UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

SISTEMA MULTI-AGENTE DE APOYO PARA EL PROCESO DE ADMINISTRACIÓN DE PROYECTOS ESPECÍFICOS DE MOPROSOFT

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRA EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

AURORA LOZADA RODRÍGUEZ

DIRECTORA DE TESIS:

M. EN C. MA. GUADALUPE ELENA IBARGÜENGOITIA GONZÁLEZ

México, D.F. **2007.**

AGRADECIMIENTOS

A mi familia que siempre me dió su apoyo incondicional y que se ha preocupado e interesado por mis avances como profesionista y ser humano. Gracias a mi madre y hermanos por enseñarme que no hay límites, que lo que me proponga lo puedo lograr y que sólo depende de mí. Se que cuento con ellos siempre.

A la M. en C. Ma. Guadalupe Elena Ibargüengoitia González por su valiosa asesoría y dirección durante el desarrollo de esta tesis.

A mis sinodales por su participación. Agradezco su tiempo, comentarios y sugerencias.

A cada uno de los profesores que participaron en mi desarrollo profesional durante el posgrado. Agradezco por transmitirme sus conocimientos, y sobre todo, porque con su ejemplo me inspiran a seguir trabajando.
.

A Adidier y su familia, personas que desde el primer momento me brindaron su apoyo.

A los amigos, familiares y a todas aquellas personas que han contribuido de alguna manera para que esto fuera una realidad.

Así mismo agradezco los apoyos académicos y económicos al Posgrado en Ciencia e Ingeniería de la Computación y al CONACyT.

A todos mi mayor reconocimiento y gratitud.

Aurora Lozada Rodríguez
Noviembre del 2007

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

MOTIVACIÓN

Desde el surgimiento del Modelo de Procesos para la Industria de Software (MoProSoft), como norma mexicana en agosto del 2005, la implantación de este modelo de calidad en empresas pequeñas y medianas de desarrollo y mantenimiento de software se ha dado paulatinamente con la finalidad de tener acceso a las prácticas de Ingeniería de Software de clase mundial.

Sin embargo, las empresas que han decidido adoptar MoProSoft manifiestan la necesidad de contar con una herramienta que les facilite la labor de implementar MoProSoft y en especial la parte que comprende la administración de proyectos debido a la complejidad que implica este proceso. Razón por la cual, se decidió crear en el curso de Ingeniería de Software Orientado a Objetos del 2006 de la Maestría en Ciencia e Ingeniería de la Computación de la Universidad Nacional Autónoma de México (UNAM), una Herramienta para la Administración de Proyectos para MoProSoft (HeAP MoProSoft).

Esta herramienta web tiene como objetivo apoyar a las pequeñas y medianas empresas (PyMES) y las áreas internas dedicadas al desarrollo y/o mantenimiento de software, en la administración de sus proyectos siguiendo MoProSoft. Con la finalidad de obtener una herramienta robusta e innovadora a través del uso de tecnologías avanzadas, se plantearon tres temas de tesis complementarios que manejarán algún aspecto específico de este desarrollo.

Los objetivos que se persiguen con estas tesis son:

1. Construir los cimientos de la herramienta utilizando el marco de trabajo “Struts” de java.
2. Incluir tableros de control para visualizar el estado de los proyectos.
3. Ofrecer una consultoría de información, proveniente de varias fuentes de datos al usuario de acuerdo a su rol.

Con respecto al último objetivo, cabe mencionar que la información a brindar debe ser una guía para la realización de las tareas que el usuario debe hacer de acuerdo a la herramienta y a lo definido en MoProSoft para la administración de proyectos.

El paradigma conocido como “agente/Sistema Multi-Agente”, perteneciente al área de Inteligencia Artificial, resulta ser la opción adecuada para desarrollar la funcionalidad requerida en el último objetivo. Numerosas aplicaciones basadas en este paradigma están siendo empleadas exitosamente en infinidad de áreas. En concreto, las aplicaciones de “gestión de información” constituyen una forma para el desarrollo de sistemas de software que requieren la gestión, de una forma racional, de la información para un usuario en específico.

Esta tecnología ofrece la posibilidad de construir herramientas donde el usuario indica la información que necesita y automáticamente la aplicación se encarga de buscar dónde puede encontrar los datos adecuados para satisfacer la solicitud.

Existe un conjunto de técnicas para la construcción de este tipo de herramientas que permiten el análisis, diseño e implementación con un desarrollo incremental para aceptar nuevos elementos. Para el mantenimiento se cuenta con la facilidad de adaptación a modificaciones y escalabilidad (añadir agentes para soportar mayor carga de trabajo o dotar a los agentes de nuevas capacidades).

De lo expuesto en los párrafos precedentes, se extraen los factores que han motivado el desarrollo de esta tesis: anexar la funcionalidad de consulta de información que guíe al usuario en la realización de tareas dentro de la herramienta HeAP MoProSoft, así como el uso de una tecnología adecuada para la resolución de este problema y que ofrezca una manera novedosa, independiente y más eficiente de hacerlo.

Se intenta mostrar que es viable la construcción de sistemas de software utilizando agentes para explotar conocimientos de áreas como la Inteligencia Artificial y la Ingeniería de Software.

OBJETIVO

El presente proyecto de tesis se centra en el desarrollo de un Sistema Multi-Agente que se integrará a la herramienta HeAP MoProSoft cuyo objetivo es brindar información al usuario mediante sugerencias, ejemplos o avisos para guiarlo en las tareas que realiza de acuerdo a lo definido en la administración de proyectos por MoProSoft.

ALCANCE Y CONTRIBUCIÓN

Cabe señalar que HeAP MoProSoft en su primera versión abarca dos procesos: Gestión de Proyectos y Administración de Proyectos Específicos. Además, los usuarios serán las personas que desempeñen los roles responsables de dichos procesos: Responsable de Gestión de Proyectos (RGPY) y Responsable de Administración del Proyecto Específico (RAPE).

Por otro lado, se pretende hacer una consulta de información muy específica tomada de dos fuentes de datos: la Base de Datos de HeAP MoProSoft y la Base de Datos de Soporte. La construcción de esta última entra dentro del alcance de este trabajo.

En lo referente a la contribución de la herramienta con el Sistema Multi-Agente propuesto, con su aplicación en la industria mexicana de desarrollo de

software se pretende favorecer la adopción de MoProSoft y apoyar las actividades relacionadas con la administración de proyectos. En concreto, se busca dar una guía y apoyo para los usuarios (con los roles RGPY y RAPE) de HeAP MoProSoft con la finalidad de aumentar su aprendizaje y desempeño en la implementación de los procesos de que son responsables.

ESTRUCTURA DE LA TESIS

La presente tesis consta de seis capítulos, las conclusiones y cinco apéndices.

En el capítulo primero, se presentan los orígenes de las contribuciones de este proyecto. Se describe brevemente el Modelo de Procesos para la industria del Software (MoProSoft) y la Herramienta para la Administración de Proyectos para MoProSoft (HeAP MoProSoft).

El capítulo segundo presenta los conceptos fundamentales relacionados con el tema: Agentes y Sistemas Multi-Agente. Con ello se pretende ofrecer un conocimiento de la teoría que sustenta el presente trabajo.

El capítulo tercero, expone el planteamiento del problema y la solución propuesta de incluir un Sistema Multi-Agente a la herramienta HeAP MoProSoft con la finalidad de dar una idea clara del contenido y propósito del proyecto de tesis. El capítulo se completa introduciendo las metodologías de trabajo y tecnologías utilizadas como parte fundamental de la solución del problema.

El cuarto capítulo se centra en la aplicación de la metodología INGENIAS correspondiente a la fase de análisis para el desarrollo del Sistema Multi-Agente. Se describe los meta-modelos usados en la especificación del sistema. Cada meta-modelo se centra en una vista o aspecto de definición del Sistema Multi-Agente (agentes, entorno, interacción, organización, objetivos y tareas).

El quinto capítulo comprende la fase de diseño del Sistema Multi-Agente aplicando la metodología INGENIAS. Se presentan los modelos a partir de los cuales se realizará la implementación. Además, el capítulo se complementa con la definición del diseño de la Base de Datos de Soporte, en donde se describe los objetivos que se persiguen con su construcción, la recolección de los datos, su modelo lógico y el uso del patrón DAO para el acceso a los datos.

El sexto capítulo aborda la construcción del Sistema Multi-Agente. Para ello, se define la arquitectura general del sistema y la explicación del desarrollo de cada uno de los componentes definidos en la fase del diseño. Se incluye también la integración con HeAP MoProSoft, así como la definición de las pruebas y los resultados obtenidos al aplicarlas.

En las conclusiones se presenta un resumen de lo obtenido durante el desarrollo del trabajo. Se presentan los resultados y contribuciones obtenidos, así como los trabajos a futuro.

El apéndice A describe la notación específica de la metodología INGENIAS.

El apéndice B contiene los modelos complementarios de las fases de análisis y diseño del Sistema Multi-agente desarrolladas en los capítulos cuarto y quinto respectivamente.

El apéndice C contiene la descripción de las tablas que conforman a la Base de Datos de Soporte descrita en el quinto capítulo.

El apéndice D contiene los resultados de las pruebas de caja negra y del sistema descritas en el sexto capítulo.

El apéndice E contiene el código para la integración del Sistema Multi-Agente con HeAP MoProSoft descrito en el sexto capítulo.

El hombre nada puede aprender sino en virtud de lo que ya sabe.

ARISTÓTELES

Capítulo 1

MoProSoft Y HeAP MoProSoft

Este capítulo muestra una panorámica general sobre los orígenes del presente trabajo. Para ello, se describe brevemente el Modelo de Procesos para la Industria del Software (MoProSoft) y la Herramienta para la Administración de Proyectos para MoProSoft (HeAP MoProSoft), así como la exposición del contexto que propicia las contribuciones propuestas.

1.1 MODELO DE PROCESOS PARA LA INDUSTRIA DE SOFTWARE

1.1.1 Antecedentes

En el año 2002 la Secretaría de Economía inició el Programa para el Desarrollo de la Industria de Software, PROSOFT [SE2001], que tiene como objetivo fortalecer a la industria de software en México. Dentro de la estrategias definidas está crear una norma que contemple: un modelo de procesos (qué procesos), un modelo de capacidades de procesos (qué evaluar) y un método de evaluación (cómo evaluar).

Como consecuencia de los acuerdos de la mesa de la Estrategia del PROSOFT y bajo un convenio con la Facultad de Ciencias de la Universidad Nacional Autónoma de México (UNAM), durante el 2002 fue desarrollado el Modelo de Procesos para la Industria del Software (MoProSoft). Para el desarrollo de este proyecto se convocó, a través de la Asociación Mexicana para la Calidad en Ingeniería de Software (AMCIS), a personas con experiencia y conocimiento en los modelos de procesos y calidad de software.

Para mayo del 2003 fue terminada la versión 1.1 y para agosto del 2005 la versión 1.3. El 15 de agosto del 2005 fue publicada, en el Diario Oficial de la Federación, la declaratoria de la Norma Mexicana NMX-059/01-NYCE-2005 referente a MoProSoft [GOB2005].

Para la segunda y tercera parte de la norma, se elaboraron el modelo de capacidades basado en la norma ISO/IEC 15504-2 y el método de evaluación *EvalProSoft* respectivamente.

1.1.2 Descripción general

El objetivo principal de MoProSoft es proporcionar a la industria mexicana, y a las áreas internas dedicadas al desarrollo y/o mantenimiento de software, un conjunto integrado de las mejores prácticas basadas en los modelos y estándares reconocidos internacionalmente. Para ello, se fundamenta en los modelos: Modelo de Madurez de Capacidades (*Capability Maturity Model*, CMM) v1.1, ISO 9001:2000, ISO/IEC TR: 15504-2:1998, Cuerpo de Conocimiento para la Administración de Proyectos (*Guide to Project Management Body of Knowledge, PMBoK)*, Cuerpo de Conocimiento para la Ingeniería de Software *(Guide to the Software Engineering Body of Knowledge, SWEBoK),* Proceso Personal de Software (*Personal Software Process*, PSP) y Proceso de Software en Equipo (*Team Software Process*, TSP).

Cabe señalar, que MoProSoft es aplicable a organizaciones que no cuenten con procesos establecidos (ajustándolo de acuerdo a sus necesidades), así como para las que ya tienen procesos establecidos (tomándolo como punto de referencia para identificar los elementos que les hace falta cubrir).

Dentro de los principales servicios que ofrece MoProSoft para cumplir con su propósito están:

- Obtener acceso a las prácticas de ingeniería de software de clase mundial.
- Mejorar la calidad del software producido por la empresa que adopta el modelo.
- Elevar la capacidad de las organizaciones para ofrecer servicios con calidad y alcanzar niveles internacionales de competitividad.
- Integrar todos los procesos de la organización y mantener la alineación con los objetivos estratégicos.
- Ofrecer el inicio a la adopción de los modelos ISO 9000 o CMMI.
- Implantar un programa de mejora continua.
- Reconocer a las organizaciones mexicanas por su nivel de madurez de procesos.
- Facilitar la selección de proveedores.

Ahora bien, las principales características de MoProSoft que le permiten ofrecer los servicios descritos anteriormente son: es específico para el desarrollo y mantenimiento de software, es sencillo de entender y adoptar (principalmente en organizaciones pequeñas con bajos niveles de madurez), facilita el cumplimiento de los requisitos de otros modelos, tiene un bajo costo (adopción y evaluación), resulta acorde con la estructura de las organizaciones mexicanas de la industria de software y está orientado a mejorar los procesos al contribuir con los objetivos de negocio (no simplemente ser un marco de referencia o certificación).

En cuanto a su estructura, propone un esquema dividido en tres categorías (Figura 1.1) que a su vez están conformadas por procesos.

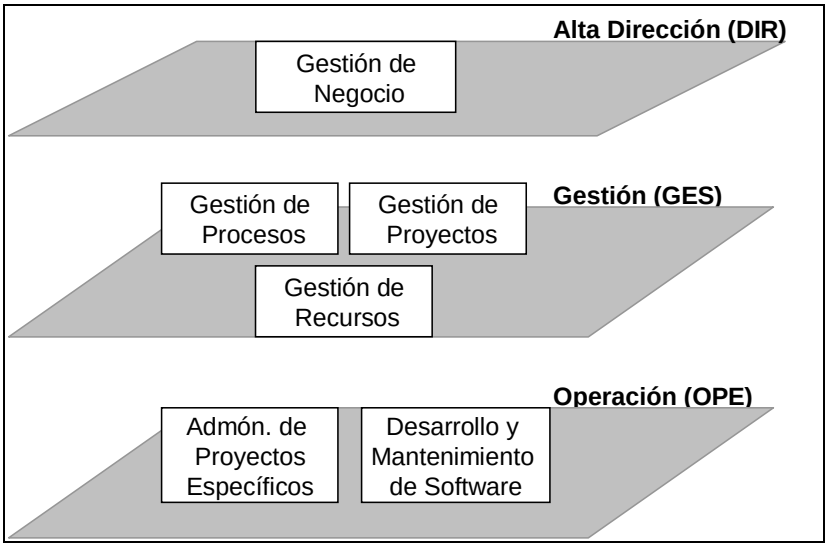


Figura 1.1: Estructura de MoProSoft.

A continuación se describe brevemente cada categoría.

- **Alta Dirección (DIR).** Esta categoría contiene el proceso de *Gestión de Negocio*. Mediante este proceso las organizaciones pueden establecer su razón social, objetivos, estrategias y ser habilitadas para responder ante situaciones de cambio. Adicionalmente, se proponen acciones para mantener una mejora continua, relación con los clientes, empleados y con la categoría de Gestión.

- **Gestión (GES***).* Está conformada por tres procesos: *Gestión de Procesos, Gestión de Proyectos y Gestión de Recursos*. Mediante las actividades y tareas que se definen en sus procesos se proporcionan los elementos necesarios en los procesos de la Categoría de Operación, se recibe y evalúa la información generada por éstos últimos y además se comunica los resultados a la Categoría de Alta Dirección.

    - En *Gestión de Procesos* se establecen los procesos de la organización y sus actividades de mejora.
    - En *Gestión de Proyectos* se asegura que los proyectos contribuyan al cumplimiento de los objetivos y estrategias de la empresa.
    - En *Gestión de Recursos* se manejan cuestiones de recursos humanos, infraestructura, ambiente de trabajo y proveedores, así como crear y mantener la *Base de Conocimiento* de la organización. Cabe señalar, que *Gestión de Recursos* se conforma de tres subprocesos: Recursos Humanos y Ambiente de Trabajo, Conocimiento de la Organización y Bienes, Servicios e Infraestructura.

- **Operación (OPE)**. Está integrada por dos procesos: Administración de Proyectos Específicos y Desarrollo y Mantenimiento de Software. Esta categoría ofrece las prácticas necesarias para el desarrollo y mantenimiento de los proyectos de la organización mediante los elementos proporcionados por la Categoría de Gestión.

    - En Administración de Proyectos Específicos se establece y ejecuta las actividades que permitan cumplir con los objetivos de un proyecto en el tiempo y costo esperados.
    - En Desarrollo y Mantenimiento de Software se manejan actividades como recolección de requerimientos, análisis, diseño, construcción, integración y pruebas.

Finalmente, mediante flujos de trabajo se integran las actividades, responsables, objetivos, indicadores y metas. Además, se indican entradas, salidas, productos internos así como actividades de verificación, validación y pruebas para mejorar su calidad.

1.2 HERRAMIENTA PARA LA ADMINISTRACIÓN DE PROYECTOS PARA MOPROSOFT

1.2.1 Antecedentes

Con la obtención de los primeros resultados de la implementación de MoProSoft, se observó que dentro de las recomendaciones hechas por las empresas de la industria mexicana de software está el contar con herramientas que apoyen su mejor comprensión e implementación. También, se evidenció el hecho de que las empresas no están llevando a cabo una buena administración de sus proyectos por lo que se solicitó un apoyo especial en esta área debido a su complejidad y falta de experiencia en algunos casos.

Para afrontar este reto, se decidió crear la Herramienta para la Administración de Proyectos para MoProSoft (HeAP MoProSoft) en el curso de Ingeniería de Software Orientado a Objetos del 2006 de la Maestría en Ciencia e Ingeniería de la Computación de la Universidad Nacional Autónoma de México (UNAM).

Para la construcción de esta herramienta, se plantearon tres temas de tesis complementarios entre sí. Se decidió que un tema de tesis abarcara la construcción de HeAP MoProSoft bajo el marco de trabajo “Struts”, y en base a este desarrollo, surgen los otros dos temas que desarrollan un sistema que al ser integrado a HeAP MoProSoft hará a la aplicación más robusta e innovadora a través del uso de tecnologías avanzadas.

A continuación se describe la herramienta HeAP MoProSoft a partir de la cual se desarrolla el Sistema Multi-Agente.

1.2.2 Descripción general

El sistema HeAP MoProSoft consiste en una aplicación web, que formará parte del software libre, para apoyar a las pequeñas y medianas empresas (PyMES) y áreas internas dedicadas al desarrollo y/o mantenimiento de software, en la administración de sus proyectos.

Mediante esta aplicación se permitirá la gestión de la información generada en los procesos de Gestión de Proyectos y Administración del Proyecto Específico de las categorías de Gerencia y Operación de MoProSoft respectivamente. Se proporciona plantillas con la información mínima requerida para realizar las tareas y los documentos necesarios en estos procesos. En consecuencia, se ofrece un apoyo para la administración de los proyectos de la organización y de sus documentos al apoyar a los roles de MoProSoft: Responsable de Gestión de

Proyectos (RGPY) y Responsable de la Administración del Proyecto Específico (RAPE).

1.2.2.1 Diagrama General de Casos de Uso

A continuación, se presenta en la figura 1.2 el diagrama general de casos de uso del sistema HeAP MoProsoft. El sistema cuenta con seis casos de uso y dos tipos de usuarios: Responsable de Gestión de Proyectos (RGPY) y Responsable de la Administración del Proyecto Específico (RAPE).

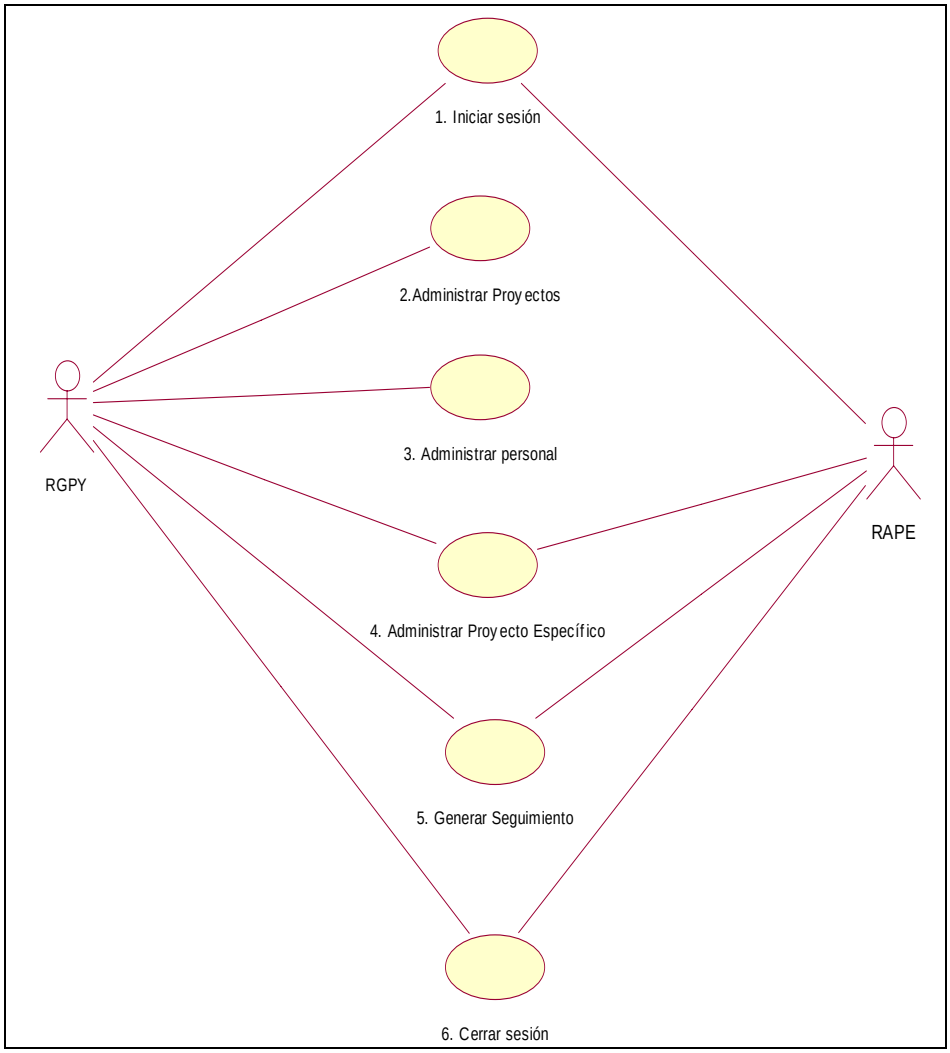


Figura 1.2: Diagrama general de casos de uso de HeAP MoProsoft.

