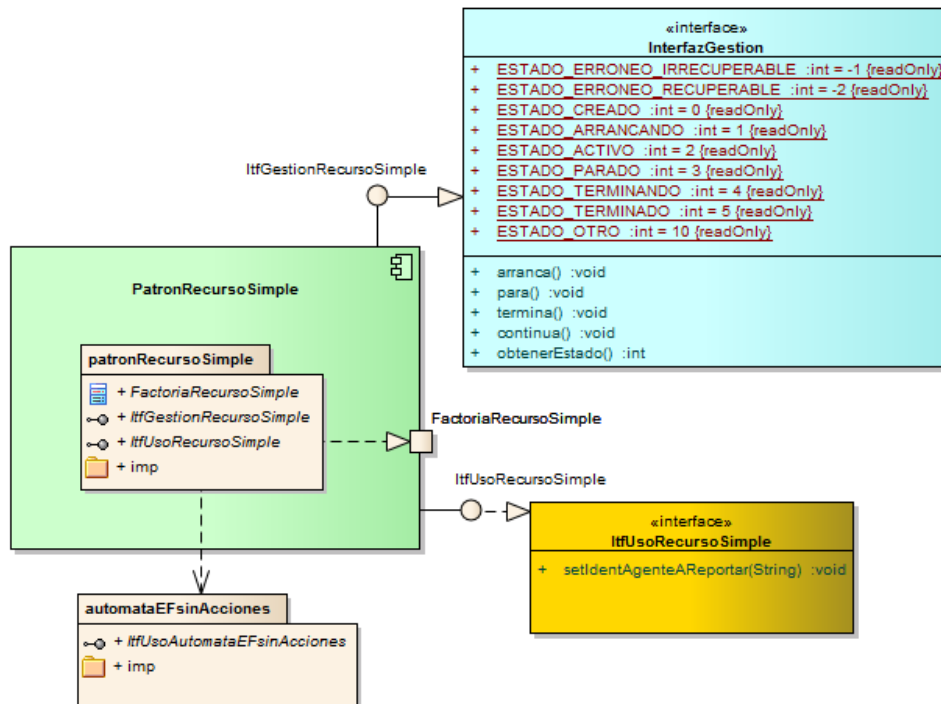


Figura 5-44 Patrón de recurso

La utilización de este patrón es similar al patrón de agentes.

- La factoría crea una instancia de recurso a partir de su descripción y registra sus interfaces de uso y de gestión.
- La clase `ImplRecursoSimple` implementa las operaciones de gestión del recurso. No se proporcionan operaciones en la interfaz de uso. Estas operaciones deben ser definidas e implementadas por el recurso específico.

Los pasos necesarios para crear un recurso de aplicación son los siguientes:

4. Definir una clase “`ClaseGeneradoraRecurso<nombreRecurso>`” que herede de la clase “`ImplRecursoSimple`”. Esta clase implementa las operaciones de las interfaces de gestión y de uso genéricas del patrón
5. Definir la interfaz de uso del recurso. Se recomienda nombrarla `ItfUso<miRecurso>`. Esta interfaz debe heredar de la interfaz de uso del patrón
6. La clase “`ClaseGeneradoraRecurso<nombreRecurso>`” debe:
 - Crear los objetos y componentes internos del recurso reportando errores. Para ello debe utilizar las operaciones de gestión cambiando el estado inicial a estado error.
 - Implementar las operaciones que ofrece el recurso a través de la interfaz de uso. Debe también capturar las excepciones que se produzcan y cambiar el estado del recurso a error.

Un ejemplo ilustrativo se ofrece a continuación con el recurso de visualización:

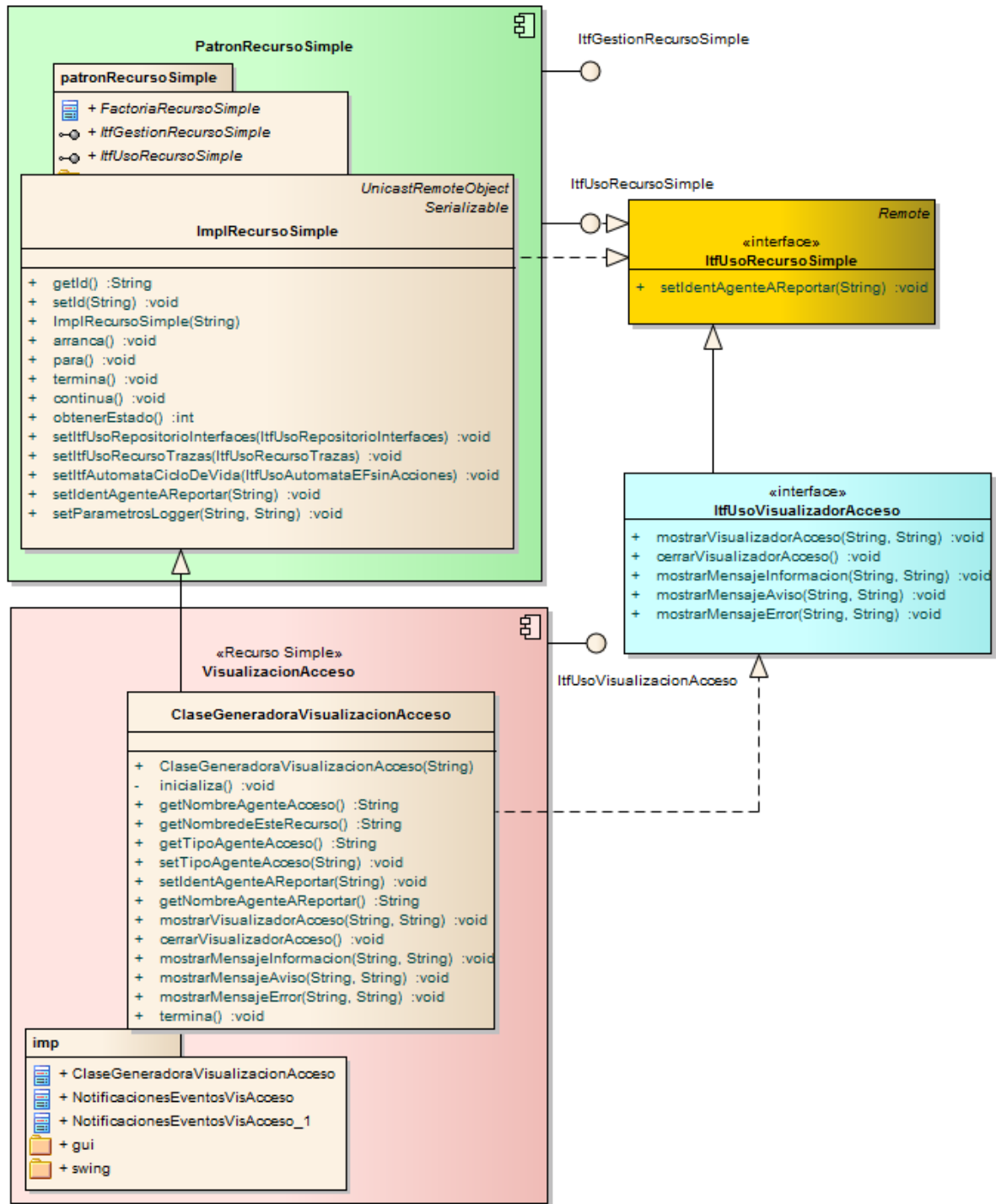


Figura 5-45 Visualizador de acceso

5.4.3 Envío de información asíncrona por medio de eventos

Al pulsar el botón “aceptar” se envía un evento para que se encole en el buzón de la percepción del agente, junto a los datos de consulta (usuario y contraseña)..