1.2.2.2 Arquitectura

Para el desarrollo de HeAP MoProSoft se decidió utilizar el marco de trabajo “Struts”, con la finalidad de mejorar la comprensión y facilitar la integración de nuevos elementos al sistema. La arquitectura está dada por el patrón de diseño

MVC (Modelo, Vista, Controlador) que consiste en separar los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos, facilitando la modificación o personalización de cada parte [CAVANESS2004].

La funcionalidad y los elementos de cada componente de la arquitectura para HeAP MoProSoft (Figura 1.3) son:

- El componente Vista se ocupa de la representación del sistema y se conforma por la interfaz gráfica que se implementará con JSP y HTML.
- El componente Control maneja la funcionalidad del sistema, asignando las acciones a la Vista según su impacto en el Modelo. Está conformado por las clases definidas en Struts: *ActionServlet, RequestProcessor, Action, ActionForm*, entre otras.
- El componente Modelo se ocupa de los cambios de estado del sistema y tiene que ver con el almacenamiento de los datos: base de datos y las clases relacionadas. Se hace uso del patrón de diseño DAO (Data Access Object).

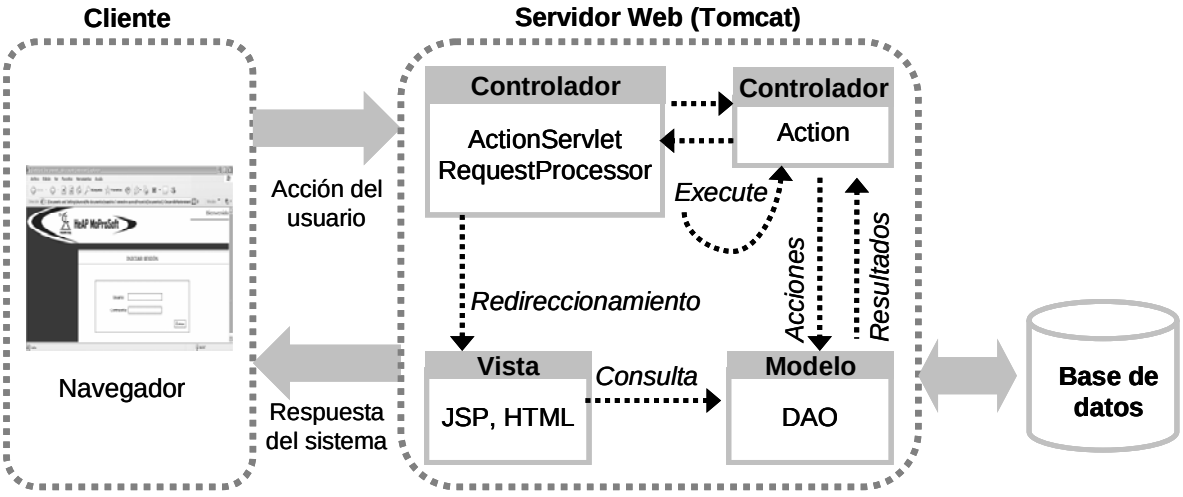


Figura 1.3: Arquitectura de HeAP MoProSoft.

La aplicación está en un servidor web (Tomcat) que es el que contiene las páginas, las reglas del negocio y el acceso a datos. Cada cliente puede acceder a ellos a través de su navegador Web.

Se ha proporcionado sólo la información necesaria de HeAP MoProSoft para la comprensión de los elementos y su relación con el presente trabajo. Para obtener una explicación más detallada sobre esta herramienta, consultar la tesis enfocada en su desarrollo [TORRES2007].

Los conceptos y principios fundamentales de la ciencia son invenciones libres del espíritu humano.

ALBERT EINSTEIN

Capítulo 2

Fundamentos de Agentes y Sistemas Multi-Agente

En este capítulo se presentan los conceptos fundamentales relacionados con el tema: Agentes y Sistemas Multi-Agente. Con ello se pretende ofrecer un conocimiento de la teoría que sustenta el presente trabajo.

2.1 INTRODUCCIÓN

Una de las abstracciones que surgió en el mundo informático como un nuevo concepto o tendencia hacia la construcción de sistemas de software complejos, son los *agentes.* Esta abstracción surge como respuesta a la necesidad de generar nuevos paradigmas que permitan modelar problemas de una forma cada vez más eficiente.

La teoría de agentes establece una serie de mecanismos y características que suelen tener aplicación en una gran cantidad de sistemas de hardware y software puesto que comparten una serie de características como la distribución de la información, el conocimiento parcial que tienen las entidades, la computación asincrónica, la ausencia de un sistema de control central y persistencia en el tiempo.

Cuando se considera una organización de agentes surge el concepto de Sistema Multi-Agente (SMA), el cual permite dar una visión integral al diseño y construcción de sistemas al permitir dar una solución eficiente y proporcionar un marco para la organización y el desarrollo de software.

Dado lo extenso que puede ser el tema, es necesario conocer las características esenciales de las entidades que integran este tipo de sistemas. Para cumplir con este propósito, a continuación se presenta la definición e integración de los conceptos fundamentales.

2.2 AGENTES

2.2.1 Teoría de agentes

Actualmente no existe una definición precisa de agente puesto que los investigadores utilizan distintas acepciones de dicho término y no existe una definición académica ampliamente aceptada, razón por la cual existe un gran número de definiciones. Las más citadas y completas que distintos autores han aportado, se muestran a continuación.

“Un agente es una entidad física o abstracta que puede percibir su ambiente a través de sensores, es capaz de evaluar tales percepciones y tomar decisiones por medio de mecanismos de razonamiento sencillos o complejos, comunicarse con otros agentes para obtener información y actuar sobre el medio en el que se desenvuelve a través de ejecutores” [RUSSELL1995].

“Sistemas computacionales que habitan en ambientes dinámicos complejos, sienten y actúan autónomamente en ese entorno y al hacerlo llevan a cabo un conjunto de objetivos o tareas para los que fueron diseñados” [MAES1995].

"Sistema computacional que está situado en un determinado entorno, y que es capaz de forma flexible y autónoma de efectuar acciones sobre ese entorno para alcanzar sus objetivos" [JENNINGS2000].

"Un agente inteligente realiza continuamente tres funciones: percepción de las condiciones dinámicas de un entorno, acción (que afecta a dichas condiciones) y razonamiento (para interpretar percepciones, resolver problemas, hacer inferencias y determinar acciones)" [HAYES1995].

"Un agente autónomo es un sistema situado dentro y como parte de un entorno, que percibe ese entorno y actúa en él a través del tiempo, a favor de su propia agenda así como del efecto que percibe en el futuro" [STAN1996].

Como se puede derivar de las definiciones anteriores, aunque no presentan una definición homogénea del concepto, si proponen aspectos como propiedades, habilidades u operaciones comunes. Por ello, habitualmente se suele recurrir a la descripción de un agente como una entidad que cumple una serie de características. Dichas características se suelen clasificar siguiendo dos definiciones: noción débil y noción fuerte de agente [WOOLDRIDGE1995].

La *noción débil* es por lo general usada en el campo de la Ingeniería del Software Orientada a Agentes (AOSE – *Agente Oriented Software Engineering*) [WOOLDRIDGE2001]. Como características mínimas en esta noción se tienen: autonomía, reactividad, iniciativa y sociabilidad.

- La autonomía se refiere a que el agente actúa sin intervención humana u otros agentes.
- La reactividad implica que el agente percibe su entorno y reacciona a los cambios que puedan ocurrir en él.
- La iniciativa (pro-actividad) proporciona que el agente sea capaz de tomar la iniciativa para realizar tareas que lo acerquen hacia sus objetivos.
- La sociabilidad permite que los agentes interactúen con su entorno, es decir, con otros agentes o humanos vía algún tipo de lenguaje de comunicación.

La *noción fuerte* añade ciertas características a la noción débil: movilidad, veracidad, benevolencia, racionalidad y aprendizaje.

- La movilidad se refiere a la capacidad del agente para moverse a través de una red electrónica.
- La veracidad proporciona la seguridad de que el agente no transmitirá información falsa conscientemente.
- La benevolencia permite asumir que el agente hará aquello para lo que se creó.

- La racionalidad proporciona la seguridad de que el agente intentará alcanzar las metas que le fueron asignadas de la mejor forma posible, basándose en la información que posee del mundo exterior.
- El aprendizaje o adaptabilidad permite al agente que en base a la experiencia adquirida, se adapte para adecuarse al entorno.

Por último, para nuestro fin se tiene la siguiente definición que reúne e integra las acepciones expuestas sobre lo que puede ser considerado como un agente:

Sistema computacional que está situado dentro y como parte en un determinado entorno, y que es capaz de forma flexible y autónoma de comunicarse y efectuar acciones sobre ese entorno para alcanzar sus objetivos de diseño.

2.2.2 Arquitecturas de agentes

Hasta el momento, se han visto los conceptos para representar las propiedades de los agentes. Ahora bien, cuando se considera cuestiones que rodean a la construcción de sistemas informáticos que satisfagan las propiedades especificadas surge el área de arquitecturas de agentes.

Las arquitecturas especifican cómo se descomponen los agentes en un conjunto de módulos que interactúan entre sí para lograr la funcionalidad requerida, así como la descripción de la interconexión entre estos módulos. En otras palabras, las arquitecturas de agentes, permiten elaborar el diseño de cómo construir sistemas que satisfagan las propiedades que se han especificado teóricamente a los agentes y qué estructuras de software y/o hardware son apropiadas.

Existen una gran variedad de arquitecturas, es por ello, que a continuación se presenta una clasificación que en la actualidad cubre el espectro de posibilidades que ofrece esta tecnología y se basa según el tipo de procesamiento empleado [WOOLDRIDGE1995]: deliberativas, reactivas e híbridas.

2.2.2.1 Deliberativas

Estas arquitecturas utilizan modelos de representación simbólica del conocimiento, en donde las decisiones se toman utilizando mecanismos de razonamiento lógico basados en la correspondencia de patrones y la manipulación simbólica, con el propósito de alcanzar los objetivos del agente.

Generalmente se basan en la teoría clásica de planificación de inteligencia artificial en la cual, a partir de un estado inicial, un conjunto de operadores/planes y un estado objetivo, la deliberación del agente consiste en determinar qué pasos

debe encadenar para lograr su objetivo, siguiendo un enfoque descendente (top-down). Estos agentes expresan la información sobre el entorno y su comportamiento (estado interno y acciones) en forma de conocimiento simbólico.

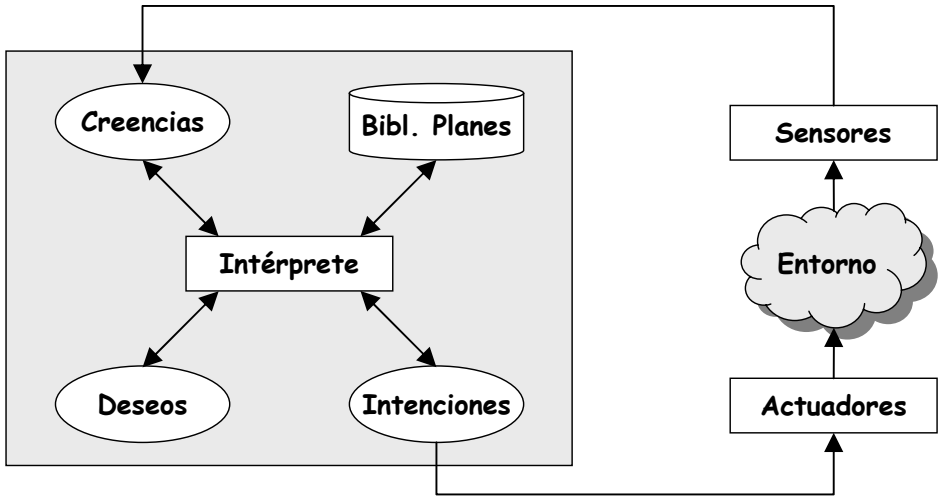


Figura 2.1: Elementos utilizados por un agente deliberativo.

Existen dos tipos principales de estas arquitecturas simbólicas: arquitecturas intencionales y arquitecturas sociales. En cuanto a las intencionales, los agentes se distinguen por su capacidad de razonar sobre sus creencias e intenciones y pueden ser considerados como sistemas de planificación que incluyen creencias e intenciones en sus planes. Con respecto a las sociales, los agentes pueden definirse como agentes intencionales que tienen un modelo explícito de otros agentes y son capaces de razonar sobre estos modelos.

Finalmente, las arquitecturas deliberativas también se pueden ubicar bajo la clasificación de horizontales debido a que los estímulos recibidos del exterior son procesados en varias capas de diferente nivel de abstracción y el nivel superior decide al final las acciones a realizar.

2.2.2.2 Reactivas

Las arquitecturas reactivas cuestionan la viabilidad del paradigma simbólico por lo que proponen un enfoque conductista con un modelo estímulo-respuesta. Se caracterizan por no tener como elemento central de razonamiento un modelo simbólico y por no utilizar razonamiento simbólico complejo [BROOKS1991], sino que siguen un procesamiento ascendente (bottom-up), para lo cual mantienen una serie de patrones que se activan bajo ciertas condiciones de los sensores y tienen un efecto directo en los actuadores.

Dentro de las principales arquitecturas reactivas están las reglas situadas, las arquitecturas incluidas (*subsumption*), los autómatas de estado finito, las tareas competitivas y las redes neuronales [MAS2005]. Estas arquitecturas

pueden clasificarse como verticales ya que los estímulos recibidos del exterior son procesados por capas especializadas que directamente responden con acciones a dichos estímulos y pueden inhibir las capas inferiores (figura 2.2).

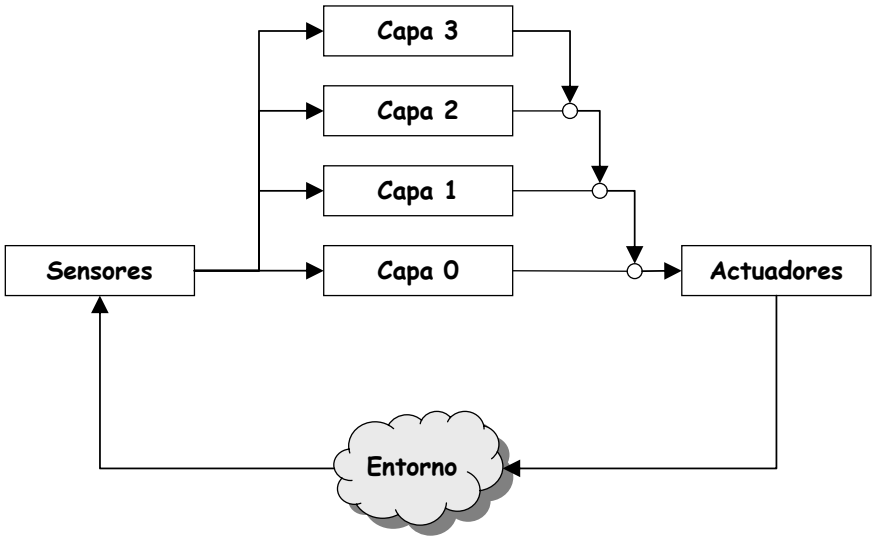


Figura 2.2: Arquitectura incluida de un agente reactivo.

2.2.2.3 Híbridas

Estas arquitecturas pretenden combinar aspectos deliberativos y reactivos, por lo que pueden ser horizontales y verticales. Los módulos reactivos se encargan de procesar los estímulos que no necesitan deliberación, mientras que los módulos deliberativos determinan qué acciones deben realizarse para satisfacer los objetivos locales y cooperativos de los agentes.

2.2.3 Tipologías de agentes

De lo expuesto en los párrafos anteriores, uno de los puntos mas controvertidos en la comunidad de investigadores de esta rama ha sido la definición de agentes y su clasificación. Para esto se han tenido en cuenta criterios como sus capacidades, sus funciones, su arquitectura, su forma de organización, entre otros. Independientemente a esta disparidad de criterios, se coincide en que un agente es un componente de software y/o hardware que es capaz de actuar sobre su entorno para llevar a cabo sus objetivos de diseño.

En cuanto a tipos de agentes, una de las primeras clasificaciones propuestas es la que propone agentes reactivos y cognitivos [FERBER1999]. Desde entonces, han surgido múltiples tipologías en base a las propiedades que definen a los agentes [JENNINGS1998] como se muestra a continuación.

- *Reactivo (percibe y actúa):* responde en tiempo a los cambios en el ambiente.
- *Autónomo:* tiene control sobre sus propias acciones.
- *Orientado a objetivos (pro-activo, con propósitos):* No actúa simplemente en respuesta al ambiente, sino a objetivos.
- *Basado en utilidad*: actúa en función de un valor (utilidad) que define el objetivo prioritario si los objetivos se encuentran en conflicto o se puede alcanzar más de un objetivo al mismo tiempo.
- *Temporalmente continúo:* es un proceso continuo.
- *Comunicativo (sociable):* se comunica con otros agentes y tal vez incluyendo personas.
- *Colaborativo o cooperativo*: se relaciona con otros agentes para que los procesos independientes de cada uno sean enfocados a ofrecer cierta funcionalidad.
- *Que aprende (adaptable):* adapta su comportamiento de acuerdo a experiencias previas.
- *Móvil:* capaz de transportarse de una máquina a otra.
- *Flexible:* las acciones no están preestablecidas.
- *Con carácter:* cuenta con personalidad y estado emocional.

Dentro de todas estas clasificaciones surgen los denominados *agentes de software* cuya principal diferencia con los ya mencionados es la especialización de su diseño, es decir, realizan un tipo en específico de tarea y se caracterizan por su proactividad, autonomía, aprendizaje o cooperativismo. Este tipo de agentes tienen una mención especial en los siguientes párrafos debido a su utilización en el proyecto de tesis.

Un agente de software se caracteriza por ser autónomo y comunicativo, además posee recursos propios y servicios que ofrecen a otros agentes por lo que tiene una representación parcial de éstos. Su comportamiento atiende a sus objetivos considerando los recursos y habilidades disponibles, sus representaciones y las comunicaciones que recibe [FERBER1999].

A continuación, se mencionan los tipos de agente de software más conocidos e importantes [NWANA1996].

- Los a*gentes de interfaz* (*asistentes personales* o *agentes de usuario)* tienen como objetivo simplificar las tareas rutinarias que realiza un usuario. Se caracterizan por ser flexibles, adaptables al medio, se basan en la delegación y actúan por iniciativa propia. Dentro de las tareas que suelen ayudar al usuario están las repetitivas o habituales, por ejemplo, aprender a filtrar el correo electrónico con base en el comportamiento habitual del usuario; planificar y negociar con otros asistentes personales para la realización de citas; detectar cuando una noticia puede ser relevante para un usuario y comunicárselo.

- Los *agentes móviles* entran en el paradigma de la computación distribuida y su movilidad se refiere al hecho de que puede transitar entre varias máquinas con la finalidad de evitar una sobrecarga de comunicación o utilizar recursos de los que no dispone en su máquina [KLUSCH2001].

- Los *agentes de internet/información* surgen de la necesidad de procesar de manera automática la información por lo que tienen la misión de navegar por la red recolectando información, indexarla y ofrecer la que puede interesar al usuario cuando realiza una consulta [JULIAN2001].

2.2.4 Lenguajes de agentes

Se define a un lenguaje de agente [WOOLDRIDGE1995] como un lenguaje que permite programar sistemas de hardware o de software en términos de algunos de los conceptos desarrollados por las teorías de agentes. Existe dos tipos principales de lenguajes de programación: lenguajes de agentes de propósito general y los lenguajes de agentes específicos. Los primeros son aquellos que están destinados a programar agentes genéricos utilizables en cualquier aplicación. Los segundos, como su nombre lo dice, son para cierto tipo de agentes específicos tales como los móviles.

Ahora bien, desde el punto de vista de la programación de agentes se manejan los siguientes tres niveles:

- Los *lenguajes de programación de la estructura del agente* permiten programar las funcionalidades básicas para definir a un agente como de creación de procesos y de comunicación entre agentes. Los lenguajes empleados pueden ser de propósito general (C, C++, Java, Lisp, Prolog, entre otros) o específicos (April,y CUBL).

- Los *lenguajes de comunicación de agentes* permiten definir el formato de los mensajes intercambiados, de las primitivas de comunicación y de los protocolos disponibles. Algunos lenguajes de este tipo son el Perl, Pitón, Tcl, FIPA ACL y KQML.

- Los *lenguajes de programación del comportamiento del agente* permiten definir lo referente al conocimiento del agente, es decir aspectos como el conocimiento inicial (modelo de entorno, creencias, deseos, objetivos), funciones de mantenimiento de dicho conocimiento (reglas, planes, etc.), funciones para alcanzar sus objetivos (planes, reglas) y funciones para desarrollar habilidades (programación de servicios). Dentro de los diferentes lenguajes de descripción de agentes existentes están: lenguajes de descripción de agente (MACE ADL, AgentSpeak y MAST/ADL), de programación orientados a agentes (AGENT0, PLACA y Agent-K), basados en reglas de producción (MAGS, DYNACLIPS y RTA) y de especificación (METATEM y DESIRE).

2.2.5 La naturaleza del entorno de agentes

De lo expuesto anteriormente, con respecto a la teoría de agentes, se menciona el término *entorno* o *ambiente*, el cual debe ser considerado para realizar un mejor diseño ya que dependiendo del tipo de entorno se define la percepción del agente, sus acciones y su control.