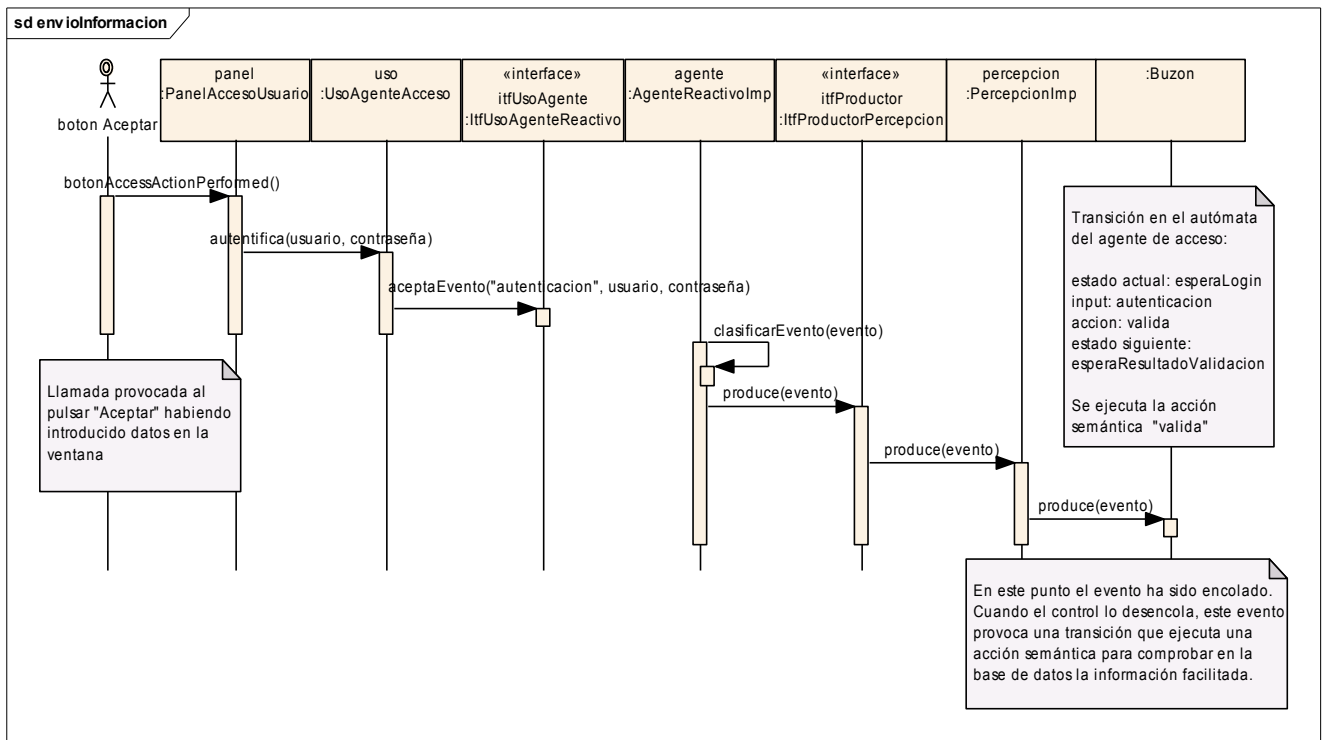
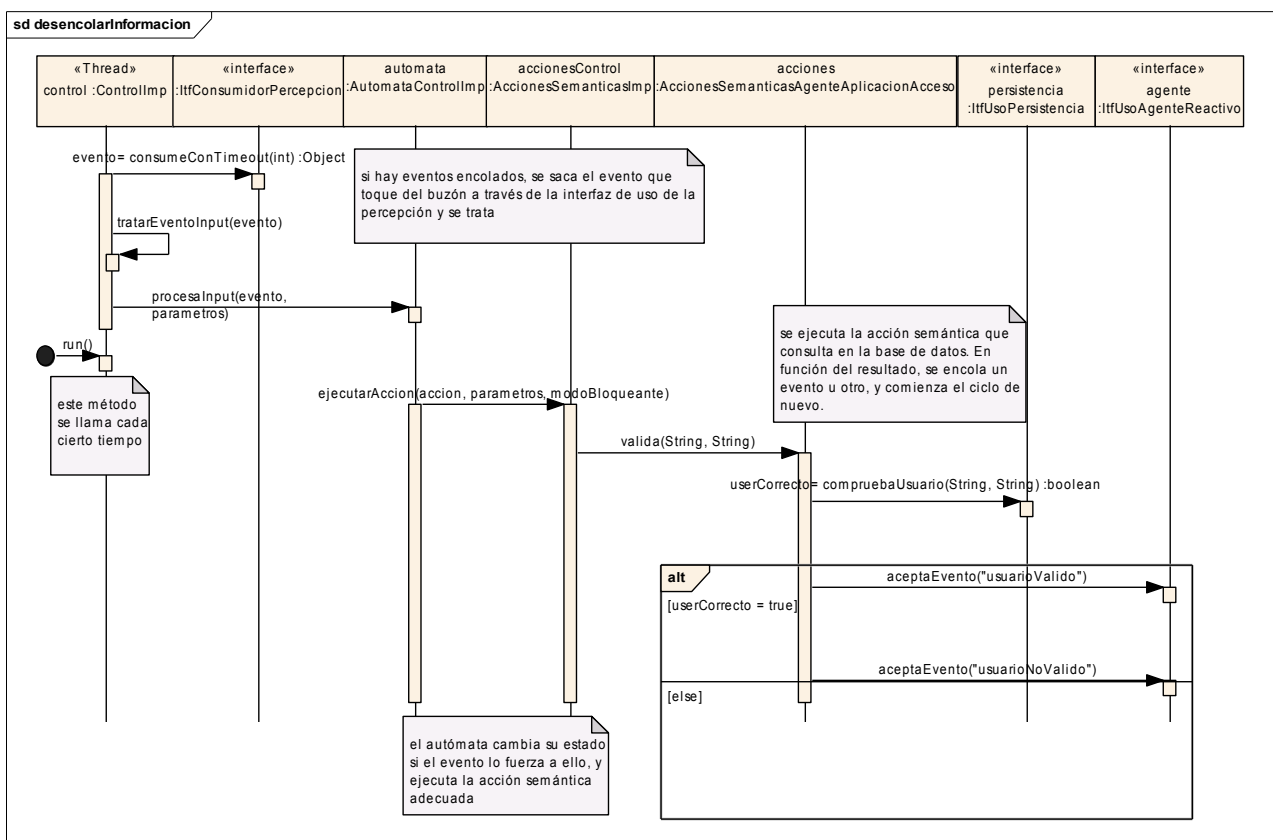


Figura 5-46 Envío de información por parte del agente de Acceso(I)

Este evento provocará, al ser desencolado, una transición en el autómata, y la ejecución de una nueva acción semántica que consultará dichos datos y enviará otro evento para provocar la siguiente transición definida en el autómata.



5.4.4 El metamodelo xsd de la Organización.

El fichero de configuración para el ejemplo de acceso (“descripcionAcceso.xml”) está basado en el esquema xsd (“DescripcionOrganizacion.xsd”). Ambos archivos deben estar situados en el directorio ./Infraestructura/config, teniendo en cuenta que el directorio raíz será el especificado al descomprimir el fichero que contiene la organización. Este esquema “DescripcionOrganizacion.xsd” propone un metamodelo a seguir a la hora de implementar un archivo xml que represente a la organización y a todos sus componentes. En la definición de la organización podremos encontrar, pues, las características de agentes, recursos y gestores que queremos que intervengan en la misma. Las relaciones entre los diferentes elementos de este archivo de esquema xsd se pueden ver a continuación en las siguientes figuras:

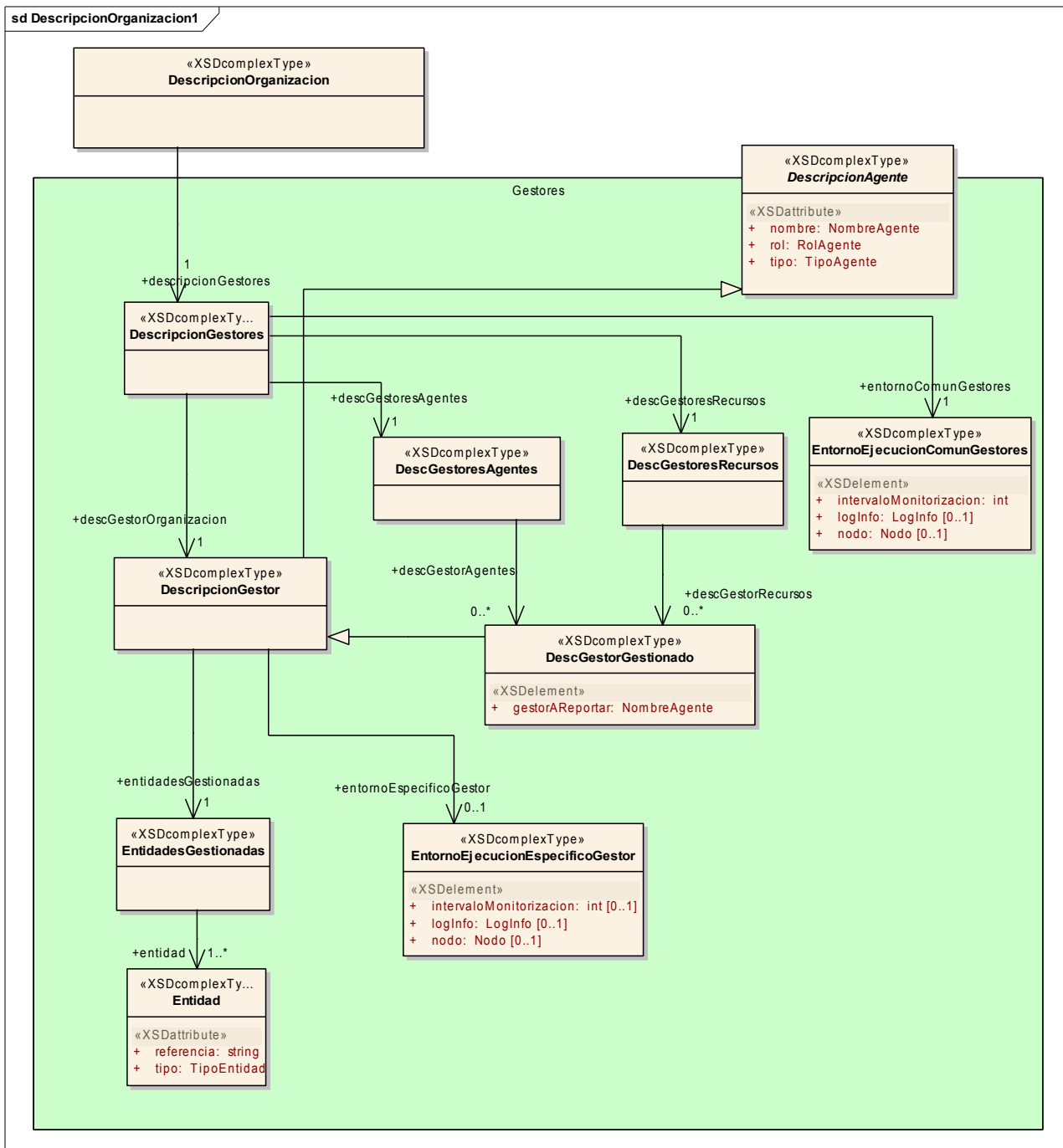


Figura 5-48 Descripción de la organización(I)