Uno de los primeros pasos para el diseño de un agente es la especificación del entorno de forma completa, sin embargo, esta tarea no es sencilla ya que implica categorizar el rango de los entornos de trabajo. Por ello, en la tabla 2.1 se muestra la clasificación de los entornos considerando cierta propiedad desde el punto de vista del agente [RUSSELL1995].

Propiedad	**Significado**
Totalmente observable	Cuando los sensores del agente detectan todos los aspectos que son relevantes en la toma de decisiones y proporcionan acceso al estado completo del medio en cada momento.
Parcialmente observable	Cuando los sensores son inexactos o que un sensor no reciba información sobre parte del sistema puede hacer que no se disponga de toda la información para tomar la decisión.
Determinista	Cuando el siguiente estado del medio está totalmente determinado por el estado actual y la acción ejecutada por el agente. Si el medio es determinista, excepto para las acciones de otros agentes, decimos que el medio es *estratégico*.
Estocástico	Cuando el siguiente estado del medio no está determinado por el estado actual y la acción ejecutada por el agente.
Episódico	Cuando la experiencia del agente se divide en episodios atómicos, los cuales consisten en la percepción del agente y una única acción posterior no dependiente de otras acciones realizadas previamente.
Secuencial	Cuando la decisión presente puede afectar a decisiones futuras.
Dinámico	Cuando el entorno puede cambiar cuando el agente está deliberando.
Semi-dinámico	Cuando el entorno no cambia con el paso del tiempo, pero el rendimiento del agente cambia.
Estático	Cuando el entorno no puede cambiar.
Discreto	Cuando existe un conjunto finito de variables a observar y un conjunto finito de acciones o estados posibles. Por ejemplo, un medio con estados discretos es el del juego de ajedrez que tiene un número finito de estados distintos
Continuo	Cuando no existe un conjunto finito de variables, acciones o estados posibles. Por ejemplo, un taxista conduciendo define un estado continuo.
Agente individual	Cuando sólo un agente interactúa con el entorno.
Multi-Agente	Cuando existen activos diversos agentes autónomos independientes que se dedican a sus propios objetivos y sólo contactan con los otros agentes para obtener información, o para contribuir a una solución coordinada de un problema general.

Tabla 2.1: Clasificación de entornos de agentes.

2.3 SISTEMAS MULTI-AGENTES

2.3.1 Concepto de Sistema Multi-Agente

Un *Sistema Multi-Agente* (SMA) consiste en un conjunto (dos o más) de agentes autónomos, generalmente heterogéneos y potencialmente independientes, que trabajan entre sí (colaborando o compitiendo) para resolver un problema. Algunas características básicas de estos sistemas son [JENNINGS1998]:

- Cada agente tiene una vista limitada del estado del mundo por lo que se dice que tiene información incompleta.

- El control global del sistema está distribuido, la información está descentralizada y la computación es asíncrona.

Como se puede observar la perspectiva de estos sistemas está centrada en la interacción entre los agentes y para que ésta sea exitosa requieren habilidades para cooperar, coordinar y negociar con los demás.

Por otra parte, suelen contar con heterogeneidad de agentes, es decir, los agentes pueden contar con arquitecturas diferentes, utilizar diferentes representaciones internas, comunicarse con diferentes lenguajes, etc. Otra característica importante es la homogeneidad de intereses, ya que en base a ésta se puede tener dos situaciones [DURFEE1994]:

- *Cooperación:* los agentes son capaces de combinar sus esfuerzos, normalmente para obtener soluciones de problemas que el propio agente, por sí mismo, no es capaz de encontrar (CMAS "Cooperative Multiagent Systems").

- *Competición*. Los agentes sólo cumplen sus objetivos (SMAS "Self-Interested Multiagent Systems"), por lo que son importantes las estrategias de negociación que permiten resolver conflictos, asignar tareas y tomar decisiones.

Finalmente, existen conceptos de gran interés por resolver u optimizar en los Sistemas Multi-Agentes: actividades conjuntas y cooperación, conflictos (resolución), negociación, compromisos y planificación de actividades, modelo del conocimiento y su comunicación, entre otros [LOPEZ2005].

2.3.2 Arquitecturas Multi-Agente

Una arquitectura Multi-Agente debe proveer un marco de desarrollo que permita la cooperación entre agentes heterogéneos. Dentro de las arquitecturas más importantes de los SMA están:

- *Arquitectura RETSINA* (*Reusable Environment for Task-Structured Intelligent Network Agents*). Es una arquitectura creada en el Instituto de Robótica de la Universidad de *Carnegie Mellon* cuya aplicación puede ser para el desarrollo de agentes cooperativos [SYCARA2003]. El entorno de RETSINA se ha desarrollado sobre la base de que los agentes forman una comunidad en la que todos ellos están al mismo nivel y sin imponer restricciones, a nivel de infraestructura, en las comunicaciones. La infraestructura de servicios de RETSINA impide la existencia de un control centralizado y los agentes buscan activamente información y cooperan para realizar tareas de recolección e integración de información.

- *Arquitectura FIPA (Foundation for Intelligent Physical Agents):* Esta arquitectura fue creada por una organización no lucrativa cuya misión principal era desarrollar estándares y facilitar la interoperación entre agentes a través de múltiples sistemas de agentes heterogéneos [FIPA2003]. FIPA proporciona la infraestructura para el desarrollo y uso de agentes, donde la idea es que en cada implementación concreta se tomen las decisiones relativas a las componentes usadas para desarrollarla, pero manteniendo siempre un conjunto de interfaces externas que permitan la interoperabilidad entre plataformas.

2.3.3 Metodologías de desarrollo de Sistemas Multi-Agente

Para la construcción de un SMA se debe conocer aspectos de los agentes como los resultados que producirán, qué actividades son necesarias para producirlos, cómo realizarlas y con qué herramientas (desarrollo y entorno de ejecución). Debido a la complejidad que representa definir lo anterior, se han emprendido numerosos esfuerzos para generar metodologías basadas en agentes.

Estas metodologías abarcan tanto la definición de procesos, actividades, resultados a producir, lenguajes para describir dichos resultados, guías, métricas y herramientas asociadas, siguiendo un paradigma de ingeniería de software y de agentes. En resumen, parten de un modelo de cómo debe ser el SMA y describen métodos y guías para su construcción. En la tabla 2.3 se muestra las principales propuestas metodológicas.

Metodología	**Descripción**
AAII/BDI	• Desarrollada por el Instituto de Inteligencia Artificial Australiano (AAII) en 1996 [KINNY1996]. • Considera dos puntos de vista: externo (identificación de agentes y sus interacciones) e interno (basado en el modelo BDI, se describe el comportamiento de cada agente). • El desarrollo del SMA es guiado por la elaboración y refinamiento de los puntos de vista. El proceso consta de varias iteraciones, se

Metodología	Descripción
	considera primero el externo seguido por el interno (retroalimenta al externo) y así sucesivamente hasta conseguir el nivel detalle necesario para la implementación.
Ingeniería de las Vocales	• Propone cinco puntos de vista correspondientes a las vocales (Agente, Entorno, Interacciones, Organización y Usuario) [RICORDEL2001]. • Las vocales (aspectos) tienen cierto orden dependiendo de la clase del sistema a desarrollar. • El propósito es considerar librerías de componentes que den soluciones a cada aspecto, por lo que se puede instanciar un modelo de agente particular. • La plataforma Volcano (orientada a componentes) aplica las ideas de esta metodología.
MAS-CommonKADS	• Hace uso de una notación orientada a objetos para estructurar el SMA, definición de casos de uso para la captura de requerimientos y técnicas de ingeniería de protocolos para especificar las interacciones entre agentes (SDL y diagramas de secuencias de mensajes [IGLESIAS1998]. • Especificación del sistema mediante los modelos de: agente, tareas, experiencia (o conocimiento), organización de la sociedad de agentes, comunicación con el usuario y coordinación.
MASSIVE	Propone siete puntos de vista: entorno, tareas, roles, interacciones, sociedad, arquitectura y sistema [LIND2000].
ODAC	Utiliza los cinco puntos de vista del estándar Open Distributed Processing (ODP): empresa, información, computacional, tecnología e ingeniería [GERVAIS2003].
MESSAGE	• Considera cinco puntos de vista e incorpora técnicas de Ingeniería de Software (se basa en el Proceso Unificado de Desarrollo de Software y un enfoque de refinamiento) con los puntos clave de la teoría de IA en cuanto a agentes y SMA [CAIRE2001]. • Provee un lenguaje de modelado basado en UML, un método, y guías de cómo aplicar todo, centrándose en las fases de análisis y diseño.
INGENIAS	• Es una extensión de los meta-modelos de MESSAGE y su validación en base a la experimentación en varios casos de estudio [PAVÓN2003]. • Se apoya en un conjunto de herramientas (INGENIAS Development Kit) construidas bajo los meta-modelos.
GAIA	• El SMA es considerado como una sociedad u organización, la cual consta de una colección de roles que son definidos bajo un modelo [WOOLDRIDGE2000]. • El análisis es basado en roles en interacción y el diseño en agrupación de roles en agentes. • Sólo define el modelo computacional por lo que no considera la implementación del sistema.

Metodología	Descripción
MaSE	• Adopta el paradigma de orientación a objetos y considera a los agentes como una especialización de los objetos la cual consiste, en la coordinación de los agentes a través de conversaciones y a su naturaleza preactiva [DELOACH2001]. • Considera todo un ciclo de desarrollo, realiza un análisis basado en roles y es soportado por la herramienta agentTool.

Tabla 2.2: Metodologías basadas en agentes.

2.3.4 Áreas de aplicación de los Sistemas Multi-Agente

La tecnología de Sistemas Multi-Agente (SMA) está realizando importantes aportaciones en la resolución de problemas donde las aproximaciones tradicionales no proporcionan soluciones suficientemente satisfactorias. En concreto, uno de los dominios donde esta tecnología proporciona una forma natural para resolver problemas es cuando son inherentemente distribuidos. En la tabla 2.4 se presenta un panorama de las principales áreas de aplicación de soluciones basadas en agentes.

Área	Aplicaciones
Industrial	Manufactura Robots clasificadores de piezas Control de procesos Telecomunicaciones y Control de tráfico aéreo Descentralización del control y gestión de redes Diseño de productos
Comercial	Recuperación y extracción de información Comercio electrónico Administración de procesos de negocio Buscadores e identificadores de recursos humanos Asistente de correo electrónico Mercado de servicios electrónicos Equipos móviles y PCs en el hogar
Entretenimiento	Juegos y entretenimiento móvil Teatro y cine interactivo Asistentes personales de viaje Chatterbots (agentes conversacionales que funcionan mediante el diálogo)
Medicina	Diagnóstico médico Monitoreo de pacientes Gestión de transplantes de órganos y tejidos Acceso a la información médica
Economía	Sistemas de negociación
Sociología	Modelado del comportamiento en organizaciones humanas

Área	**Aplicaciones**
Inteligencia Artificial	Resolución de problemas complejos por cooperación Construcción de aplicaciones concurrentes

Tabla 2.3: Algunas aplicaciones de los Sistemas Multi-Agentes por áreas.

La formulación de un problema, es más importante que su solución.

ALBERT EINSTEIN

Capítulo 3

Planteamiento del problema y propuesta de solución

En este capítulo se presenta a profundidad el planteamiento del problema y la solución propuesta con el fin de dotar al lector del conocimiento completo del contenido y propósito del proyecto de tesis. Adicionalmente, como parte importante de la solución, se describen las metodologías de trabajo y tecnologías utilizadas.

3.1 PLANTEAMIENTO GENERAL DEL PROBLEMA

Con base en la experiencia recabada durante la implementación de MoProSoft en ciertas empresas de la industria mexicana de software, se observó que dentro de las recomendaciones hechas está el ofrecer una consultoría, artefactos y ejemplos que aceleren dicha implementación [GONZÁLEZ2006].

Por otro lado, las pruebas controladas de MoProSoft y EvalProSoft en cuatro empresas durante el año 2005 mostraron que se alcanzaron los objetivos en promedio satisfactoriamente; sin embargo, dentro de los procesos que obtuvieron un nivel de madurez menor al objetivo fueron: Administración de Proyectos Específicos y Desarrollo y Mantenimiento de Software [PRUEBAS2005].

Es por ello, que un punto muy particular en cuanto a recomendaciones está orientado a la parte de administración de proyectos puesto que, se evidenció el hecho de que las organizaciones no realizan una buena administración de sus proyectos.

Para aminorar esta problemática, surge en el curso de Ingeniería de Software Orientado a Objetos del 2006 de la Maestría en Ciencia e Ingeniería de la Computación de la Universidad Nacional Autónoma de México (UNAM), la propuesta de construir una Herramienta para la Administración de Proyectos según MoProSoft (HeAP MoProSoft) diseñada específicamente para apoyar la implementación de la administración de proyectos.

Con el fin de obtener una aplicación que cumpliera con las expectativas identificadas, se decidió ampliar la herramienta con algunas innovaciones tecnológicas a través de trabajos complementarios.

Se planteó en primer lugar, el construir a HeAP MoProSoft bajo una tecnología que fuera exitosamente utilizada en la industria para la creación de aplicaciones web y de distribución libre. Además, debería proporcionar plantillas con la información mínima requerida para realizar las tareas y documentos necesarios para los procesos de MoProSoft relacionados a la administración de proyectos. El desarrollo de este trabajo es el objetivo central de la tesis *Herramienta de Administración de Proyectos para MoProSoft* [TORRES2007].

Con base en el trabajo anterior, se plantearon otras dos propuestas que desarrollarían una funcionalidad complementaria y que debería ser integrada a HeAP MoProSoft.

Una funcionalidad requerida consistió en anexar cierta tecnología computacional que facilitara la actividad de *Evaluación y Control* del proceso de *Gestión de Proyectos* en HeAP MoProSoft, mediante la cual, el usuario podrá llevar el control de los proyectos al saber donde se encuentra ubicado su desarrollo y tomar decisiones con el fin de alinear los proyectos a la estrategia de negocio. El desarrollo de esta funcionalidad es el objetivo central de la tesis

Sistema de tableros de control para gestión de proyectos para MoProSoft [CRUZ2007].

La otra propuesta debería agregar a HeAP MoProSoft la consulta de información especializada que ayudara y guiara al usuario en el aprendizaje e implantación de la administración de proyectos definida por MoProSoft. Esta información deberá guiar en la realización de las tareas comprendidas en el proceso y que estén implementadas en HeAP MoProSoft. El desarrollo de esta funcionalidad es el objetivo de esta tesis.

3.2 SOLUCIÓN PROPUESTA

Para cumplir con la expectativa de guiar al usuario en el aprendizaje e implantación de la administración de proyectos definida por MoProSoft, se propone la construcción de un sistema semi-independiente a HeAP MoProSoft que cumpla con la funcionalidad complementaria de consulta de información guía y que además haga uso de una innovación tecnológica con la finalidad de que sea una contribución importante al desarrollo de sistemas de este tipo.

La asistencia proporcionada al usuario mediante esta información especializada, implica el apoyo en las tareas involucradas para que le sea más sencillo y rápido entenderlas, aprenderlas y aplicarlas.

3.2.1 Objetivos

En este contexto, el objetivo general de la presente tesis es desarrollar un Sistema Multi-Agente, que será integrado al sistema HeAP MoProSoft, que brinde información mediante sugerencias, ejemplos o avisos para guiar al usuario en las tareas que realiza en la herramienta de acuerdo a lo definido en la administración de proyectos por MoProSoft.

Para alcanzar este objetivo general de la tesis, es necesario concretarlo en unos objetivos específicos, sintetizados en los siguientes puntos:

1. Se plantea *aplicar técnicas de las áreas de Ingeniería de Software e Inteligencia Artificial* con el fin de desarrollar una herramienta que contribuya a la demostración práctica de ofrecer mejores funcionalidades en sistemas de software. Para lo cual, se debe realizar un estudio de las metodologías y herramientas adecuadas para el desarrollo.

2. Se persigue *desarrollar un sistema* que al ser incorporado a HeAP MoProSoft, *permita aumentar la eficiencia y conocimiento del usuario* que participa en la administración de proyectos, dentro de una empresa. En este sentido, se considera conveniente realizar un análisis crítico de que funcionalidades son necesarias para lograr la aportación esperada.

3. Se pretende *ofrecer el servicio de monitoreo y tutoría de las tareas del usuario* con base en las acciones e información dadas. Para ello, se estudiará las acciones posibles a monitorear y la información que sea realmente de ayuda para los usuarios al realizar sus tareas.

4. Se busca *analizar los fundamentos teóricos de Agentes/Sistemas Multi-Agente* para determinar los elementos necesarios para dar la guía requerida.

3.3 MARCO METODOLÓGICO Y TECNOLÓGICO

Para poder llevar a cabo los objetivos planteados en el proyecto de tesis es necesario establecer un conjunto de metodologías, es decir, la aplicación de ciertos métodos que permitan alcanzar el resultado deseado. Así mismo, es importante poder contar con el soporte tecnológico (entornos y herramientas) adecuado, que facilite el desarrollo del sistema requerido. En los siguientes apartados se explican propuestas de metodologías y tecnologías para el desarrollo de la tesis y para el Sistema Multi-Agente.

Desarrollo de la tesis

Con la finalidad de llevar a cabo un ciclo de desarrollo de software de calidad, de tener un orden en las actividades a realizar y generar una documentación adecuada para consultas o modificaciones posteriores; durante el desarrollo de este proyecto se consideraron los siguientes flujos de trabajo del estándar de desarrollo Orientado a Objetos llamado Proceso Unificado de Desarrollo de Software (PU) [JACOBSON2000]: administración del proyecto, administración de la configuración, definición de requerimientos, análisis, diseño, implementación y pruebas.

Desarrollo del Sistema Multi-Agente

Una vez que se han establecido los objetivos que debe satisfacer el Sistema Multi-Agente, para poder llevar a cabo su construcción es necesario utilizar una metodología de desarrollo de agentes. Como se mencionó en el capítulo 2, existen investigaciones que tratan de presentar metodologías propias para el desarrollo de Sistemas Multi-Agente, a partir de modificaciones de propuestas existentes y otras que presentan concepciones completamente nuevas para desarrollo de software con estas características.

Puesto que aún no existe un estándar de facto para el desarrollo de este tipo de sistemas, se ha seleccionado a INGENIAS [GRASIA2005] con base en los siguientes criterios dentro del análisis comparativo de las metodologías existentes: utilización de diferentes vistas para la especificación del sistema, incorporar la idea de proceso de desarrollo, e integrar técnicas de ingeniería y teoría de agentes.

Dentro de las características que hacen a esta metodología la más conveniente para el desarrollo del Sistema Multi-agente están:

- Cubre la mayor parte de las características que las metodologías basadas en agente deben cubrir.

- Proporciona un análisis y diseño de manera completa mediante vistas o modelos con gráficos.

- Mediante el lenguaje de especificación del SMA (meta-modelos) se estudian qué actividades y productos constituyen la especificación del sistema y se integra dentro de un proceso de ingeniería del software.

- Utiliza un conjunto de herramientas, el INGENIAS Development Kit (IDK), para modelar, verificar y generar el código de los agentes.

- Cuenta con ejemplos de desarrollo y tiene guías para orientar en las demás etapas de desarrollo.

En cuanto al aspecto tecnológico, INGENIAS propone principalmente dos herramientas: el entorno de especificación y el generador de código. Para el entorno de especificación se hace uso de la herramienta llamada el editor gráfico y para al generador de código, una herramienta que se está desarrollando, por lo que se optó utilizar la herramienta de modelado INGENIAS IDK para documentar las fases de análisis y diseño y como plataforma de implementación el Marco de trabajo en Java para el Desarrollo de Agentes (*Java Agent Development Framework, JADE*).

En los siguientes apartados se describe a mayor detalle la metodología y las herramientas a utilizar para el desarrollo del Sistema Multi-Agente.

3.3.1 INGENIAS

3.3.1.1 Descripción general

INGENIAS es una metodología desarrollada por el grupo GRASIA de la Universidad Complutense de Madrid (UCM), que extiende la metodología MESSAGE [CAIRE2001] y proporciona un conjunto de herramientas para modelar y generar código de Sistemas Multi-Agente [PAVÓN2003].

Propone un lenguaje de especificación de Sistemas Multi-Agente así como su integración en el ciclo de vida al definir un conjunto de entregas y actividades involucradas en el desarrollo. Existe también un conjunto de herramientas de soporte (INGENIAS Development Kit, IDK) [INGENIAS2005].

El lenguaje de especificación se define por un conjunto de meta-modelos con los que hay que describir el sistema: agentes aislados, organizaciones de agentes, el entorno, interacciones entre agentes o roles, tareas y objetivos. Estos meta-modelos se construyen mediante un lenguaje de meta-modelado, el GOPRR (*Graph, Object, Property, Relationship, and Role*) [LYYTINEN1999].

Para realizar instancias de estos meta-modelos se define un conjunto de actividades cuya ejecución termina en un conjunto de modelos. Estas actividades a su vez se organizan siguiendo un paradigma de Ingeniería de Software, el Proceso Unificado de Desarrollo de Software [JACOBSON2000]. También permite validar y verificar el SMA al aplicar la Teoría de la Actividad [LEONTIEV1978] mediante la comprobación del modelo software e intentar detectar contradicciones y corregirlas en la medida de lo posible.

En cuanto a la implementación de un modelo de SMA se puede realizar sobre distintos tipos de plataforma de agentes o sobre un entorno de objetos tradicional. Además, la metodología facilita y promueve el desarrollo de herramientas de generación de código que faciliten el paso del modelo (análisis y diseño) a la implementación. También cuenta con la característica de ser adaptable a nuevos lenguajes y estándares (por ejemplo AUML).

Cabe mencionar que esta metodología sigue refinándose puesto que el lenguaje de especificación de SMA que propone descansa sobre resultados de investigación. Por ello, se ha habilitado diferentes líneas de desarrollo que proponen innovaciones a los resultados ya presentados.

Por último, la documentación que conforma a esta metodología es muy completa puesto que implica una descripción del lenguaje de especificación, la definición del ciclo de vida mediante la integración con cada una de las etapas del PU y las diferentes actividades requeridas para generar modelos, ejemplos completos de especificación siguiendo esta metodología y herramientas de soporte.

Sin embargo, debido a que el objetivo no es realizar un caso práctico para esta metodología sino el desarrollar un SMA que dé solución al problema planteado, el uso de INGENIAS se limita a la identificación de los requerimientos que pueden ser resueltos por el enfoque orientado a agentes, el seguimiento de las actividades comprendidas en el análisis y diseño de la metodología, y algunas guías para el desarrollo, implementación y pruebas del Sistema Multi-Agente. Así, en el siguiente apartado se expone los conceptos esenciales de esta metodología.

3.3.1.2 Conceptos fundamentales

Esta metodología introduce y define términos que permiten plasmar los conceptos de la teoría de agentes a un nivel práctico para el desarrollo de

sistemas. Dentro de estos conceptos están: agente, organización, objetivo, tarea, rol, recurso e interacción.

En INGENIAS, un ***agente*** es una entidad atómica autónoma, caracterizada por tener propósitos y una identidad única, así como la capacidad de perseguir objetivos mediante las asociaciones con roles y tareas.

Una ***organización*** es un grupo de agentes que persiguen objetivos comunes y es el motivo por el cual se han agrupado. Es una entidad autónoma que no tiene capacidad de ejecutar tareas ni para tomar decisiones, son los agentes que la componen quienes se encargan de ello.

Un ***objetivo*** es una entidad de control que indica al agente qué es lo que le motiva y guía su comportamiento. Puesto que se satisface con la presencia de evidencias, un objetivo puede estar: pendiente, recogiendo evidencias, refinado, en proceso, satisfecho o fallido.

Una ***tarea*** es un proceso integrado en un flujo de trabajo que transforma el estado global y que implica un conjunto de instrucciones que han de ejecutarse.

El término ***rol*** es una abstracción de un conjunto de responsabilidades (las funciones asociadas), protocolos (debido a que el rol tiene estado implícito) o permisos de actuación (cambiando los roles de un agente, se cambia su capacidad de actuación) que para existir necesita de una entidad no abstracta que lo desempeñe, en este caso el agente.

Un **recurso** es todo aquel objeto del entorno que no proporciona una funcionalidad concreta, pero que es indispensable para la ejecución de tareas y cuyo uso se restringe a consumir o restituir. Cuando el uso sea más complejo, como la funcionalidad requerida de una base de datos, se empleará el término *aplicación*.

El término **interacción** identifica las dependencias entre los componentes y contribuye a la especificación de su comportamiento así como la funcionalidad asociada.

INGENIAS propone un método de desarrollo que concibe al SMA como la representación computacional de un conjunto de modelos, donde cada uno de estos modelos muestra una visión parcial del SMA:

- los agentes que lo conforman
- las interacciones que existen entre ellos
- la forma en cómo se organizan para proporcionar la funcionalidad del sistema
- qué información es relevante en el dominio
- cómo es el entorno en el que se ubica el sistema a desarrollar.

Ahora bien, para realizar la especificación sobre cómo deben ser estos modelos se definen *meta-modelos*, los cuales son una representación de los tipos de entidades que pueden existir en un modelo, sus relaciones y restricciones de aplicación. Existen cinco meta-modelos que giran alrededor de dos entidades principales la *organización* y el *agente* (Figura 3.1).

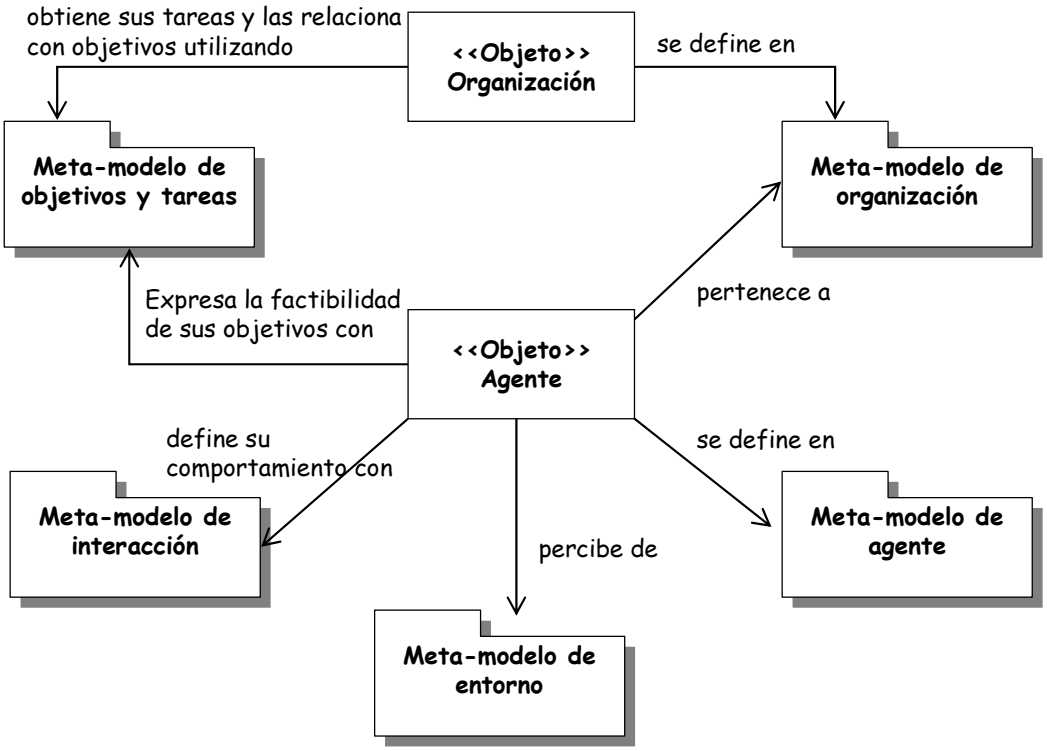


Figura 3.1: Relaciones entre los diferentes meta-modelos y las dos entidades principales.

A continuación se describe cada uno de los cinco meta-modelos.

- *Meta-modelo de agente.* Describe agentes particulares y los estados mentales en que se encontrarán a lo largo de su vida, por lo que se centra en la funcionalidad del agente y en el diseño de su control. Proporciona información sobre responsabilidades (asociación de tareas, objetivos y roles al agente) y el comportamiento (tipo de control, especificación de estado mental y su evolución). Por otra parte, este modelo se utiliza para describir estados intermedios de agentes, dichos estados se representan usando objetivos, hechos, tareas o cualquier otra entidad del sistema que ayude en la descripción del estado.

- *Meta-modelo de tareas y objetivos*. Su propósito es recoger las motivaciones del SMA, definir las acciones identificadas en los modelos de organización, interacciones o de agentes y cómo afectan estas acciones a sus responsables. Se hace una identificación de objetivos y tareas generales y una descomposición más concreta de éstos para que puedan ser asignados a agentes. Se usa para expresar la motivación que hay

detrás de las tareas y qué opciones de actuación se le presentan a un agente en un momento dado. También se utiliza para expresar cuáles son las entradas y las salidas de las tareas y cuáles son sus efectos sobre el entorno o el estado mental del agente.

- *Meta-modelo de organización*. Describe qué componentes del sistema (agentes, roles, recursos y aplicaciones) se agrupan juntos y la funcionalidad del sistema: qué tareas se ejecutan en común, qué objetivos comparten y qué restricciones hay que imponer sobre el comportamiento de los agentes. Las restricciones se expresan en forma de relaciones subordinadas y cliente-servidor.

- *Meta-modelo de interacción*. Se detalla cómo se coordinan y comunican los agentes, es decir, qué interacciones existen entre agentes. Este meta-modelo se construye sobre agentes, roles, objetivos, interacciones y unidades de interacción. Cada declaración de la interacción incluye a los agentes implicados, los objetivos perseguidos por la interacción y una descripción del protocolo que sigue la interacción.

- *Meta-modelo de entorno.* Define la percepción del agente en términos de elementos existentes en el sistema. También identifica recursos de sistema y quién es responsable de su administración. Su representación distingue tres posibles tipos: agentes, recursos y aplicaciones.

En suma, un sistema estará compuesto de varios modelos como los descritos previamente y no hay restricciones en el número de modelos de cada tipo, sino que se deja a consideración del desarrollador. En cuanto a su aplicación en las fases de análisis y diseño, se tiene que durante la fase de análisis se utilizan para buscar una primera versión del SMA puesto que actúan como guías para el analista, indicando qué entidades deben ser identificadas.

Durante la fase de diseño, se refinan los meta-modelos, al ir identificando nuevos componentes y relaciones entre ellos, para alcanzar el nivel apropiado de detalle. En paralelo, los componentes se traducen a entidades de cómputo, como máquinas de estado, administradores de sesión o motores de producción de reglas. Al final del proceso hay una especificación del SMA con un diseño basado en entidades concretas y realizables.

3.3.1.3 Herramientas

INGENIAS propone un conjunto de herramientas para la especificación, verificación e implementación de los Sistemas del Multi-Agente bajo el nombre de INGENIAS Development Kit (IDK) (Figura 3.2). Existen dos principales tipos de herramientas [INGENIAS2005]:

- *Editor de modelos.* Herramienta visual hecha en Java que permite crear y modificar especificaciones de un SMA usando conceptos de agentes. Para dichas especificaciones se pueden realizar como diagramas UML o usar la notación definida en la metodología INGENIAS. Esta herramienta para modelado visual almacena la especificación del sistema utilizando XML.

- *Módulos.* Permiten trabajar con las especificaciones del SMA para realizar la verificación de características y generación automática de código. Para la generación de código se utiliza unas plantillas configurables, especificadas con XML, para distintas plataformas de agentes (Jade, Robocode, Servlets). Aparte de los módulos que se proporcionan, los desarrolladores pueden ampliar la funcionalidad del IDK creando sus propios módulos para realizar la verificación de características específicas o la generación automática de código para una plataforma en particular. La implementación de los módulos es soportada por un marco de trabajo que facilita el pasar por las especificaciones y producir una cierta salida, que puede ser código o el resultado de una verificación de algunas características.

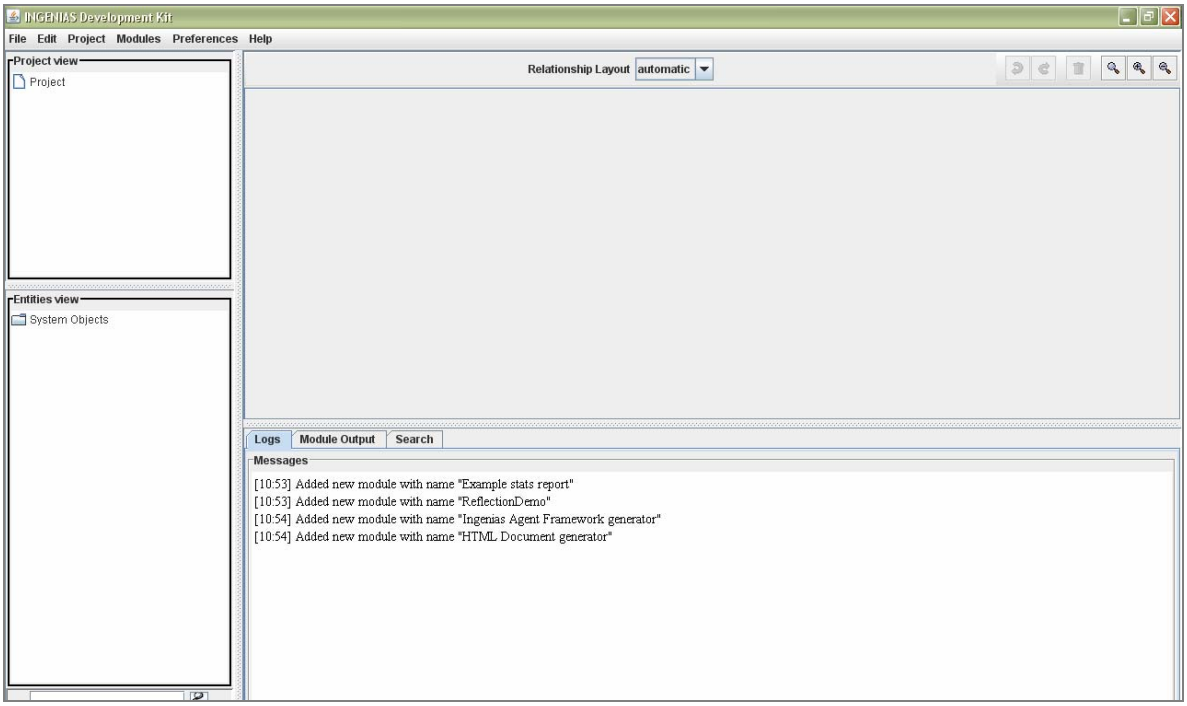


Figura 3.2: Pantalla principal del IDK.

En el editor principal del IDK (Figura 3.2) se puede crear un nuevo proyecto o cargar una especificación previamente almacenada. A través de esta interfaz se puede crear y navegar por la estructura de la especificación dada por los diferentes modelos y diagramas. También permite verificar si lo especificado es consistente y obtener el código para la plataforma JADE mediante el generador de código.

3.3.2 JADE

3.3.2.1 Descripción general

JADE (Java Agent DEvelopment Framework) es un paquete compuesto por un conjunto de herramientas y aplicaciones escritas totalmente en lenguaje Java. Es un framework que simplifica el desarrollo de Sistemas Multi-agentes al asegurar el cumplimiento de las especificaciones FIPA a través de un conjunto de herramientas gráficas que facilitan la depuración e implantación [BELLIFEMINE2001].

Adicionalmente con las librerías JADE se incluye el entorno de ejecución de agentes que puede instalarse y ejecutarse simultáneamente en varias estaciones de trabajo ya que sólo se requiere la maquina virtual de Java. La configuración de este entorno se puede realizar de varias formas: por medio de la línea de comandos, una interfaz gráfica local o remota o por el movimiento u operación de los gentes sobre una plataforma durante el ciclo de ejecución de las distintas aplicaciones.

En cuanto a su disponibilidad, JADE es distribuido gratuitamente como código abierto bajo los términos de LGPL por TILAB [JADE2005]. En suma, dentro de las características principales de la plataforma se encuentran [JADEADM 2005]:

- Provee una plataforma de agentes (entorno de ejecución) y un marco de trabajo de desarrollo (bibliotecas de clases).

- Incluye como parte fundamental especificaciones FIPA para la administración, comunicación, arquitectura y aplicación de los agentes.

- Cada agente se implementa como un hilo y como lenguaje de comunicación entre agentes utiliza FIPA-ACL, cuenta con constructores básicos de los agentes y su comportamiento se programa a través de creencias.

- Cuenta con *protocolos de comunicación* para comunicarse con otras plataformas: Java RMI, notificación de eventos, HTTP e IIOP; además de otros protocolos a través de interfaces MTP e IMTP JADE.

- Cuenta con *protocolos de interacción* que son utilizados por los agentes para comunicarse entre sí. Implementa la mayoría de los definidos por FIPA dentro de un paquete llamado jade.proto, estos son: FIPA-Request, FIPA-query, FIPA-Request-When, FIPA-recruiting, FIPA-brokering.

- Cuenta con *Comportamientos* que son realizados con una clase especial llamada Behaviour, mediante la cual es posible implementar las tareas que el agente ejecuta en respuesta a eventos externos o que el mismo genera y que debe efectuar para lograr un estado deseado.

- Cuenta con Ontologías para proporcionar entendimiento entre agentes de tal manera que tanto el emisor como el receptor de los mensajes estén de acuerdo en el significado que estos tienen. Se incluye la posibilidad de manipular la información de los mensajes ACL, generalmente en formato String (codec SL) o Bytes (codec LEAP), en forma de objetos y hacerlos más fáciles de manejar.

- Proporciona servicios obligatorios (gestión del ciclo de vida, páginas blancas, páginas amarillas, transporte de mensajes y análisis gramatical) y servicios opcionales (integración software-agente, ontología e interacción Humano-Agente).

- Cuenta con herramientas gráficas (DF, RMA, AMS) que soportan las fases de compilación, administración y monitoreo.

- Permite el desarrollo de aplicaciones tanto para dispositivos móviles como fijos (estaciones de trabajo, computadoras, PDA, teléfonos celulares).

En la figura 3.3 se puede apreciar la arquitectura general de la plataforma de JADE, en la cual un Sistema Multi-Agente puede estar en una JVM o en varias.

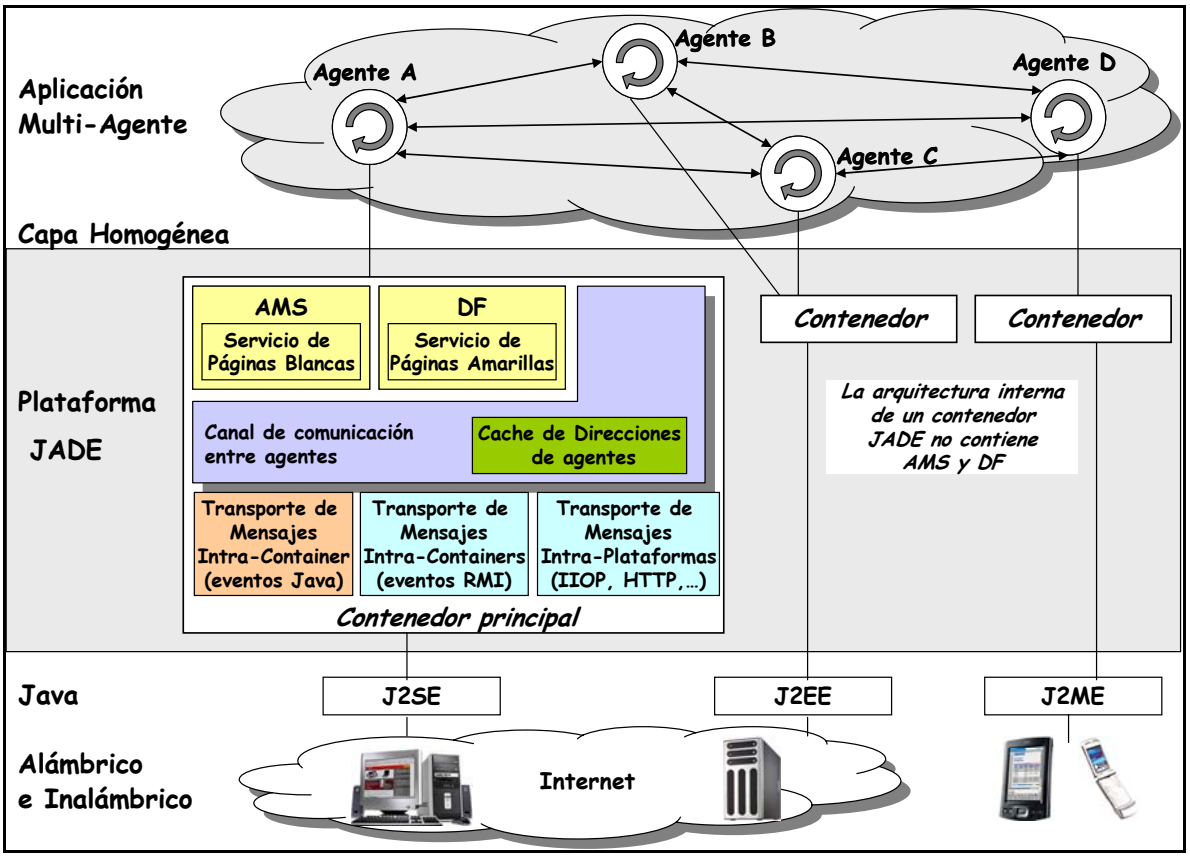


Figura 3.3. Arquitectura general de la Plataforma JADE

3.3.2.2 Herramientas

Como ya se ha mencionado, JADE proporciona herramientas gráficas para la gestión de agentes de gran utilidad: Sistema de Administración de Agentes (Agent Management System, AMS), Agente de Monitoreo Remoto (*Remote Monitoring Agent, RMA*), Agente de Servicio de Directorio (*DF*), agentes auxiliares para probar a los agentes desarrollados (*Dummy, Sniffer e Introspector*). Además, proporciona una librería con la mayoría de los protocolos de interacción definidos por FIPA, gestión de ontologías y soporte para lenguajes de contenido.

En la tabla 3.1 se explica más detalladamente cada uno de estos componentes debido a la importancia que tienen en el desarrollo del sistema:

Herramienta	**Descripción**
Sistema de Administración de Agentes (AMS)	Componente que garantiza que cada agente en la plataforma tenga un único nombre (AID). Además se encarga de proporcionar los servicios de páginas blancas y ciclo de vida, y de mantener el directorio de los identificadores de agentes y su estado.
Servicio de Directorio (DF)	Componente que brinda el servicio de páginas amarillas a los agentes y permite tanto al usuario como a los demás componentes de un Sistema Multi-Agente observar, eliminar, modificar y buscar la descripción de un determinado agente previamente registrado.
Agente de Monitoreo Remoto (RMA)	Permite controlar el ciclo de vida de la plataforma y todos los agentes registrados en ella. La arquitectura distribuida de JADE permite un monitoreo remoto a través de la GUI (interfaz de usuario).
Agente Dummy	Inspecciona el intercambio de mensajes entre agentes facilitando la validación de una interfaz de agentes antes de integrarla al Sistema Multi-Agente y la ejecución de consultas de prueba en caso de que uno de estos falle. Su interfaz grafica también provee soporte para edición, composición, recepción y envió de mensajes ACL desde y hacia cualquier agente además de permitir almacenar sus transacciones para su posterior análisis.
Agente Sniffer	Permite rastrear mensajes intercambiados en la plataforma JADE. Cuando el usuario decide conocer cada mensaje dirigido a o desde de un agente o grupo se rastrea y se visualiza en la interfaz.
Agente Introspector	Permite monitorear y controlar el ciclo de vida de los agentes que se ejecutan, además de interceptar los mensajes, tanto los enviados como los recibidos.

Tabla 3.1: Componentes de la interfaz gráfica de la plataforma JADE.

Una definición es una frase que significa la esencia de una cosa.

ARISTÓTELES

Capítulo 4

Análisis del Sistema Multi-Agente

En este capítulo se presenta la especificación del sistema mediante la definición de requerimientos y la descripción de los meta-modelos de la metodología INGENIAS correspondiente a la fase de análisis.

4.1 DEFINICIÓN DE REQUERIMIENTOS

En el capitulo anterior se ha especificado los elementos clave para la compresión del presente proyecto de tesis al abarcar los antecedentes, la descripción del problema, la solución que se propone y los objetivos que se buscan alcanzar con el desarrollo del Sistema Multi-Agente.