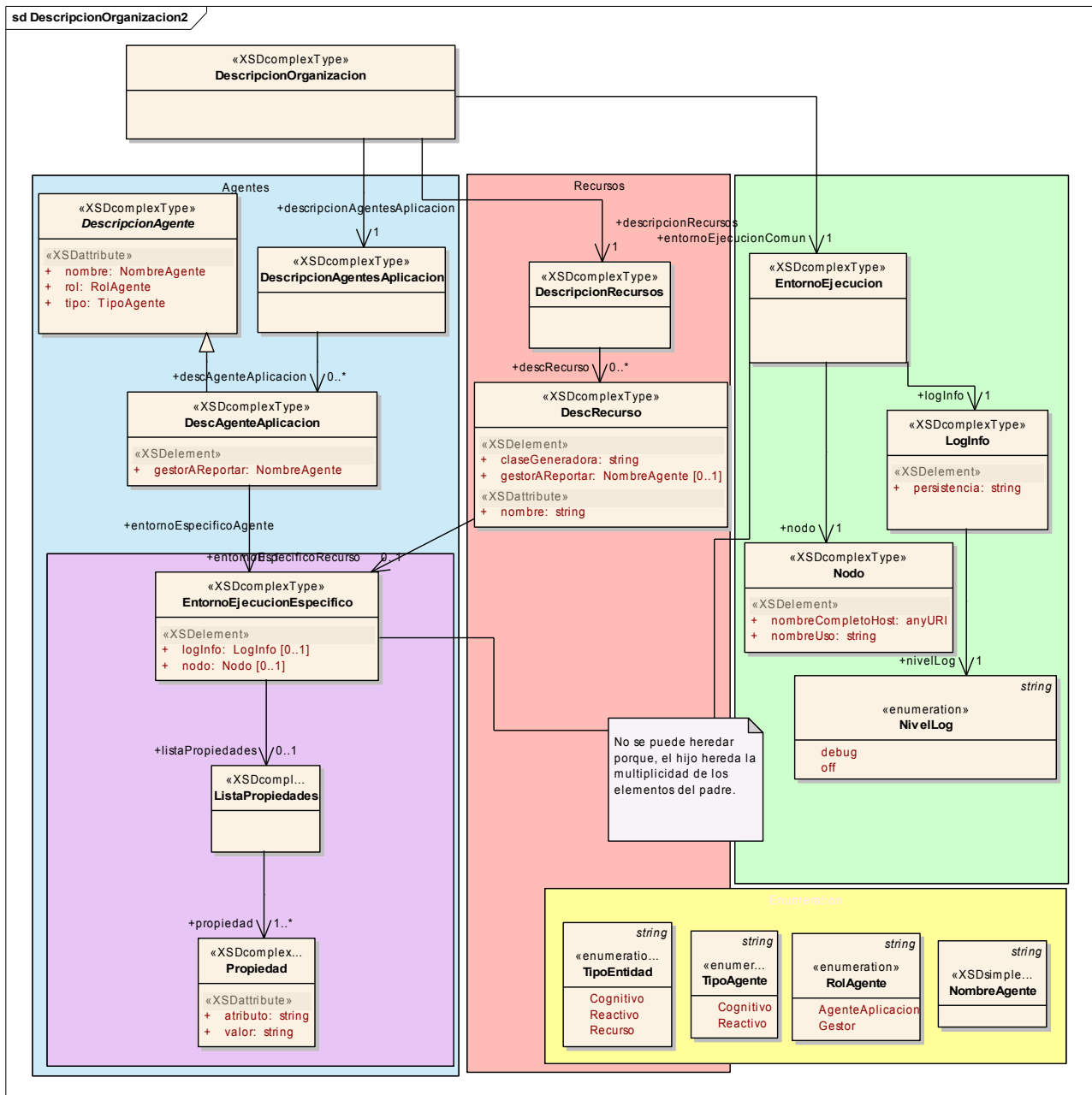


Figura 5-49 Descripción de la organización(II)

A continuación se muestra otro gráfico que especifica la estructura de este esquema, haciendo énfasis en la composición de cada uno de sus elementos:

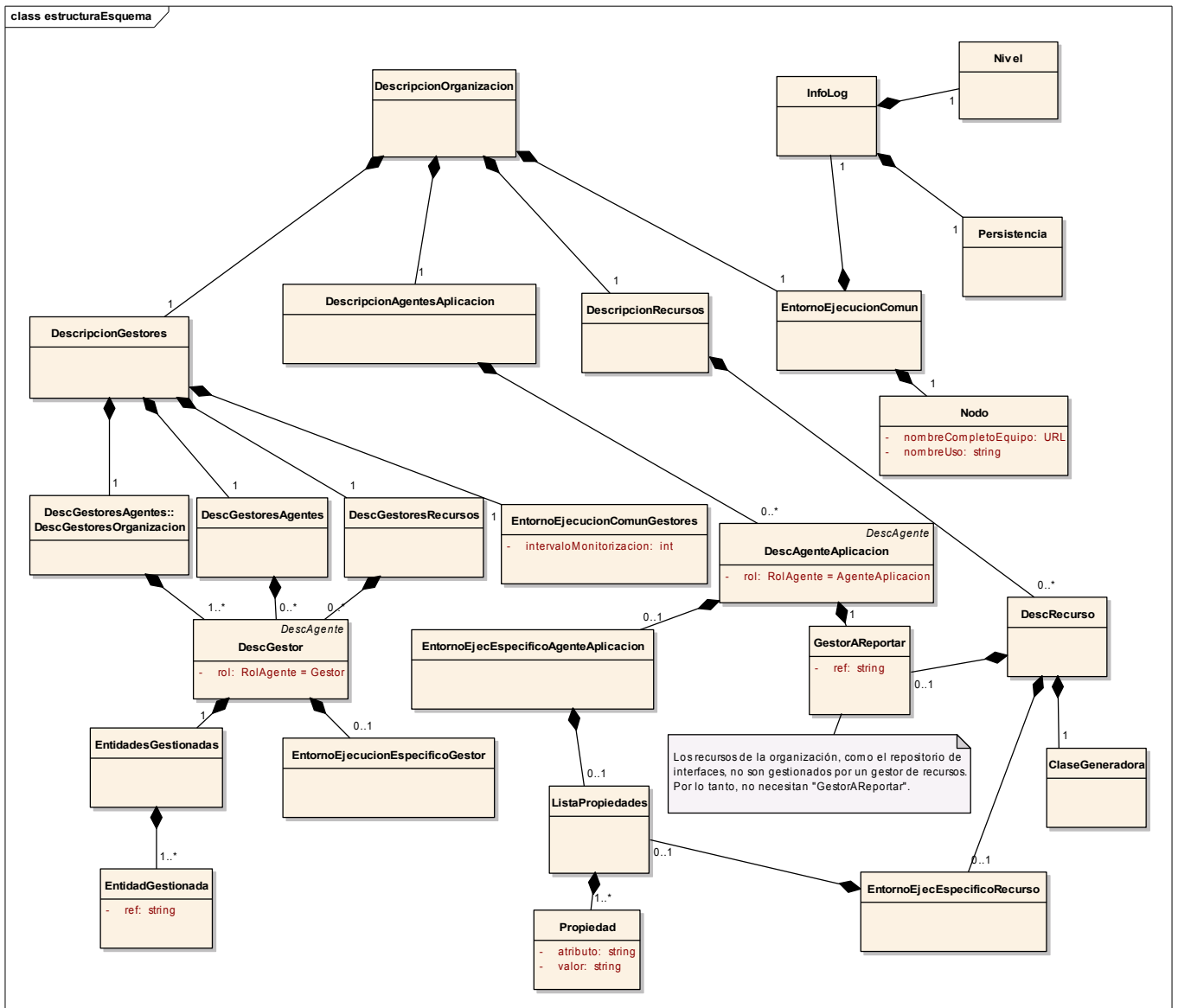


Figura 5-50 Esquema de elementos de la descripción

Seguidamente, se facilita otro gráfico en el que se pueden ver las relaciones de herencia entre los distintos elementos que componen el esquema:

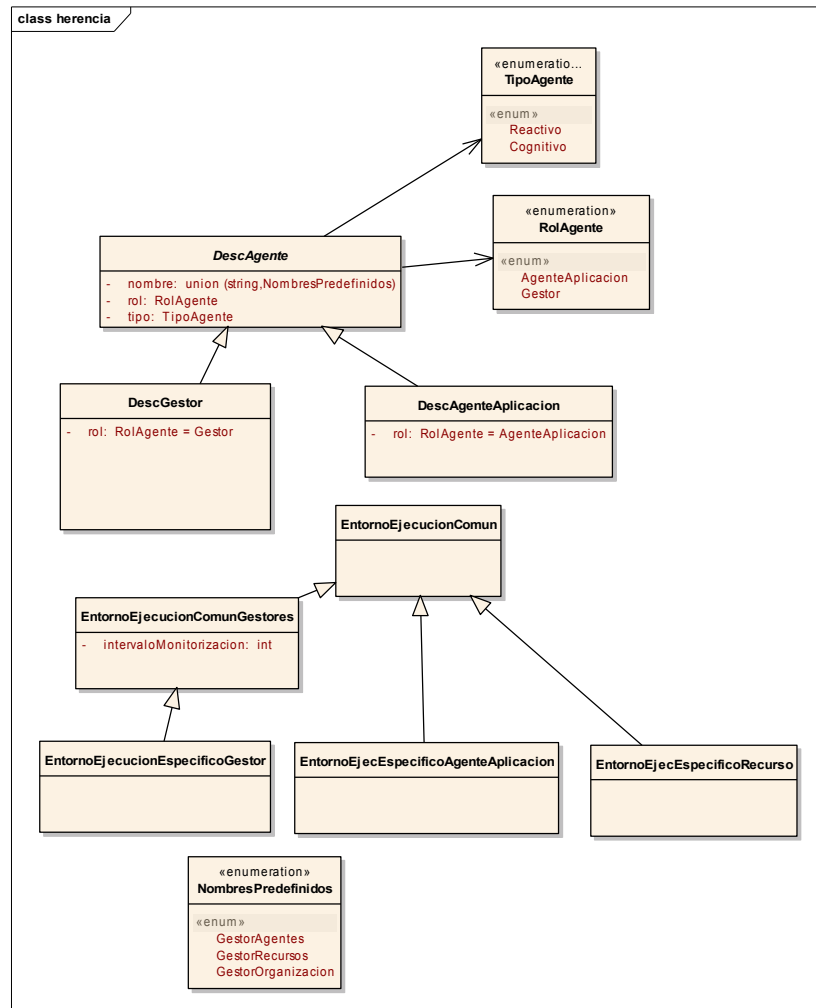


Figura 5-51 Relaciones de herencia entre elementos de la descripción

Por ultimo, señalamos que el código del archivo xsd se ha generado con la herramienta “Enterprise Architecture” a partir del diseño, lo cual resulta muy útil en caso de que se quiera modificar alguna característica de manera visual.