El propósito del proyecto es desarrollar e integrar un Sistema Multi-Agente a la herramienta HeAP MoProSoft para ofrecer consulta de información especializada que ayude o guíe al usuario en el aprendizaje e implantación de la Administración de Proyectos definida por MoProSoft

Para lograr este propósito se pretende que el usuario interactué con los agentes mediante la interfaz Web de HeAP MoProSoft permitiéndole obtener información sobre sus responsabilidades y posibilidades dentro de la empresa tales como: los procesos y roles relacionados, las tareas que debe realizar, los productos a generar, el estado de las tareas que realiza en la herramienta y sugerencias de la manera de cómo llevar a cabo estas tareas.

De acuerdo a lo anterior, se identificaron los siguientes requerimientos para el Sistema Multi-Agente:

1. Proveer información al usuario acerca del proceso del cual es responsable en función de su rol: descripción, propósito, categoría, objetivos, indicadores, autoridad, procesos relacionados, entradas, salidas, productos internos, roles involucrados, verificaciones, validaciones, actividades y tareas.

2. Indicar al usuario el estado de las tareas que debe realizar, es decir, aquellas que son soportadas por HeAP MoProSoft de acuerdo a su rol.

3. Brindar buenas prácticas de la Ingeniería de Software al ofrecer sugerencias sobre cómo se deben llevar a cabo las tareas que realiza el usuario.

4. Permitir al usuario el registro de nuevas sugerencias correspondientes a las tareas que realiza en HeAP MoProSoft.

5. Tener interacción con el usuario a través de la interfaz de usuario de HeAP MoProSoft.

6. Ser software de distribución libre y que cumpla con las características de calidad: correcto, eficiente, confiable, robusto, usable, verificable, mantenible, reparable, reusable, portable, entendible e interoperable [OKTABA2005].

4.2 ANÁLISIS

Como ya se ha mencionado, el desarrollo del sistema se realizará siguiendo la metodología INGENIAS. El análisis del Sistema Multi-Agente tiene como propósito realizar una especificación del sistema a desarrollar, la cual da como resultado un conjunto de modelos formados por diversos diagramas y esquemas que ofrecen una interpretación del problema a ser resuelto.

Ahora bien, para la fase de análisis en INGENIAS se indica los resultados a obtener de acuerdo al meta-modelo que se desee instanciar [GRASIA2005]. En la tabla 4.1 se muestran dichos resultados.

Meta-modelo	**Resultados**
De agente	• **Funcionalidad del agente**. Se refiere a un conjunto de roles que desempeña cada agente. Un conjunto de tareas asociadas a los agentes o a roles que estos jueguen. • **Requerimientos del agente**. Qué cualidades se esperan del agente. A este nivel basta con una descripción en lenguaje natural acompañada de casos de uso que lo ejemplifiquen.
De interacciones	• **Interacciones relevantes en el sistema.** Indicando participantes y objetivos perseguidos. • **Esquema inicial de intercambio de mensajes.** Realizado con diagramas de colaboración o de secuencia UML.
De tareas y objetivos	• **Tareas y objetivos**. Los objetivos que se persiguen en el sistema, que han de ser resumen de todos los incluidos en los modelos de agente y organización. Se admite la descomposición estructural de tareas y objetivos, con lo cual se espera disminuir su complejidad. Esta descomposición se acompaña de dependencias entre objetivos. • **Tareas asociadas a objetivos**. La asociación constituye la justificación de la ejecución de la tarea. • **Precondiciones y postcondiciones tentativas.** Las precondiciones mínimas son determinar qué entidades mentales se consumen, qué interacciones y entidades mentales se producen y qué aplicaciones se usarán en el proceso.
De organización	• **Estructura de la organización.** La identificación de los elementos del sistema es primordial. • **Flujos de trabajo.** Aunque no se llegue a un gran detalle, hay que generar una lista de las tareas, con sus responsables correspondientes, conectadas unas con otras y resaltando aquellas tareas cuya ejecución afecte a otros agentes.

Meta-modelo	Resultados
De entorno	• **Entidades del entorno.** Una enumeración de las entidades que preceden al sistema a desarrollar. Estas entidades se categorizan como aplicaciones, agentes o recursos. Cada entidad se acompaña de una descripción de sus funciones, que se entiende han de extraerse de la captura de requisitos. • **Percepción del agente.** La percepción del agente se asocia con las aplicaciones del entorno.

Tabla 4.1: Resultados a obtener en el análisis.

En base a lo anterior, a continuación se presentan los modelos obtenidos en la fase de análisis.

4.2.1 Descripción funcional del sistema

Para realizar la especificación funcional del SMA se hace uso de la técnica de casos de uso, así como una descripción de los objetivos y tareas relacionadas con dicha funcionalidad. A continuación se describen estos elementos.

4.2.1.1 Casos de uso

EL diagrama de casos de uso de la figura 4.1 muestra las principales funcionalidades identificadas.

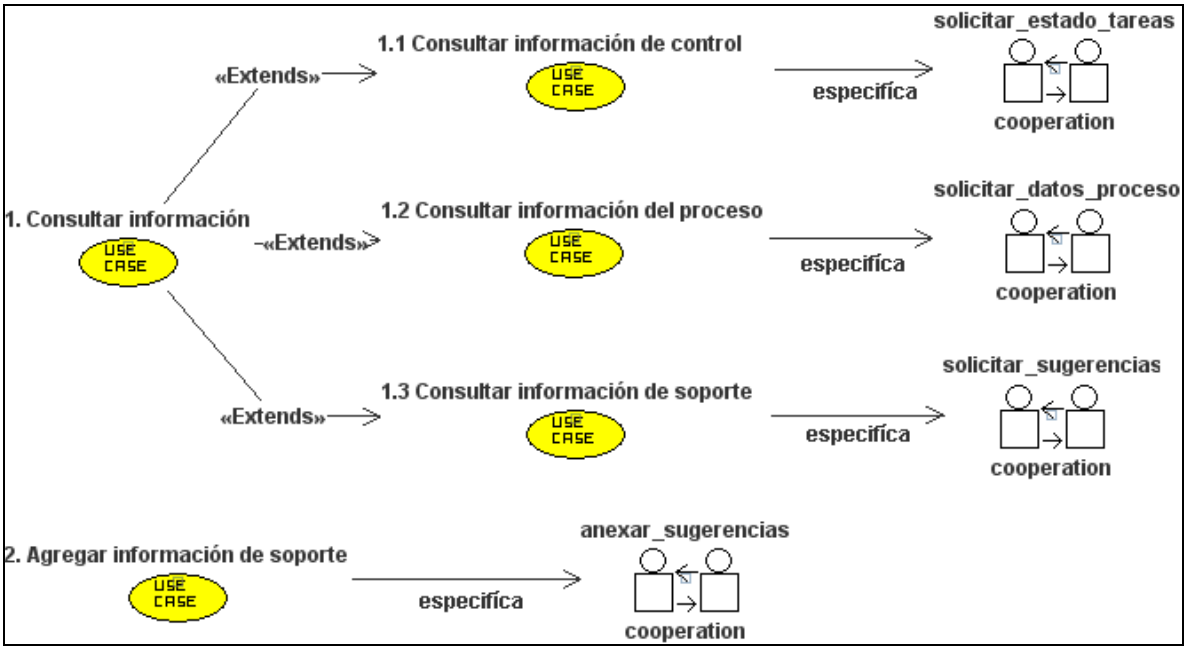


Figura 4.1: Diagrama de Casos de Uso.

Se presenta a continuación la descripción de cada caso de uso identificado.

1. Consultar información. El usuario debe poder consultar información sobre el proceso especificado en MoProSoft del que es responsable, el estado de las tareas que realiza en la herramienta y sugerencias de cómo llevar acabo estas tareas.

1.1 **Consultar información de control.** La consulta de información de control se refiere a la visualización de un listado de las tareas soportadas por la herramienta junto con su estado actual.

1.2 **Consultar información del proceso**. La consulta de información del proceso se refiere a la visualización de datos especificados en MoProSoft que están relacionados con el proceso del que es responsable el usuario de acuerdo a su rol: descripción, propósito, categoría, objetivos, indicadores, autoridad, procesos relacionados, entradas, salidas, productos internos, roles involucrados, verificaciones, validaciones, actividades y tareas.

1.3 **Consultar información de soporte**. La consulta de información de soporte se refiere a la visualización de un listado de las sugerencias de cómo puede realizar cierta tarea soportada por la herramienta HeAP MoProSoft.

2. Agregar información de soporte. El usuario debe poder anexar nuevas sugerencias de cómo llevar acabo las tareas que son soportadas en la herramienta HeAP MoProSoft.

Para detallar lo que se pretende con cada caso de uso, se utilizan los modelos de interacción que se presentan en la sección 4.2.3.

4.2.1.2 Descripción de objetivos

Para la descripción de objetivos se realiza una descomposición y una descripción de su satisfacción mediante la asociación de tareas.

4.2.1.2.1 Descomposición de objetivos

Para la descripción de los objetivos identificados a partir de los que persigue el sistema y del papel que se prevé van a jugar los agentes, a continuación se plantea la descomposición de objetivos (Figura 4.2).

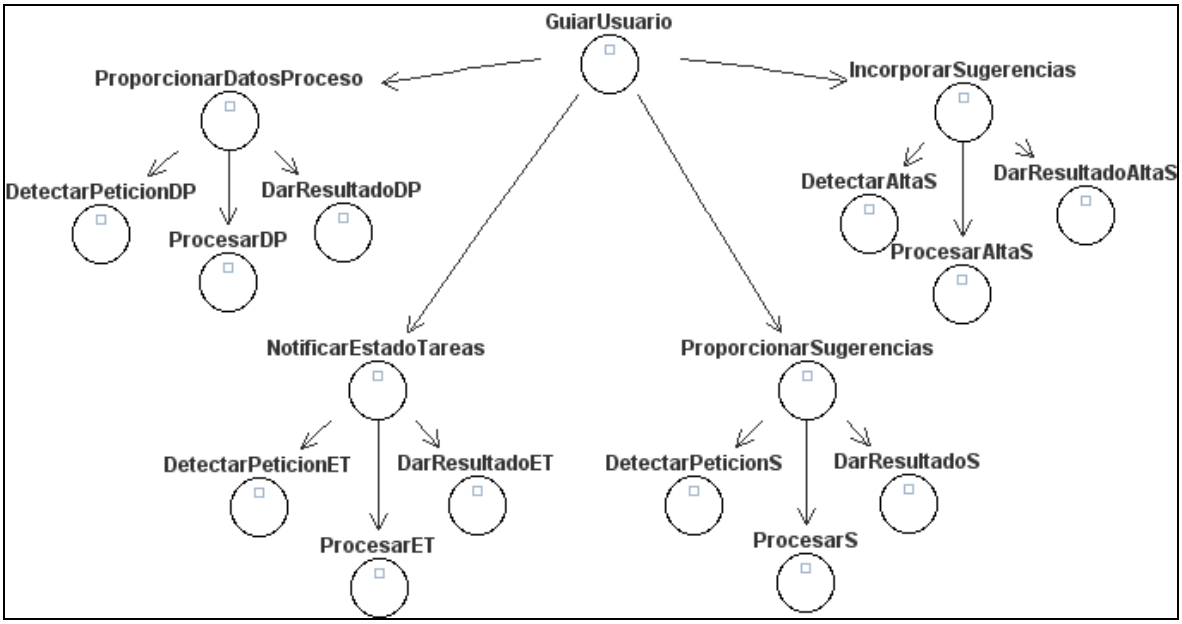


Figura 4.2: Meta-modelo de objetivos y tareas. Descomposición de objetivos.

A continuación se presenta en la tabla 4.2 a las entidades que conforman al meta-modelo de descomposición de objetivos.

Objetivo	Descripción
GuiarUsuario	El objetivo principal del sistema es guiar al usuario para maximizar su desempeño y aprendizaje en la implementación de la Administración de Proyectos definida por MoProSoft a través de la herramienta HeAP MoProSoft.
ProporcionarDatosProceso	Proporcionar información sobre el proceso del que es responsable el usuario de acuerdo a su rol.
NotificarEstadoTareas	Indicar al usuario el estado actual de las tareas que se deben realizar en HeAP MoProSoft.
ProporcionarSugerencias	Sugerir mediante recomendaciones o ejemplos la manera de realizar cierta tarea que es soportada por HeAP MoProSoft.
IncorporarSugerencias	Incorporar sugerencias por parte del usuario sobre las tareas que realiza en HeAP MoProSoft para aumentar el conocimiento disponible.
DetectarPeticionDP	Identificar que el usuario ha solicitado la acción de consultar los datos de un proceso y obtener la información necesaria para procesarla.
ProcesarDP	Procesar la petición de proporcionar los datos de un proceso.
DarResultadoDP	Ofrecer al usuario el resultado obtenido después de procesar la petición de proporcionar los datos de un proceso.
DetectarPeticionET	Identificar que el usuario ha solicitado la acción de consultar el estado de las tareas y obtener la información necesaria para procesarla.
ProcesarET	Procesar la petición de notificar el estado de las tareas.

Objetivo	Descripción
DarResultadoET	Ofrecer al usuario el resultado obtenido después de procesar la petición de notificar el estado de las tareas.
DetectarPeticionS	Identificar que el usuario ha solicitado la acción de consultar las sugerencias de una tarea y obtener la información necesaria para procesarla.
ProcesarS	Procesar la petición de proporcionar las sugerencias de una tarea.
DarResultadoS	Ofrecer el resultado obtenido después de procesar la petición de proporcionar las sugerencias de una tarea.
DetectarAltaS	Identificar que el usuario ha solicitado la acción de incorporar una nueva sugerencia y obtener la información necesaria para procesarla.
ProcesarAltaS	Procesar la petición de incorporar una nueva sugerencia.
DarResultadoAltaS	Ofrecer al usuario el resultado obtenido después de procesar la petición de incorporar una nueva sugerencia.

Tabla 4.2: Instancias de entidades del meta-modelo de descomposición de objetivos.

4.2.1.2.2 Satisfacción de objetivos

Para la satisfacción de los objetivos planteados se asocian a las tareas identificadas que ayudan a alcanzarlos como se muestra en la figura 4.3. La asociación es una instancia de GTAfecta con el propósito de poner de manifiesto que la ejecución de una tarea afecta al estado mental del agente al satisfacer o fallar un objetivo. Cabe mencionar que estas tareas más adelante se incorporan a los flujos de trabajo de la organización.

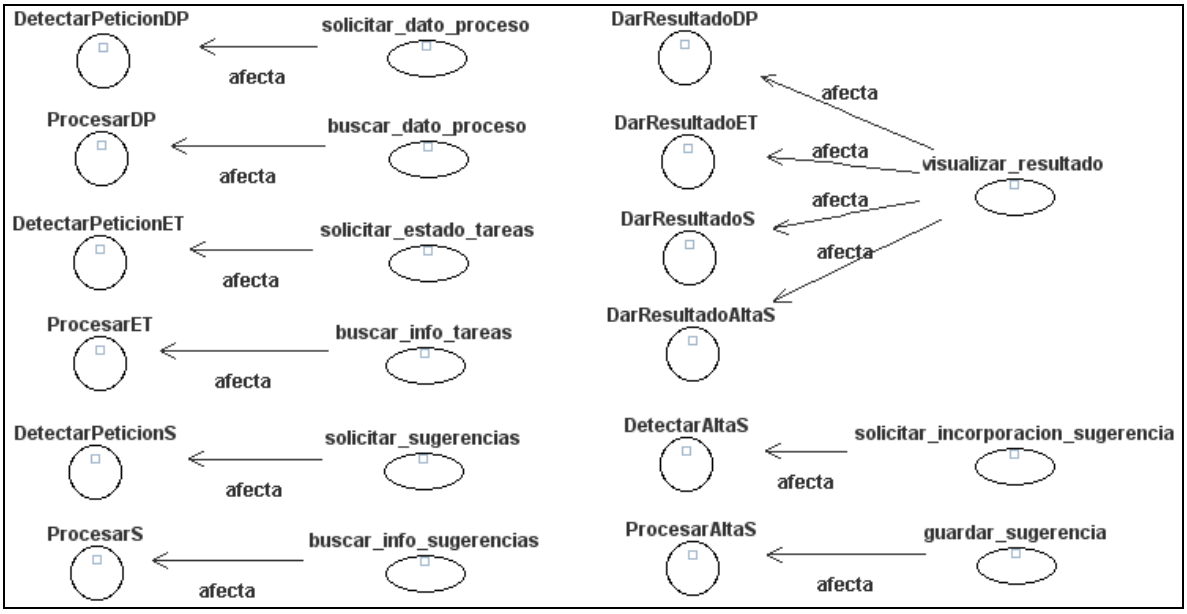


Figura 4.3: Meta-modelo de objetivos y tareas. Tareas asociadas a objetivos.

4.2.1.3 Descripción de tareas

Para la descripción de las tareas identificadas se muestran las entidades que son producidas y consumidas por dichas tareas. En el siguiente apartado se muestra la descripción de la tarea *solicitar_dato_proceso,* las demás descripciones se pueden consultar en el apéndice B de este documento.

Descripción de la tarea*: solicitar_dato_proceso*

Esta tarea toma como entrada los datos obtenidos del usuario a través de la herramienta HeAP MoProSoft (hecho *DatosUsuario*). Para atender la solicitud hace uso del elemento ServletAgente. El procesamiento de la solicitud de un dato del proceso se realiza dentro de la interacción *solicitar_datos_proceso*.

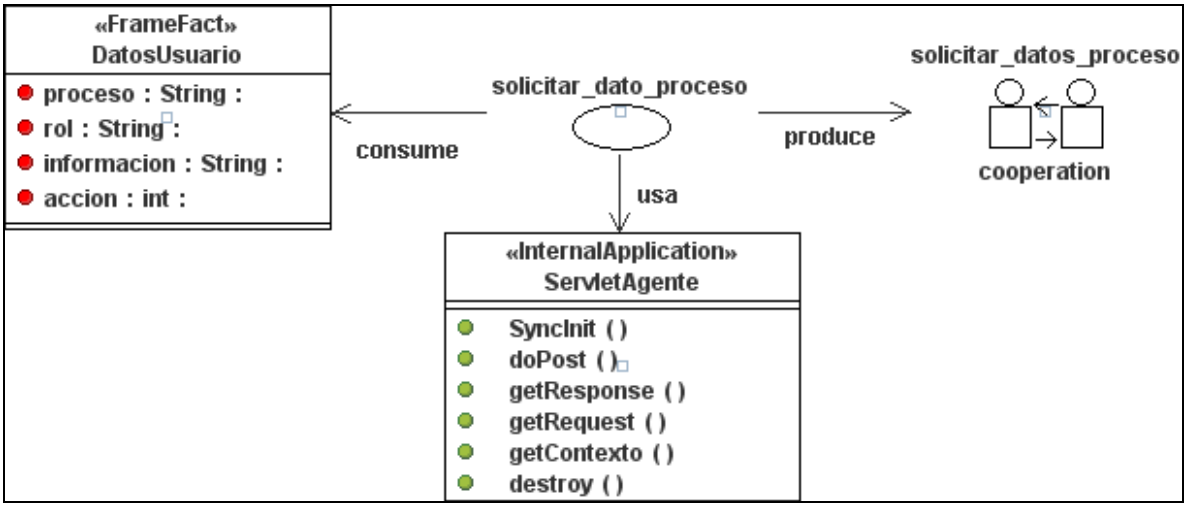


Figura 4.4: Descripción de la tarea *solicitar_dato_proceso.*

4.2.2 Descripción de agentes

Los agentes que forman al sistema son cuatro: *Coordinador, Supervisor, Asesor y Buscador*. A continuación se describe para cada uno de ellos un Modelo de Agente que muestra sus objetivos y los roles que desempeña.

4.2.2.1 Agente Coordinador

Este agente (Figura 4.5) se encarga de ofrecer la interfaz entre el usuario y el Sistema Multi-Agente mediante la observación de las acciones del usuario y cuando sea requerido, llamar a los agentes del grupo: *Supervisor*, *Asesor* y *Buscador*. Persigue cuatro objetivos: proporcionar los datos sobre el proceso que es responsable el usuario, notificar el estado actual de las tareas que realiza en la herramienta, proporcionar sugerencias de cómo realizar estas tareas e incorporar nuevas sugerencias. Estos objetivos los consigue desempeñando los roles:

Monitor_acciones y Visualizador_resultado. La forma exacta en que intervienen estos roles se define a continuación.

- ***Monitor_acciones:*** Inspecciona las acciones del usuario para llamar y coordinar a los agentes necesarios para satisfacer la solicitud realizada por el usuario.

- ***Visualizador_resultado***: Se encarga de obtener el resultado generado por los demás agentes y lo visualiza al usuario que realizó la petición.

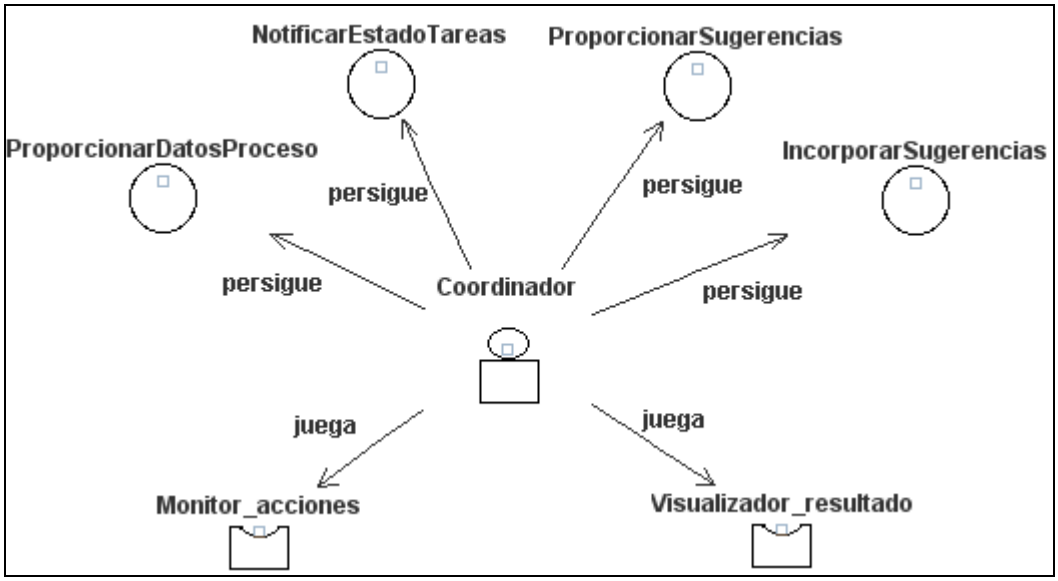


Figura 4.5: Meta-modelo del agente *Coordinador*.

4.2.2.2 Agente Supervisor

Este agente (Figura 4.6) supervisa las tareas que realiza el usuario en la herramienta HeAP MoProSoft para ofrecerle información sobre su estado. Para realizar la notificación del estado de las tareas hace una solicitud de información al agente *Buscador*. Persigue el objetivo de notificar el estado actual de las tareas que realiza el usuario en la herramienta y lo consigue desempeñando los roles: *Solicitante_informacion y Evaluador_tareas*. La forma exacta en que intervienen estos roles se define a continuación.

- ***Solicitante_informacion***: Obtiene e identifica la solicitud y los datos necesarios provenientes del agente *Coordinador.* Solicita una consulta de información al agente *Buscador.*

- ***Evaluador_tareas:*** Recibe la información solicitada al agente *Buscador* y la evalúa en base a la identificación del estado de cada tarea. Manda el resultado al agente *Coordinador*.

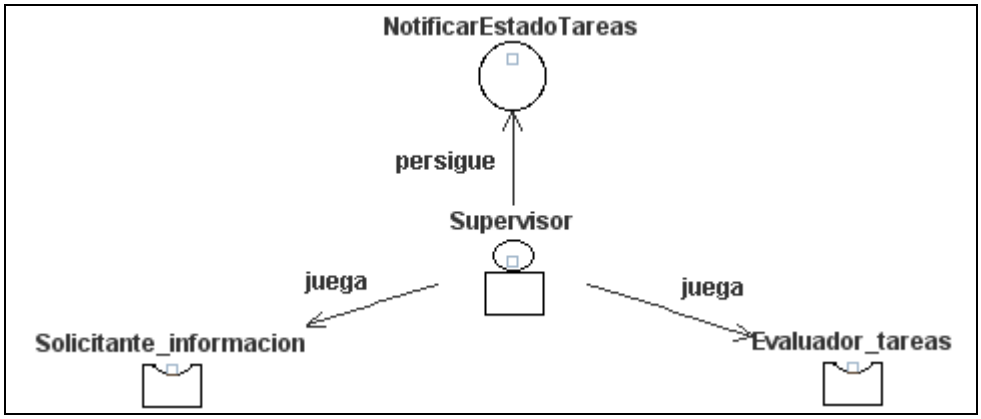


Figura 4.6: Meta-modelo del agente *Supervisor*.

4.2.2.3 Agente Asesor

Este agente (Figura 4.7) ofrece sugerencias de cómo llevar a cabo las tareas que realiza el usuario en la herramienta HeAP MoProSoft. Persigue el objetivo de proporcionar sugerencias y lo consigue desempeñando los roles: *Solicitante_informacion* y *Evaluador_sugerencias*. La forma exacta en que intervienen estos roles se define a continuación.

- ***Solicitante_informacion***: Obtiene e identifica la solicitud y los datos necesarios provenientes del agente *Coordinador.* Solicita una consulta de información al agente *Buscador.*
- ***Evaluador_sugerencias:*** Recibe la información solicitada al agente *Buscador* y la evalúa con base a la identificación de las sugerencias correspondientes a la tarea solicitada. Manda el resultado al agente *Coordinador*.

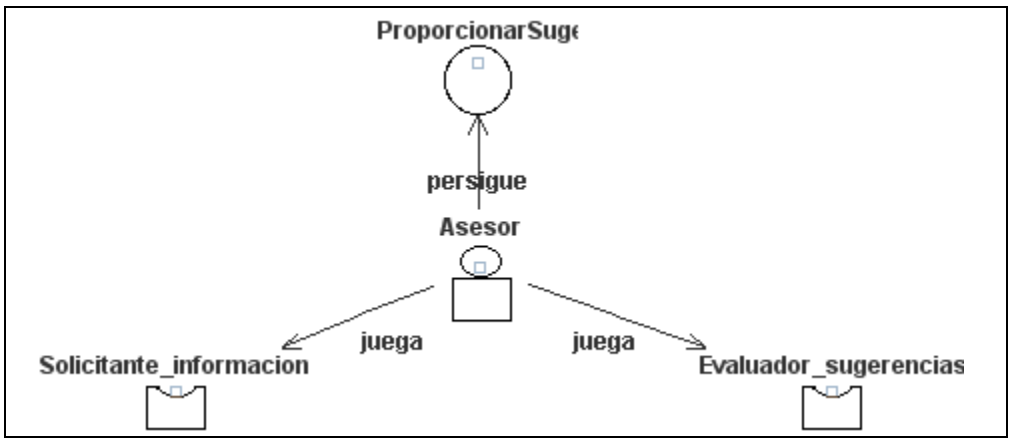


Figura 4.7: Meta-modelo del agente *Asesor*.

4.2.2.4 Agente Buscador

Este agente (Figura 4.8) maneja la información almacenada en la Base de Datos de Soporte y en la Base de Datos de HeAP MoProSoft para proporcionar a los demás agentes del grupo la consulta de ciertos datos o la alta de sugerencias. Persigue cuatro objetivos y los consigue desempeñando los roles: *Receptor_solicitud, Buscador_informacion* y *Almacenador_sugerencia*. La forma exacta en que intervienen estos roles se define a continuación.

- ***Receptor_solicitud:*** Recibe la solicitud de información y los datos necesarios por parte de un agente para buscar o almacenar en el repositorio adecuado lo que le fue requerido.

- ***Buscador_informacion***: Busca en el repositorio adecuado los datos que sean necesarios y envía el resultado de la búsqueda al agente apropiado.

- ***Almacenador_sugerencia***: Almacena en el repositorio adecuado el registro recibido y envía el resultado de la transacción al agente *Coordinador*.

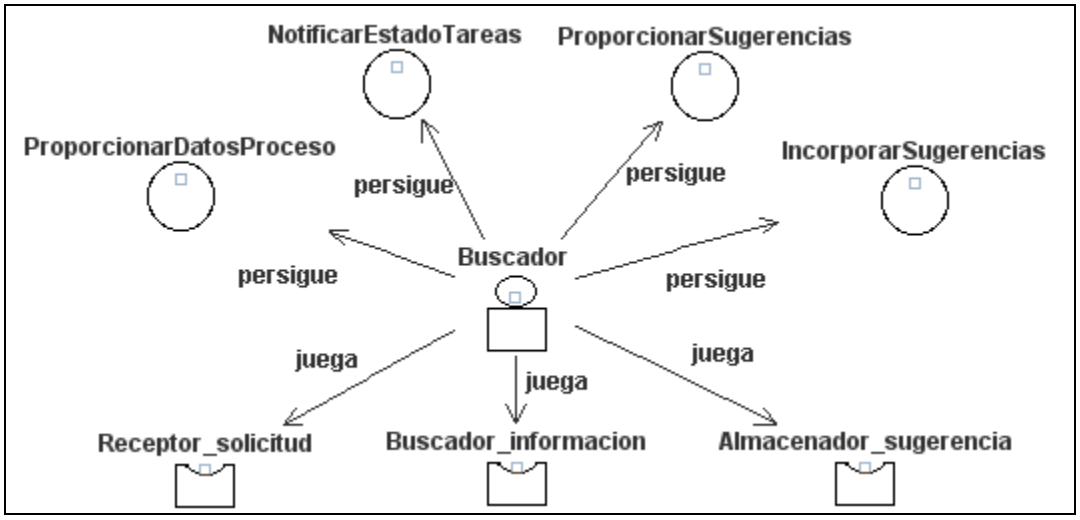


Figura 4.8: Meta-modelo del agente *Buscador*.

4.2.3 Descripción de la interacción entre los agentes del sistema

Para el sistema se identificaron cuatro interacciones principales: *solicitar_datos_proceso, solicitar_estado_tareas, solicitar_sugerencias y anexar_sugerencias*. A continuación se muestra sólo la interacción *solicitar_datos_proceso,* las demás interacciones se pueden consultar en el apéndice B.

Interacción: solicitar_datos_proceso

Solicitar datos del proceso (Figura 4.9) consiste en ofrecer al usuario información sobre el proceso que es responsable de acuerdo a su rol. Dicha información puede ser referente a la descripción, propósito, categoría, objetivos, indicadores, autoridad, procesos relacionados, entradas, salidas, productos internos, roles involucrados, verificaciones, validaciones, actividades y tareas.

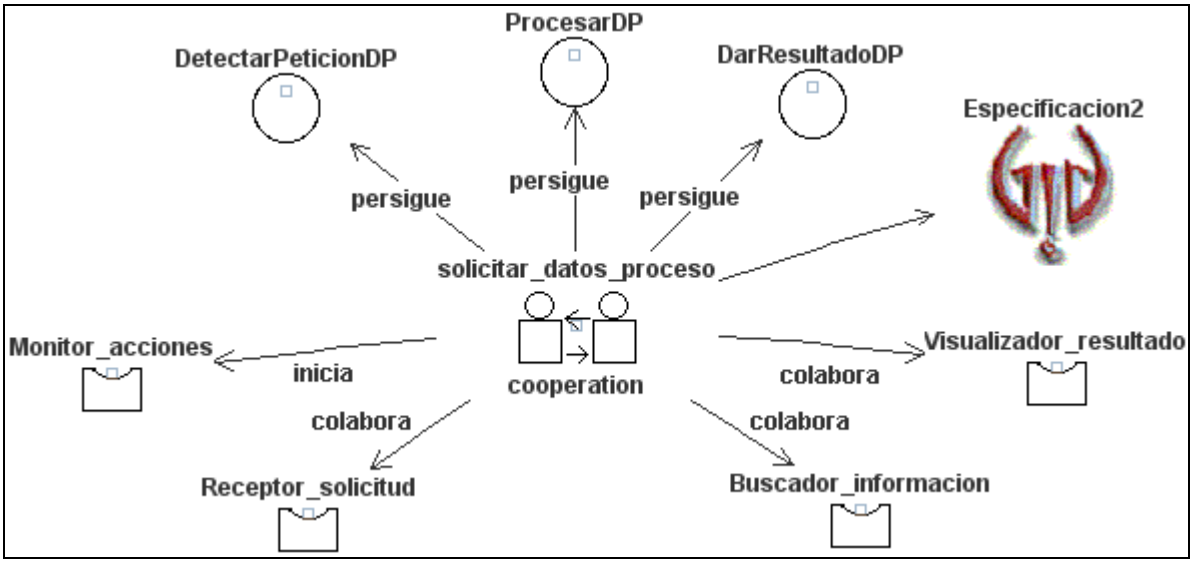


Figura 4.9: Meta-modelo de la interacción *solicitar_datos_proceso*.

A continuación se muestra en la figura 4.10 el diagrama de colaboración para esta interacción.

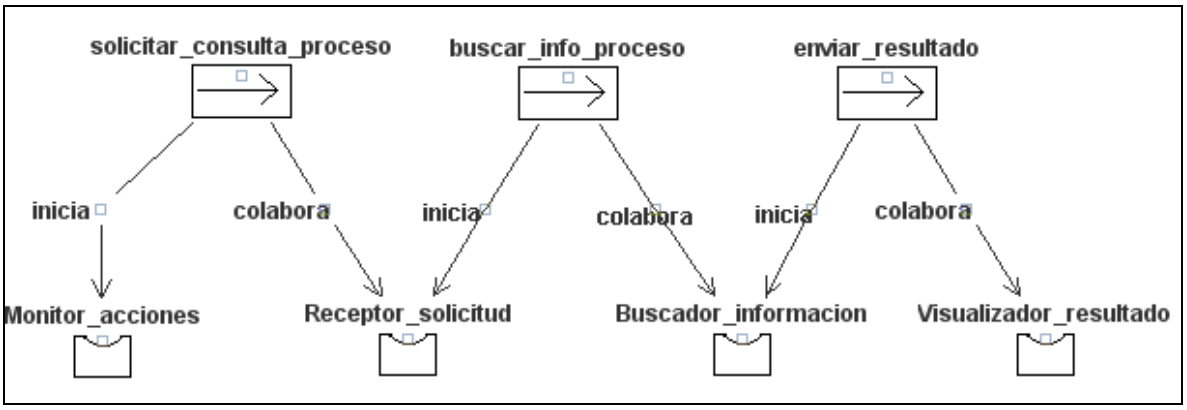


Figura 4.10: Diagrama de colaboración para la interacción *solicitar_datos_proceso*.

4.2.4 Descripción del entorno del sistema

Como parte del entorno del sistema se tienen a los usuarios, los cuales son las personas que laboran en la empresa de desarrollo de software que utiliza la aplicación HeAP MoProSoft. Cabe señalar, que la herramienta tiene como tipos de usuarios a los roles de MoProSoft: Responsable de Gestión de Proyectos (RGPY) y Responsable de la Administración del Proyecto Específico (RAPE).

Para un mayor detalle de la descripción del entorno, a continuación se realiza una explicación de los recursos disponibles para el Sistema Multi-Agente.

4.2.4.1 Recursos disponibles

Los recursos o aplicaciones disponibles (Figura 4.11) en el entorno del SMA son:

Aplicaciones de Entorno

- ***Base de datos de HeAP MoProSoft (BDHeAPMoProSoft)*.** Base de datos utilizada para la gestión de los datos de la aplicación HeAP MoProSoft. Mediante la consulta de este repositorio, el SMA da a conocer información de control al usuario, en este caso, el estado de las tareas que realiza en la herramienta.

Aplicaciones Internas

- **Base de datos de Soporte.** Repositorio donde se almacena información de los procesos especificados en MoProSoft y las sugerencias de cómo llevar a cabo las tareas definidas en estos procesos. Mediante la manipulación de esta base de datos el SMA da a conocer información del proceso y de soporte al usuario, así como la opción de incorporar nuevas sugerencias.

- **ServletAgente**. Paquete conformado por tres clases que hacen posible la integración del agente Coordinador en la interfaz gráfica de HeAP MoProSoft. Este servlet se ejecuta cuando el usuario realiza alguna acción en la interfaz de HeAP MoProSoft que tiene que ver con los servicios del SMA. También contesta la petición *httpRequest* al proveer la información adecuada generada por el Sistema Multi-Agente.

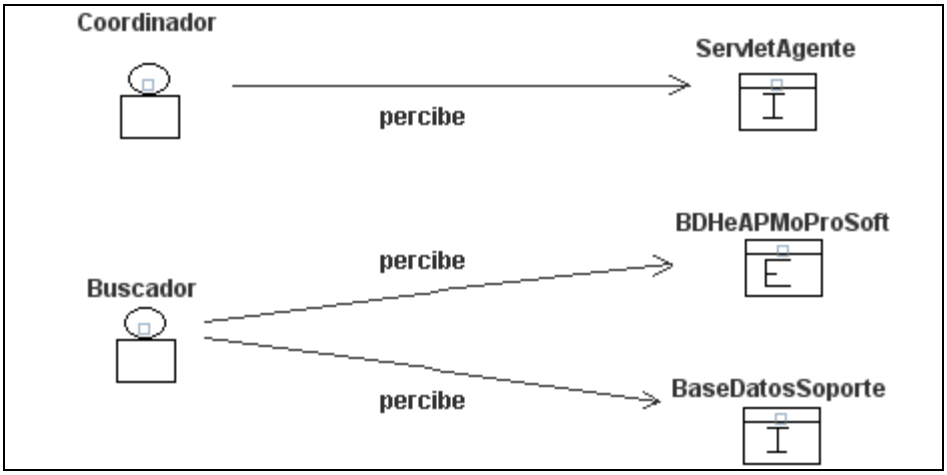


Figura 4.11: Meta-modelo de entorno: recursos y aplicaciones identificadas.

4.2.5 Descripción de la organización del sistema

En el siguiente apartado, se muestra la estructura general y los elementos que conforman al Sistema Multi-Agente. Se da una visión general de los objetivos que se persiguen, los agentes participantes, los recursos disponibles, las relaciones que se dan entre estos y como están organizados.

4.2.5.1 Estructura del sistema

Con base en la información mostrada en las diferentes instancias de los meta-modelos anteriores, se tiene la siguiente estructura del Sistema Multi-Agente (Figura 4.12). La forma exacta en que intervienen las entidades que la conforman se define a continuación.

- ***SMA:*** Esta organización persigue el objetivo de guiar al usuario en el aprendizaje e implantación de la Administración de Proyectos definida por MoProSoft. Está conformada por dos grupos: agentes y recursos. Las actividades de la organización son especificadas por cinco flujos de trabajo que se detallarán adelante.

- ***Agentes:*** Este grupo es conformado por un agente Coordinador, Supervisor, Asesor y Buscador.

- ***Recursos:*** En este grupo se encuentran los recursos utilizados por los agentes del sistema para gestionar las peticiones del usuario. Se contemplan como recursos a cuatro aplicaciones: la Base de Datos de HeAP MoProSoft, la Base de Datos de Soporte y el elemento SelvletAgente.

- ***Actividades_SMA:*** Es el flujo de trabajo global que abarca todas las actividades del SMA. Está compuesto por los flujos de trabajo: Obtener_estado_tareas, Obtener_datos_proceso, Obtener_sugerencias y Agregar_sugerencias.

- ***Obtener_estado_tareas:*** Este flujo de trabajo incluye las tareas para proporcionar al usuario la consulta del estado de las tareas que realiza en HeAP MoProSoft.

- ***Obtener_datos_proceso:*** Este flujo de trabajo incluye las tareas para proporcionar al usuario la consulta de información sobre el proceso que es responsable de acuerdo a su rol.

- ***Obtener_sugerencias:*** Este flujo de trabajo incluye las tareas para proporcionar al usuario la consulta de sugerencias de cómo llevar a cabo cierta tarea.

- ***Agregar_sugerencias:*** Este flujo de trabajo incluye las tareas para proporcionar al usuario la posibilidad de anexar nuevas sugerencias sobre cierta tarea que realiza en HeAP MoProSoft.

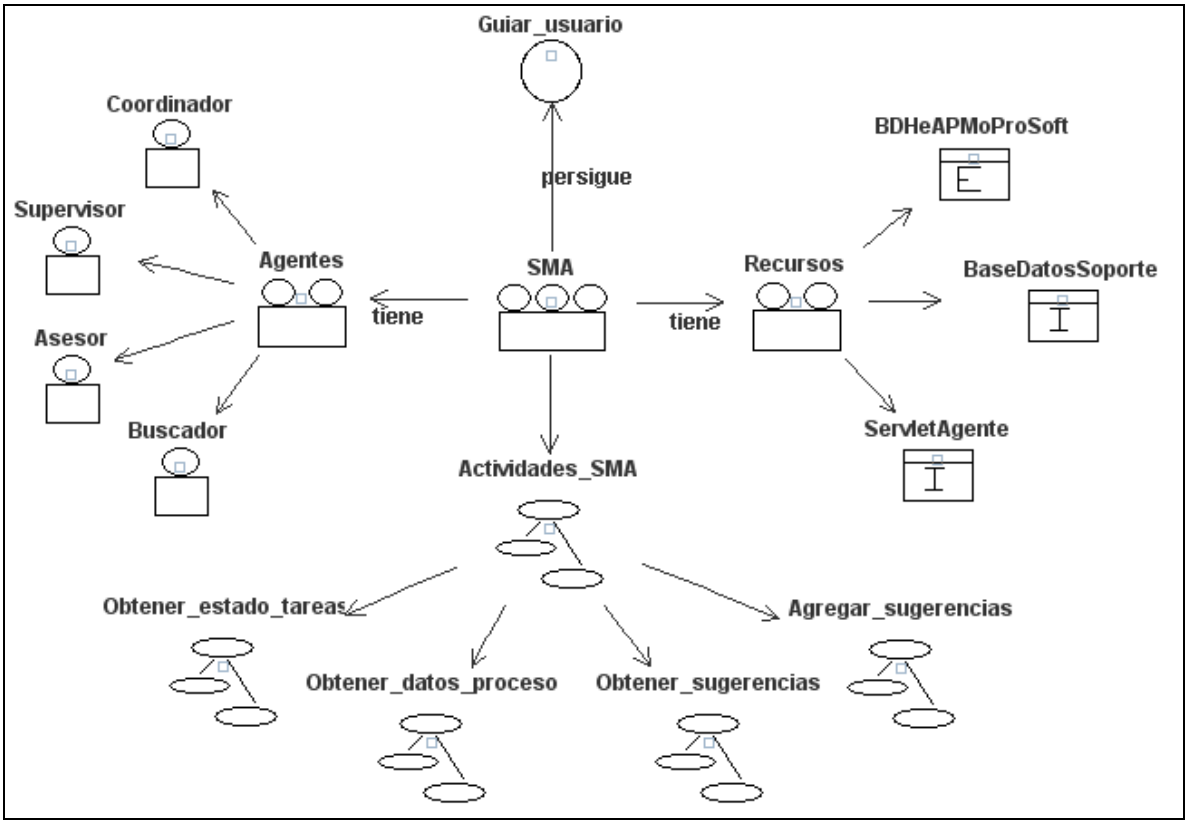


Figura 4.12: Estructura del sistema.

Pregúntate si lo que estás haciendo hoy te acerca al lugar en el que quieres estar mañana.

J. BROWN

Capítulo 5

Diseño del Sistema Multi-Agente

En este capítulo se presenta los modelos correspondientes a la fase de diseño aplicando la metodología INGENIAS, a partir de los cuales se realizará la implementación. Además, se presenta la definición del diseño de las dos aplicaciones internas a desarrollar: la Base de Datos de Soporte y el elemento ServletAgente.

Una vez identificados los requerimientos del sistema, los objetivos que éste debe cumplir, la estructuración del mismo y las tareas necesarias para cumplir con los objetivos; se pasa a la fase de diseño que consiste en el refinamiento de los diagramas de la fase de análisis.