5.4.5 Generación de clases conectoras Java con JAXB.

Dentro de la carpeta “./Infraestructura/src” (que habrá sido creada en el directorio especificado al descomprimir), encontramos un subdirectorio llamado “organización/infraestructura/recursos/configuración/imp/jaxb”. Esta carpeta contiene una serie de clases destinadas a permitir la navegación a través del fichero xml y extraer información del mismo. Estas clases han sido creadas mediante el script “creacionJAXB.bat”, que se encuentra en “./Infraestructura/dev”. Si se edita este script, se observa que con un par de pequeños cambios podríamos regenerar nuevas clases a partir de un fichero xsd distinto, depositándolas bajo el directorio que deseemos:

xjc -d rutaOrigenCódigoFuente rutaOrigenFicheroXSD -p nombre.paquete.destino

El uso de estas clases es transparente al usuario, ya que se ha diseñado un recubrimiento que ofrece las operaciones necesarias para utilizar en la organización. Este recubrimiento se explica con detalle a continuación.

5.4.6 La clase “Configuración”.

Este recubrimiento permite extraer la información necesaria del fichero xml para utilizarla en la organización a través de una serie de operaciones suficientes. Esto justifica que el recubrimiento se haya modelado como un recurso más de la organización, con su interfaz de uso y de gestión. Su apariencia es la siguiente:

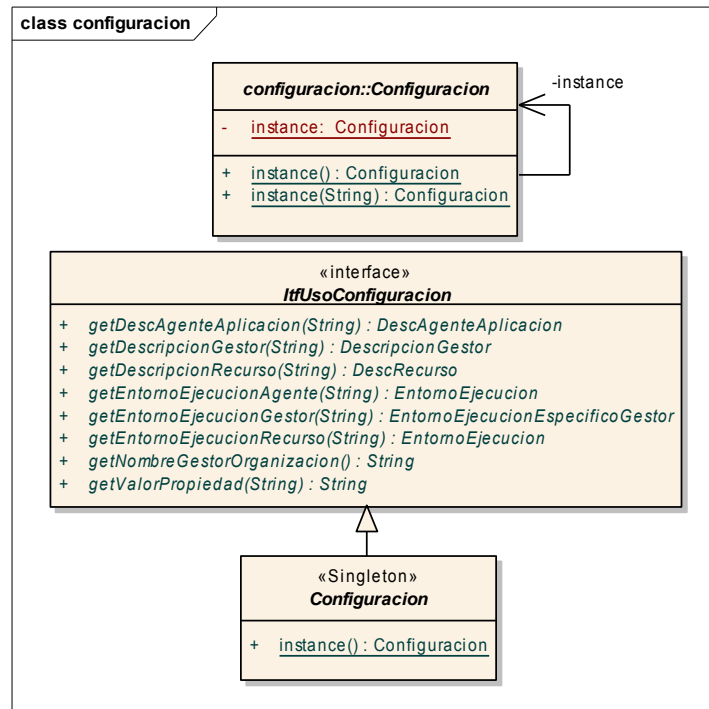


Figura 5-52 Diagrama UML del recurso de configuración

A continuación se explican con detalle cada una de las operaciones que ofrece la interfaz de uso de este recurso propio de la organización:

- `getDescAgenteAplicacion(String)`
 - Entradas: nombre clave del agente de aplicación del cual queremos su descripción
 - Salidas: descripción del agente buscado como instancia de la clase “DescAgenteAplicacion”
- `getDescripcionGestor(String)`
 - Entradas: nombre clave del gestor del cual queremos su descripción
 - Salidas: descripción del gestor buscado como instancia de la clase “DescripcionGestor”
- `getDescripcionRecurso(String)`
 - Entradas: nombre clave del recurso del cual queremos su descripción
 - Salidas: descripción del recurso buscado como instancia de la clase “DescRecurso”
- `getEntornoEjecucionAgente(String)`
 - Entradas: nombre clave del agente del cual queremos su entorno
 - Salidas: entorno del agente buscado como instancia de la clase “EntornoEjecucion”
- `getEntornoEjecucionGestor(String)`
 - Entradas: nombre clave del gestor del cual queremos su entorno
 - Salidas: entorno del gestor buscado como instancia de la clase “EntornoEjecucionEspecificoGestor”
- `getEntornoEjecucionRecurso(String)`
 - Entradas: nombre clave del agente del cual queremos su entorno

- Salidas: entorno del recurso buscado como instancia de la clase “EntornoEjecucion”
 - getNombreGestorOrganizacion()
- Entradas: ninguna
- Salidas: String con el nombre clave del gestor de organización, que se encuentra en el xml
 - getValorPropiedad(String)
- Entradas: nombre clave de una propiedad incluida en el xml
- Salidas: valor de dicha propiedad, que también habrá sido especificado en el xml

Por último, se ofrece el diagrama UML del recurso, haciendo énfasis en sus componentes internos. Cabe reseñar que el recubrimiento posee una dependencia con las clases generadas por JAXB.

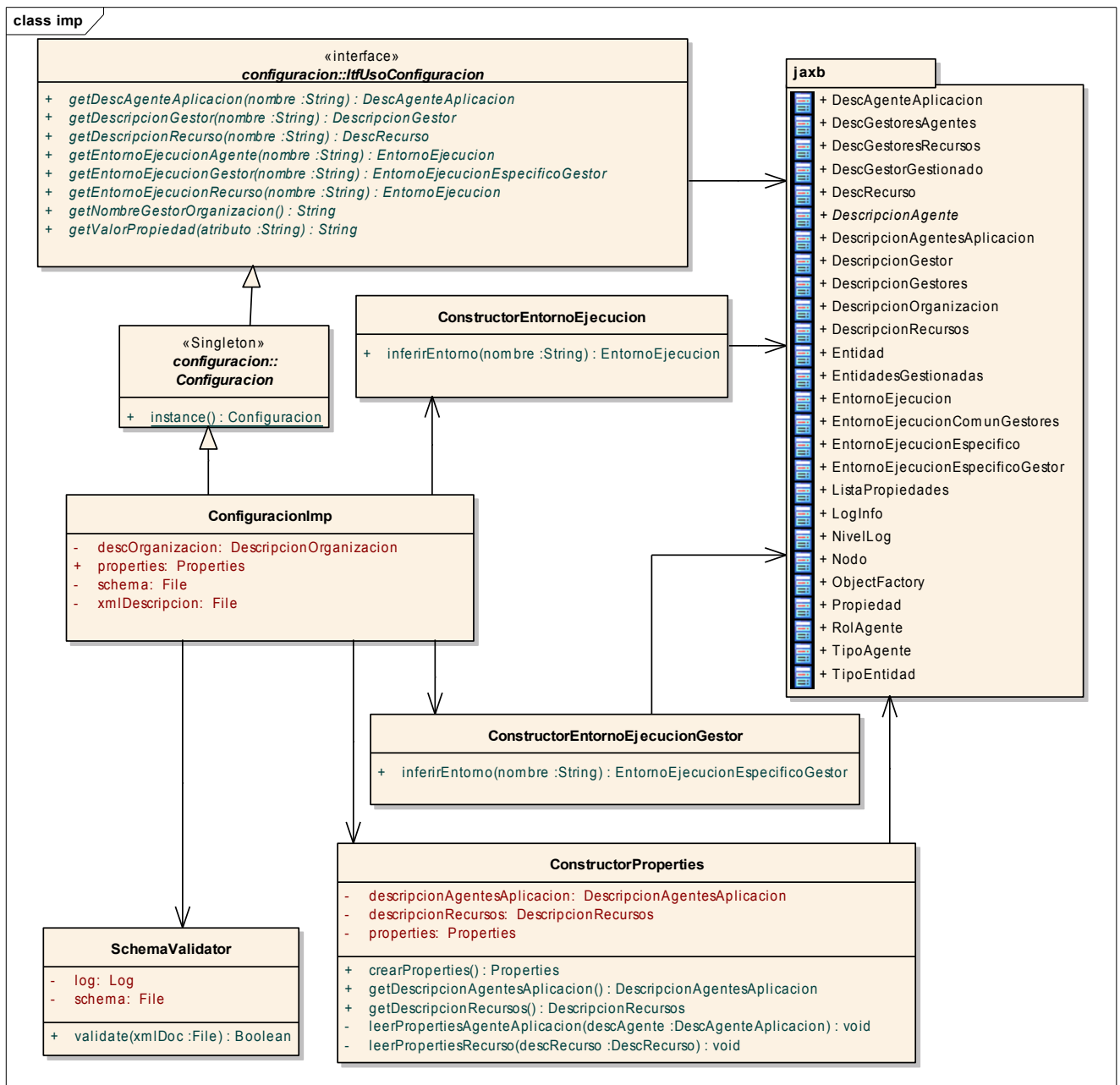


Figura 5-53 Diagrama UML para la implementación de la Configuración

6 Referencias (incompletas)

1. Ana Mas, Agentes Software y Sistemas Multi-Agentes: Conceptos, Arquitecturas y Aplicaciones, section 2.3
2. Ana Mas, Agentes Software y Sistemas Multi-Agentes: Conceptos, Arquitecturas y Aplicaciones, section 2.2.2
3. Garijo, F J., Bravo S., Gonzalez, J. Bobadilla E. BOGAR_LN: An Agent Based Component Framework for Developing Multi-modal Services using Natural Language. L.N.A. I, Vol 3040. Pp 207-. Springer-Verlag.2004
4. Charles Forgy, Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, Artificial Intelligence, 19, pp 17-37, 1982