Los resultados esperados en el diseño en INGENIAS se centran en refinar los flujos de trabajo y las tareas asociadas, las interacciones, cómo es el control del agente y cómo se satisfacen los objetivos del sistema. En la tabla 5.1 se muestran los resultados a obtener de acuerdo al meta-modelo que se desee instanciar [GRASIA2005].

Meta-modelo	**Resultados**
De agente	• **Restricciones de control**. Se representan mediante los estados mentales por los que pasa el agente a lo largo de las interacciones. • **Medios de control.** Cómo es el control que asegura la transición entre los distintos estado mentales identificados. La especificación de estos medios puede hacerse en lenguaje natural o utilizando referencias a modelos de objetivos y tareas.
De interacciones	• **Descripciones detalladas de las interacciones.** Han de ser tan detalladas como para permitir una generación automática de código. El nivel de detalle es función de las entidades que se escogen para implementar las interacciones. • **Contexto de las interacciones.** Las interacciones contextualizadas en los flujos de trabajo de la organización y con información adicional acerca de las condiciones de participación en la interacción. • **Conjunto de unidades de interacción asociadas a los participantes.** Cada participante en las interacciones comparte con las otras unidades de interacción. Esto forma parte de la descripción funcional del agente.
De tareas y objetivos	• **Refinamiento de las dependencias entre objetivos.** Se detalla la dependencia indicando si se trata de dependencia Y (GTDependeY) o dependencia O (GTDependeO). • **Condiciones de satisfacción o fallo de los objetivos.** Las relaciones GTAfecta entre tareas y objetivos evolucionan a GTFalla y GTSatisface. Es necesario además indicar bajo qué condiciones se asume el éxito o fracaso del objetivo. • **Precondiciones detalladas.** El detalle en las precondiciones viene de añadir los recursos que necesita la tarea (WFUsa) y de incluir nuevas instancias de WFConsume, GTDestruye, GTModifica.

Meta-modelo	Resultados
	• **Postcondiciones detalladas.** Se asocian nuevas instancias WFProduce con recursos para expresar la restitución de los mismos. Adicionalmente, se incluyen instancias WFUsaLlamada que expresan qué operaciones se están utilizando de las aplicaciones asociadas. Asimismo, se incluyen instancias de GTDestruye, GTModifica y GTCrea para mostrar cómo se modifica el estado mental.
De organización	• **Contexto de ejecución de las tareas.** Las tareas se asocian con sus ejecutores para indicar quién se ve afectado por ellas, qué recursos y entidades mentales se consumen y qué entidades de información se producen. • **Refinamiento de las dependencias sociales.** Hay que detallar qué tipo de relaciones se están definiendo cuando se interconectan dos entidades. Concretamente, se debe determinar si hay o no subordinación o si se trata de una relación cliente-servidor.
De entorno	• **Configuración de la percepción del agente.** Cada agente asociado con aplicaciones explicita cómo va a percibir, si va a utilizar mecanismos de muestreo o notificación y qué es lo que espera recibir. • **Relaciones con tareas.** Aunque se considera en otros meta-modelos, se identifican asociaciones de producción y utilización de aplicaciones y recursos con tareas. • **Definición detallada de los recursos.** Consiste en especificar exactamente cómo es cada recurso, detallando su categoría concreta, el límite inferior de consumo, el superior y el estado en que se encuentra inicialmente.

Tabla 5.1: Resultados a obtener en el diseño.

Tomando en cuenta lo anterior, en los siguientes apartados se ofrece los diagramas de la fase de análisis con un nivel mayor de detalle, las nuevas instancias de los meta-modelos que son necesarias en la fase de diseño y la definición de diseño de las dos aplicaciones internas necesarias para el Sistema Multi-Agente: Base de Datos de soporte y ServletAgente

5.1 DESCRIPCIÓN FUNCIONAL DEL SISTEMA

5.1.1 Descripción de objetivos

Para la descripción de objetivos se detalla las dependencias del modelo de descomposición del análisis, así como la descripción de la satisfacción de los

objetivos mediante las evidencias o propiedades que deben cumplirse para considerar un objetivo como fracasado o satisfecho.

5.1.1.1 Descomposición de objetivos

En la figura 5.1 se detallan las dependencias del modelo de descomposición indicando si se trata de dependencia Y (el objetivo padre puede ser alcanzado si todos los sub-objetivos son satisfechos) o dependencia O (requiere que al menos uno de los sub-objetivos haya sido satisfecho).

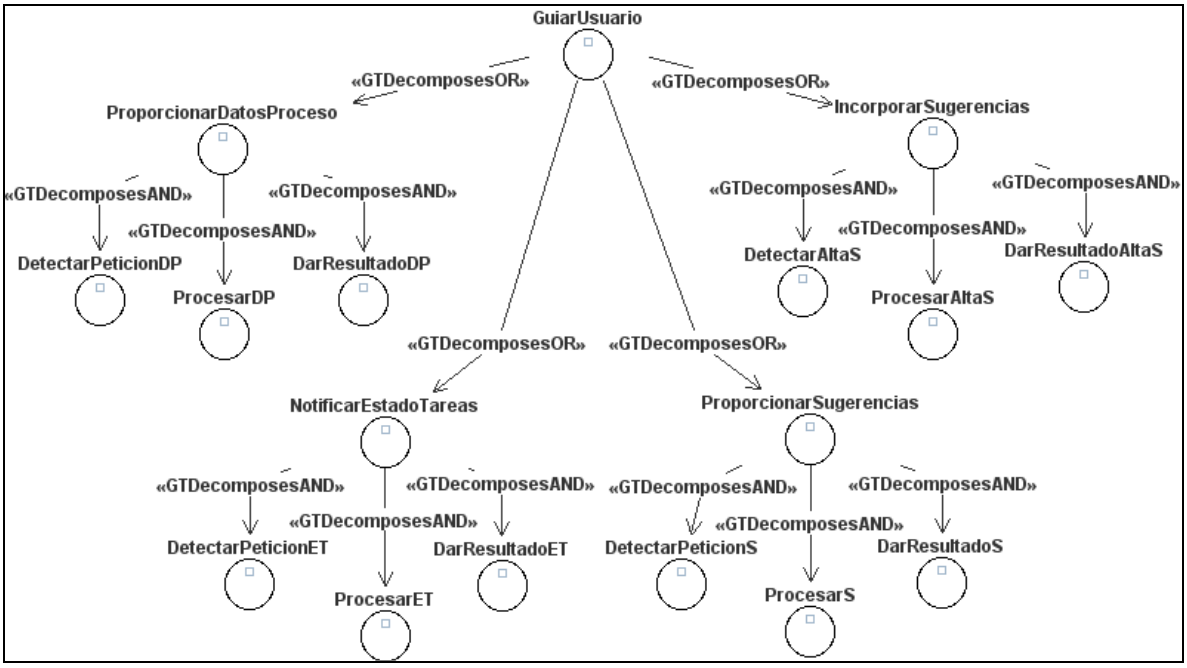
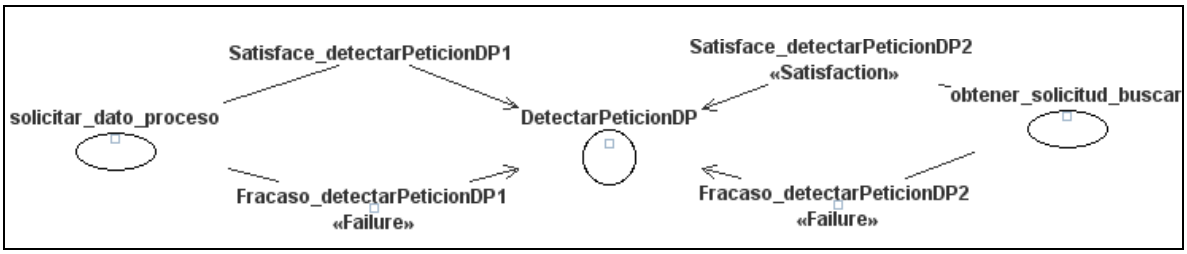
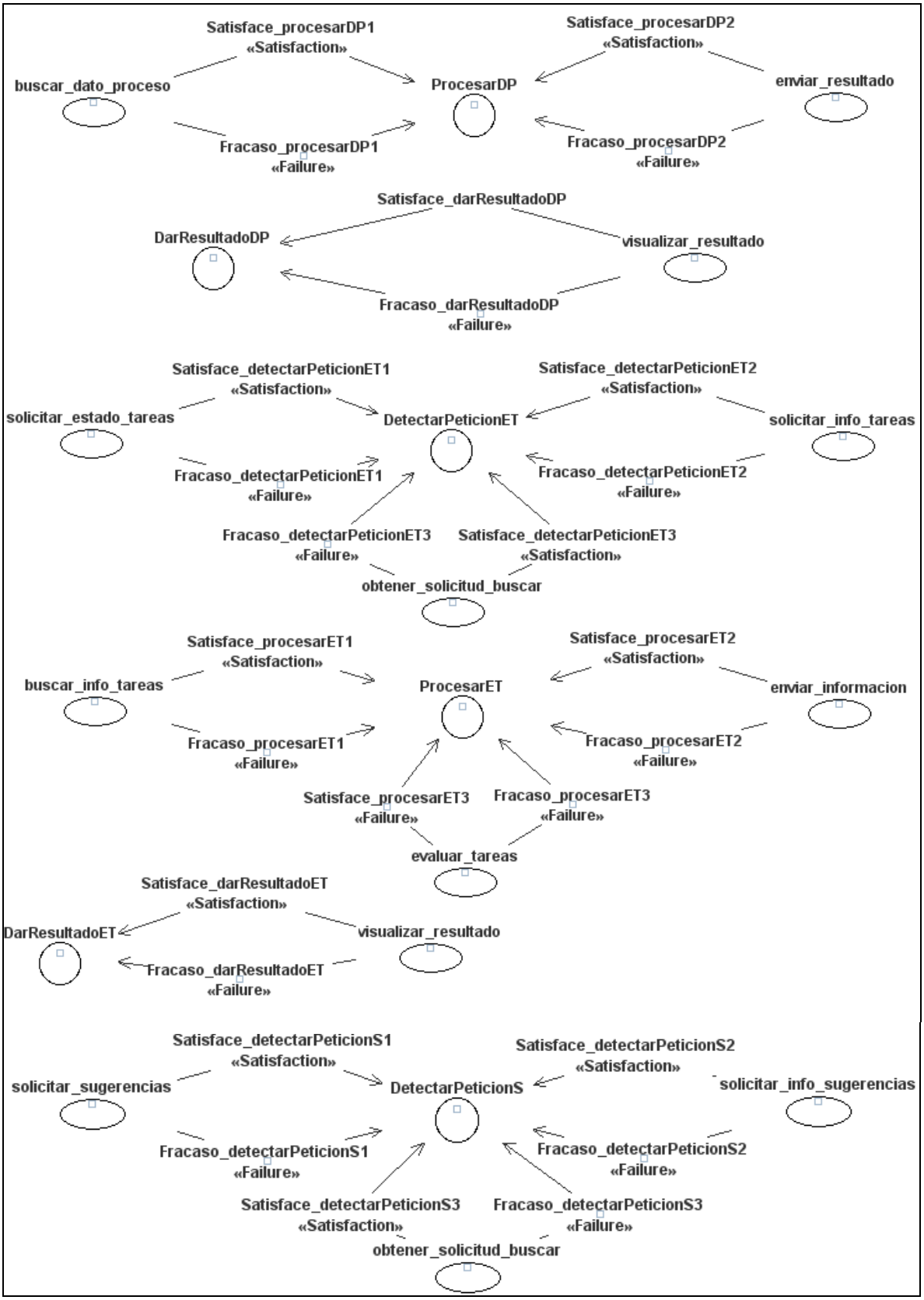


Figura 5.1: Descomposición de objetivos con relaciones.

5.1.1.2 Satisfacción de objetivos

Para la satisfacción de los objetivos planteados se describe como se alcanzan al mostrar que condiciones mentales deben cumplirse para considerar el objetivo fracasado o satisfecho como se muestra en la figura 5.2.




Satisface_procesarDP1
«Satisfaction»
buscar_dato_proceso
ProcesarDP
Satisface_procesarDP2
«Satisfaction»
enviar_resultado
Fracaso_procesarDP1
«Failure»
Fracaso_procesarDP2
«Failure»
Satisface_darResultadoDP
DarResultadoDP
visualizar_resultado
Fracaso_darResultadoDP
«Failure»
Satisface_detectarPeticionET1
«Satisfaction»
Satisface_detectarPeticionET2
«Satisfaction»
solicitar_estado_tareas
DetectarPeticionET
solicitar_info_tareas
Fracaso_detectarPeticionET1
«Failure»
Fracaso_detectarPeticionET2
«Failure»
Fracaso_detectarPeticionET3
«Failure»
Satisface_detectarPeticionET3
«Satisfaction»
obtener_solicitud_buscar
Satisface_procesarET1
«Satisfaction»
Satisface_procesarET2
«Satisfaction»
buscar_info_tareas
ProcesarET
enviar_informacion
Fracaso_procesarET1
«Failure»
Fracaso_procesarET2
«Failure»
Satisface_procesarET3
«Failure»
Fracaso_procesarET3
«Failure»
evaluar_tareas
Satisface_darResultadoET
«Satisfaction»
DarResultadoET
visualizar_resultado
Fracaso_darResultadoET
«Failure»
Satisface_detectarPeticionS1
«Satisfaction»
Satisface_detectarPeticionS2
«Satisfaction»
solicitar_sugerencias
DetectarPeticionS
solicitar_info_sugerencias
Fracaso_detectarPeticionS1
«Failure»
Fracaso_detectarPeticionS2
«Failure»
Satisface_detectarPeticionS3
«Satisfaction»
Fracaso_detectarPeticionS3
«Failure»
obtener_solicitud_buscar

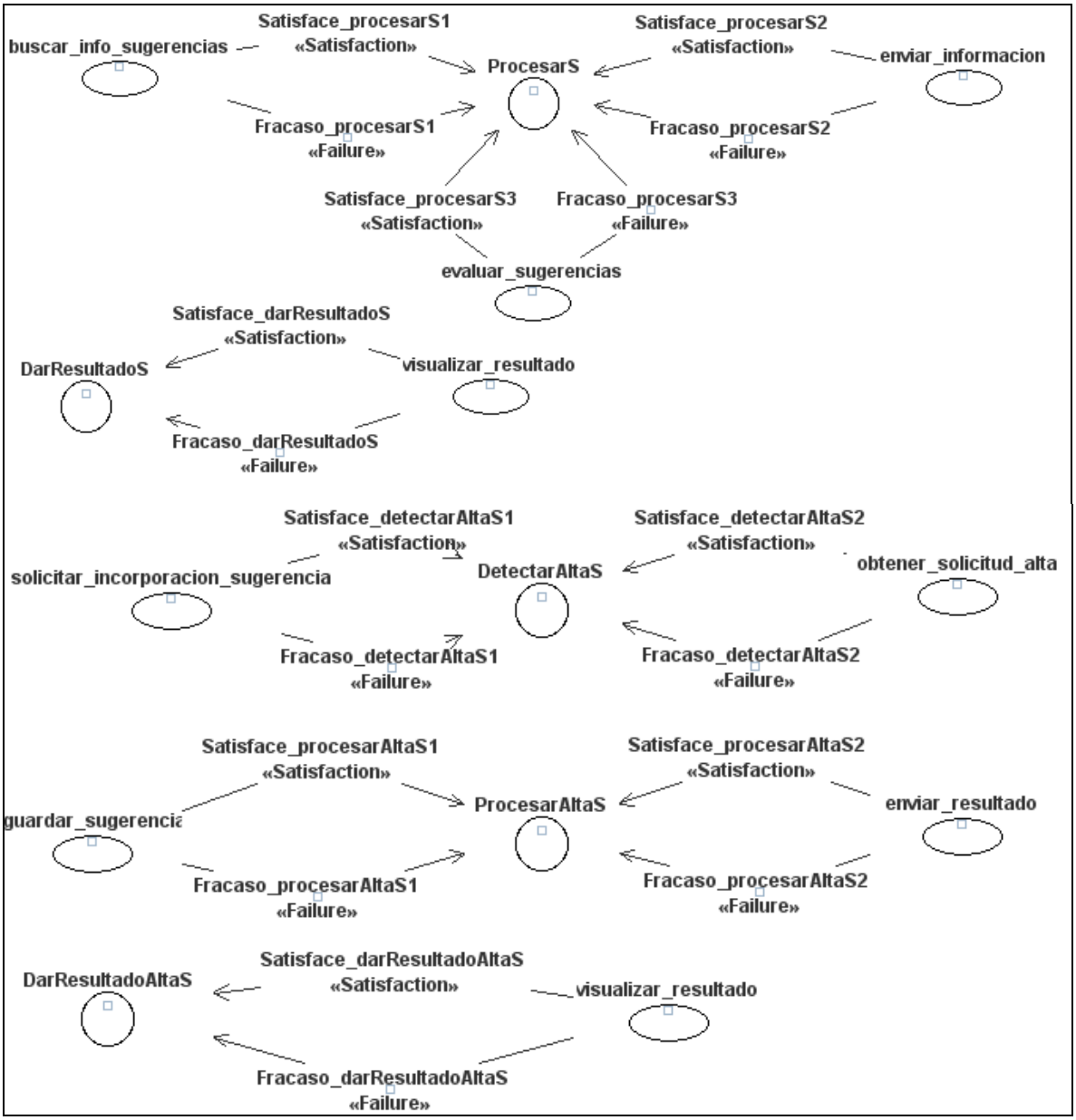


Figura 5.2: Meta-modelo de objetivos y tareas. Satisfacción de objetivos.

La forma exacta en que intervienen estas condiciones mentales se definen en el apéndice B.

5.2 DESCRIPCIÓN DE LA INTERACCIÓN ENTRE LOS AGENTES

Para terminar la descripción de las interacciones entre agentes del sistema, se presentan los diagramas de colaboración realizados en la fase de análisis con el tipo de cada unidad de interacción identificada (paso de mensaje), el orden existente entre estas unidades y las tareas asociadas. Las tareas son relacionadas

a las unidades de interacción como las acciones esperadas por el emisor y el receptor.

Tomando en cuenta lo anterior, la comunicación entre los agentes está dada por el lenguaje de contenido *FIPA-ACL/SL0*, dado que es soportado por JADE [JADEPG2006] y es adecuado para el tipo de comunicación que se requiere entre los agentes del Sistema Multi-Agente. Como protocolo de interacción se utiliza el *FIPA-query Protocol*, en donde el agente que inicia la conversación tiene la intención de que el receptor le informe de algo concreto.

Se muestra en la figura 5.3 el diagrama de colaboración detallado de la interacción *solicitar_datos_proceso,* los demás diagramas se pueden consultar en el apéndice B.

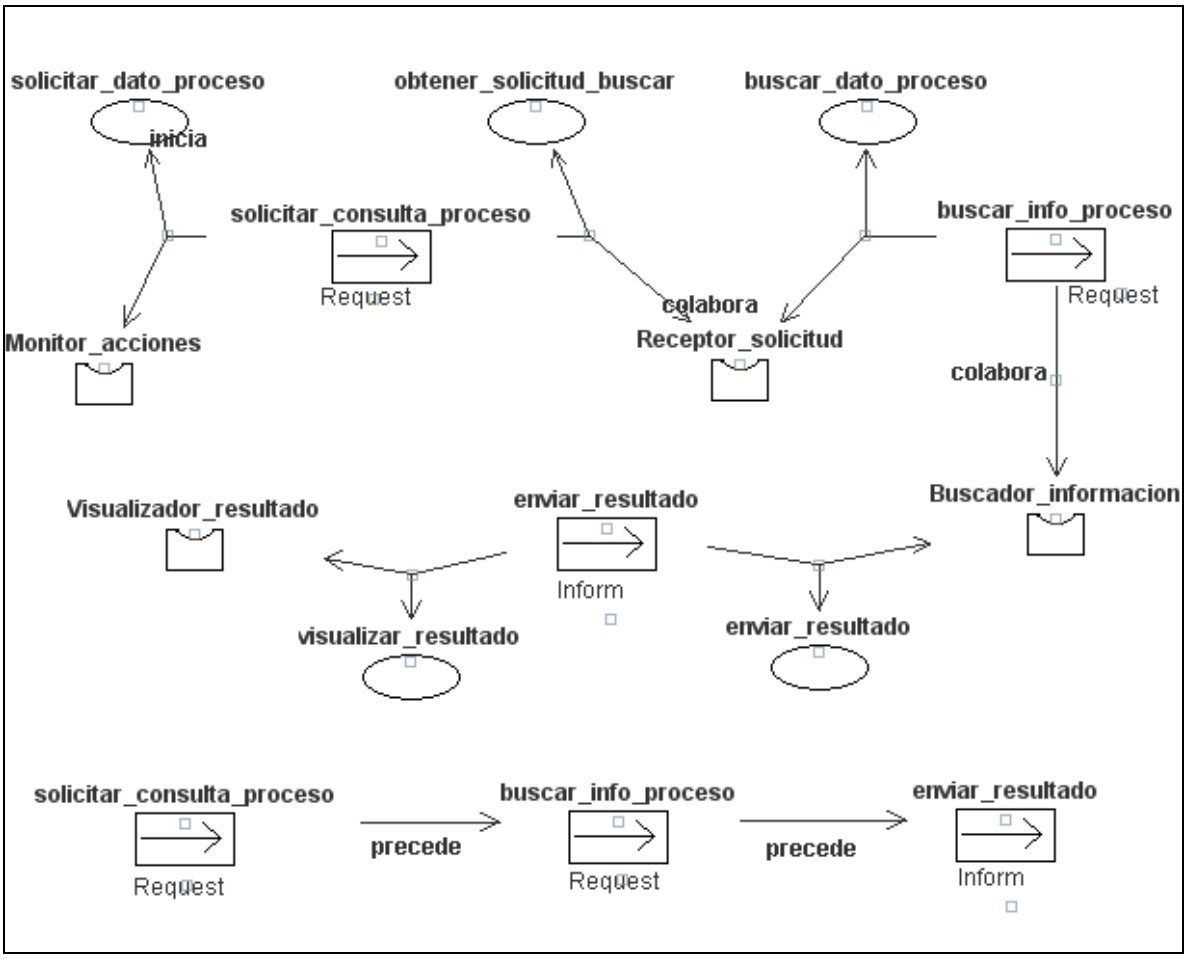


Figura 5.3: Diagrama de colaboración detallado para la interacción *solicitar_datos_proceso*.

5.3 DESCRIPCIÓN DEL ENTORNO DEL SISTEMA

Para complementar la descripción del entorno en el diseño, se muestra a continuación la percepción de los agentes en términos de las aplicaciones existentes.

5.3.1 Percepción de los agentes

Los agentes con percepción son el agente *Coordinador* y *Buscador* (Figura 5.4).

- El agente *Coordinador* percibe a través del elemento ServletAgente las acciones realizadas por el usuario en la interfaz gráfica de la herramienta HeAP MoProSoft.

- El agente *Buscador* percibe a las bases de datos utilizadas por el sistema mediante los métodos *consultar* e *insertar* de la interfaz (clases DAO) para el acceso a estos recursos.

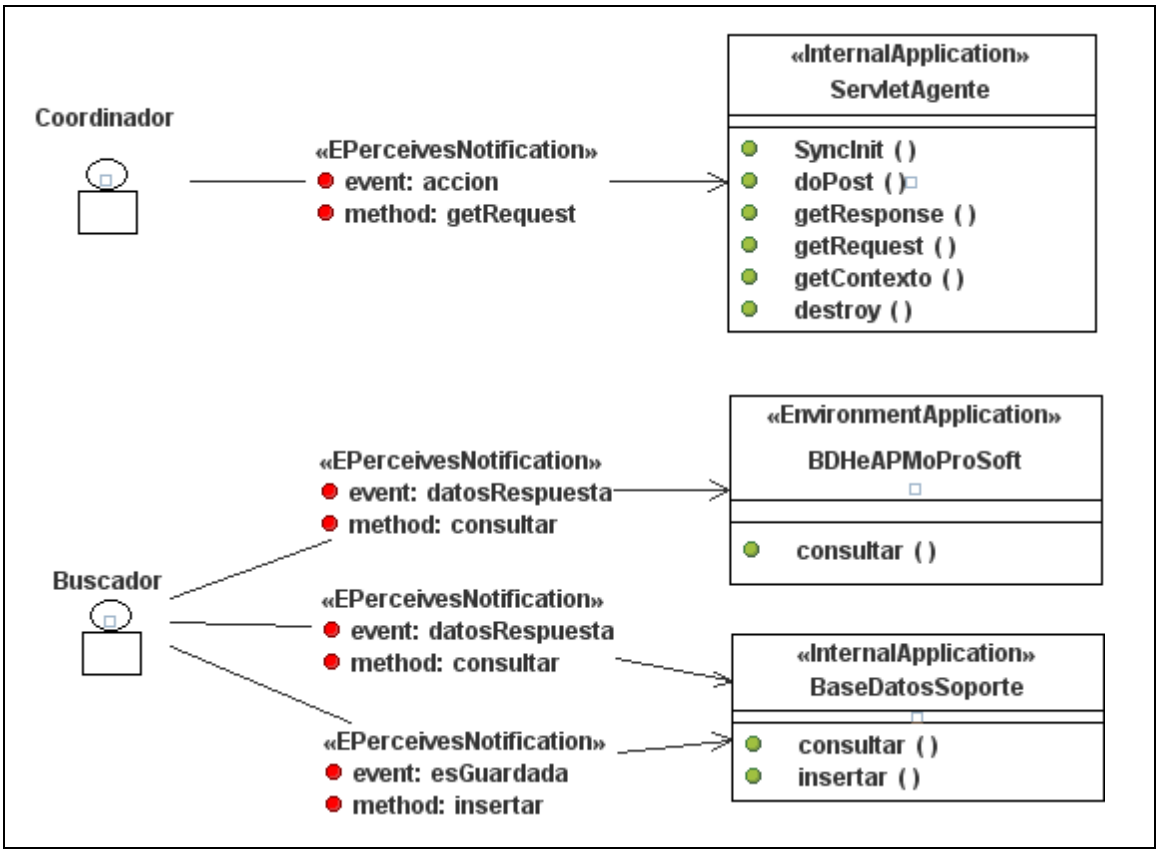


Figura 5.4: Percepción de los agentes.

5.4 DESCRIPCIÓN DE LA ORGANIZACIÓN DEL SISTEMA

Para complementar la descripción de la organización del sistema, se muestra a continuación la funcionalidad a través de la descripción de los flujos de trabajo identificados en el análisis y las dependencias sociales entre la organización, grupos y agentes.

5.4.1 Funcionalidad: descripción de flujos de trabajo

Los flujos de trabajo se modelaron descomponiéndolos para su mejor comprensión. En primer lugar, se identificaron las tareas que están relacionadas con los flujos de trabajo, luego se vieron sus relaciones y el orden en que las tareas se realizan, identificando qué tareas inician interacciones concretas. Finalmente, se asocian las tareas con los responsables de la ejecución de éstas (agentes o roles).

Cabe señalar que a continuación se muestra sólo la descripción del flujo de trabajo *Obtener_datos_proceso* y que las demás descripciones se pueden consultar en el apéndice B de este documento.

Flujo de trabajo: Obtener_datos_proceso

Define las tareas necesarias para proporcionar al usuario la consulta de información sobre el proceso que es responsable de acuerdo a su rol. Se inicia cuando el usuario solicita este servicio al seleccionar en la interfaz de HeAP MoProsoft la opción correspondiente y ha elegido el dato que desea consultar del proceso. En la tabla 5.2 se presentan por orden alfabético a las tareas que conforman a este flujo de trabajo.

Tarea	Descripción
buscar_dato_proceso	Procesa la búsqueda solicitada en la fuente de datos adecuada.
enviar_resultado	Envía el resultado obtenido al realizar la transacción solicitada en la fuente de datos.
obtener_solicitud_buscar	Obtener la solicitud de búsqueda y los datos necesarios para llevarla a cabo.
solicitar_dato_proceso	Solicita un dato del proceso. Esta tarea dispara una interacción que persigue el proporcionar al usuario la información solicitada del proceso.
visualizar_resultado	Procesa la visualización del resultado obtenido para el usuario.

Tabla 5.2: Tareas que conforman al flujo de trabajo *Obtener_datos_proceso*.

A continuación se presentan las cinco instancias del meta-modelo de Organización para la descripción detallada del flujo de trabajo.

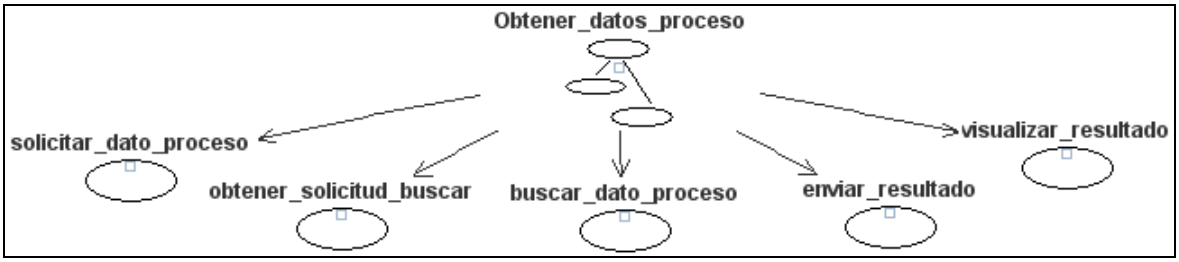


Figura 5.5: Tareas asociadas al flujo de trabajo *Obtener_datos_proceso*.

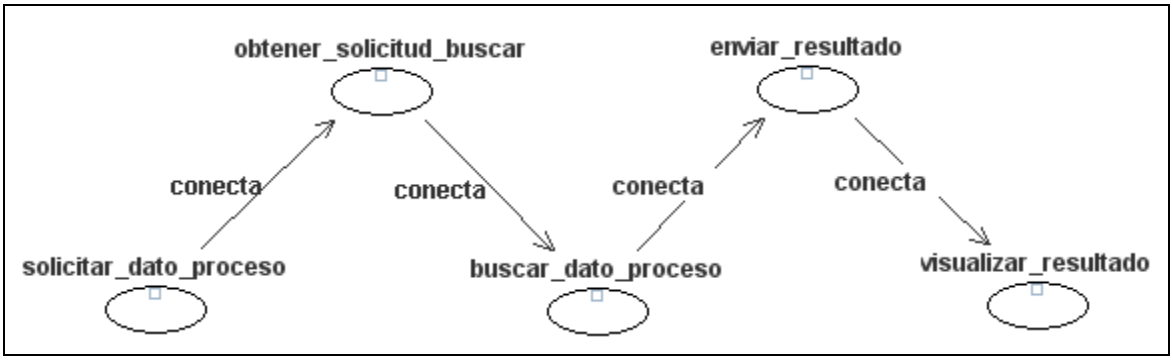


Figura 5.6: Dependencias entre las tareas en el flujo de trabajo *Obtener_datos_proceso*.

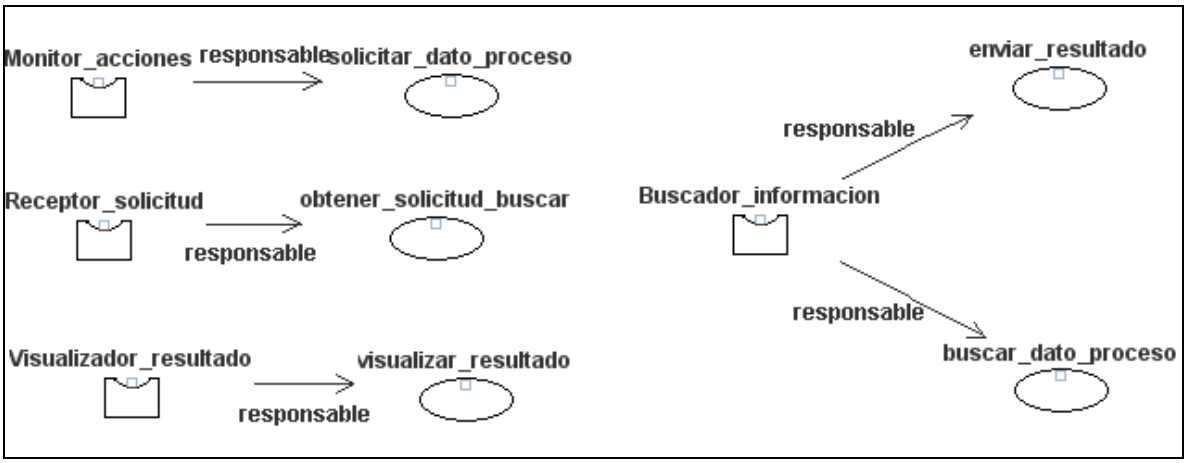


Figura 5.7: Roles responsables de las tareas del flujo de trabajo *Obtener_datos_proceso*.

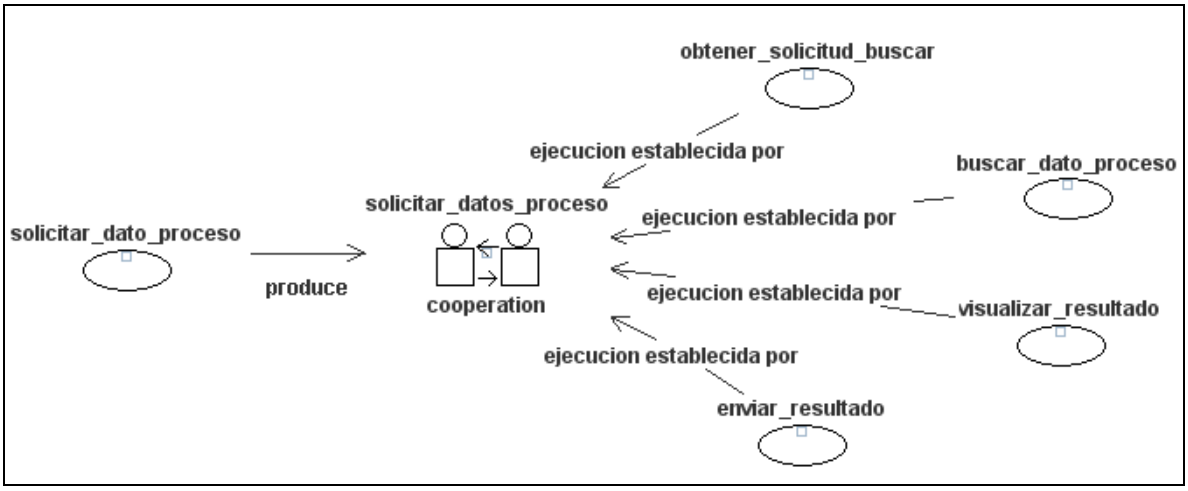


Figura 5.8: Relación de las tareas del flujo de trabajo *Obtener_datos_proceso* con la interacción *solicitar_datos_proceso*.

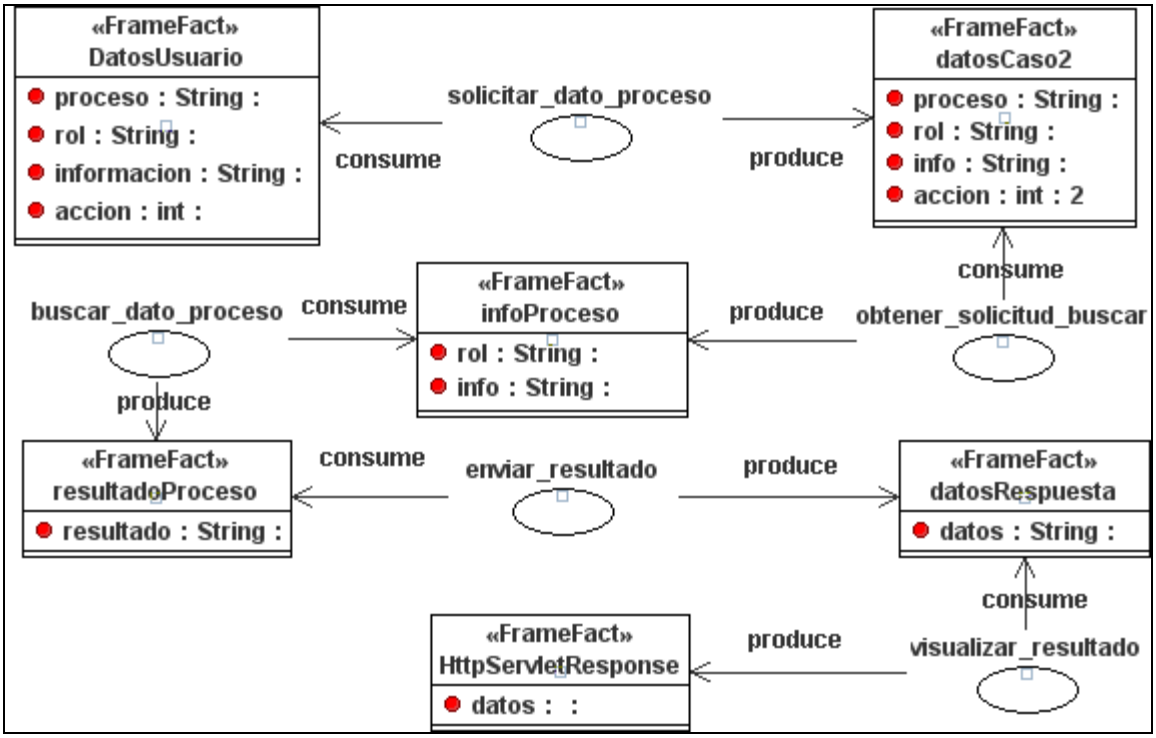


Figura 5.9: Descripción detallada del flujo de trabajo *Obtener_datos_proceso*.

5.4.2 Dependencias sociales

Se presenta en la figura 5.10 las relaciones entre la organización, grupos y agentes que configuran a las restricciones sociales que limitan a la interacción entre estas entidades. En concreto, se incluye la prioridad entre las relaciones sociales (tres niveles: organizativo, estructura de organización y de agente y rol), así como las relaciones de subordinación en donde se tiene una obediencia

incondicional (AGOSubordinacionIncondicional) que implica en obedecer todas y cada una de las órdenes.

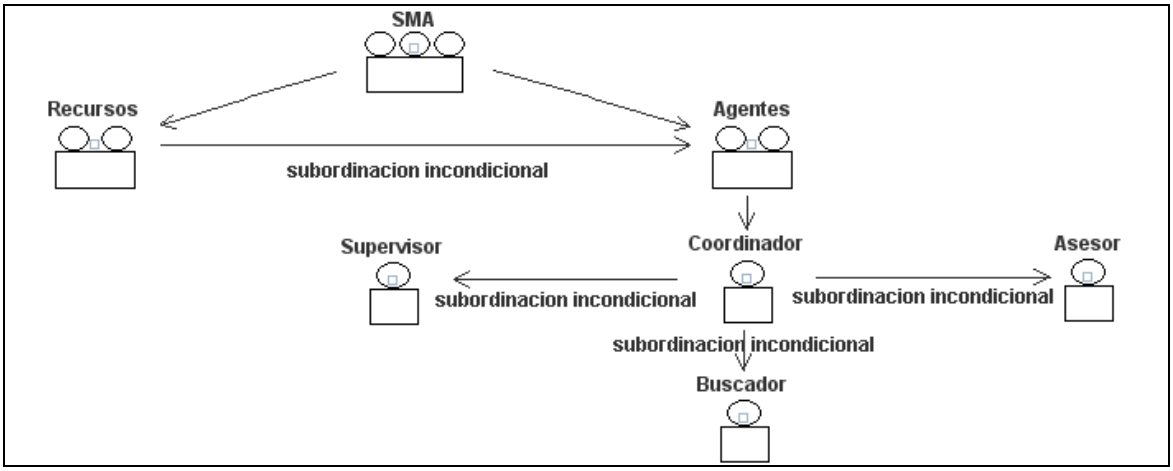


Figura 5.10: Dependencias sociales del sistema.

5.5 DISEÑO DEL SERVLETAGENTE

Para que el agente *Coordinador* cumpliera con el rol de monitor de las acciones realizadas por el usuario en la interfaz gráfica de la herramienta HeAP MoProSoft, fue necesario utilizar el elemento *ServletAgente*. Este elemento permite incluir al agente dentro de la interfaz para tener comunicación de los eventos del usuario.

El elemento *ServletAgente* está compuesto por tres clases: DirectoryServlet, Interaction y Synchronizer (Figura 5.11).

- ***DirectoryServlet***. Mediante esta clase se maneja un servlet que es una extensión del *HttpServlet* con referencias estáticas a los cuatro agentes del SMA. A través del método de inicialización de este servlet, se llama a un método estático sincronizado el cual permite la creación de los contenedores principal y secundario (Contenedor-1) y los agentes. Con el método doPost se crear un objeto interaction que envuelve la petición y la respuesta http, y sincronizar el servlet con el comportamiento del agente *Coordinador*. Finalmente, con el método de finalización del servlet se destruyen los contenedores y agentes creados.

- ***Synchronizer***. Esta clase es utilizada por el agente *Coordinador* para notificar al servlet que ha iniciado y está listo para manejar peticiones.

- ***Interaction***. Esta clase permite crear un objeto que envuelve la petición y la respuesta, y que es puesto en la cola de objetos del agente *Coordinador*.

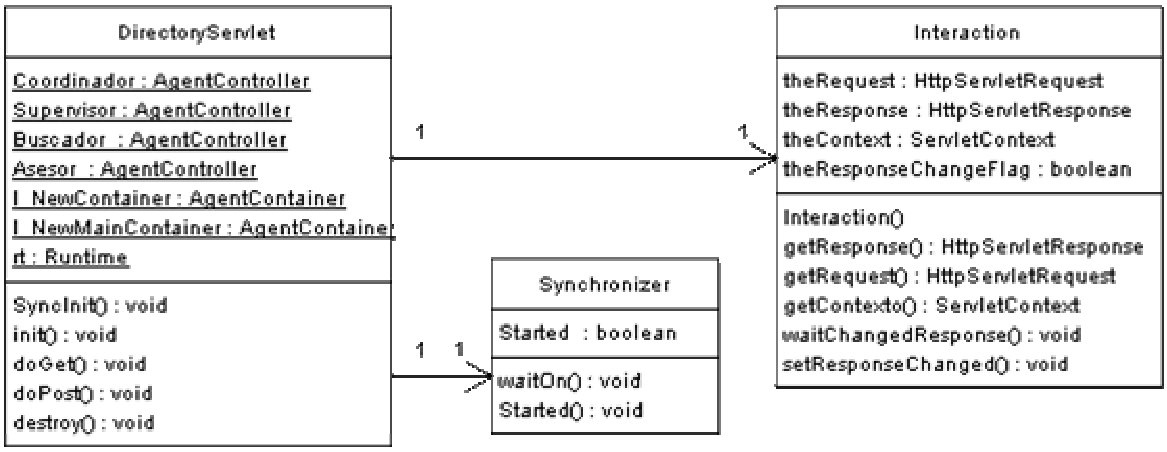


Figura 5.11: Diagrama de clases del elemento ServletAgente.

5.6 DISEÑO DE LA BASE DE DATOS DE SOPORTE

A continuación se presenta el diseño de la base de datos utilizada por el sistema: Base de Datos de Soporte. Cabe señalar que la otra base de datos utilizada, Base de Datos de HeAP MoProSoft (BDHeAPMoProSoft), ha sido desarrollada independientemente y sólo se realizarán consultas en ella.

5.6.1 Objetivos

Los objetivos que este repositorio deberá alcanzar a través de los agentes son:

1. Dar información en respuesta a peticiones de agentes.

2. Brindar la información establecida en MoProSoft sobre los procesos, en este caso es sobre el proceso de Administración de Proyectos Específicos.

3. Brindar una guía que indique la manera de cómo se puede llevar a cabo las tareas del modelo de procesos MoProSoft (específicamente las tareas realizadas en la herramienta HeAP MoProSoft por el Responsable de Gestión de Proyectos y el Responsable de la Administración del Proyecto Específico.

4. Aumentar el conocimiento del usuario al proveerle de información que probablemente no conoce o en el momento no recuerda.

5.6.2 Recolección de datos

En cuanto a la recolección de la información para esta base de datos se han considerado las siguientes características que deben cumplir tanto los datos referentes a MoProSoft como los relacionados a los de soporte.

Datos MoProSoft

- Los datos deben cubrir la información referente a los procesos especificados en MoProSoft.

- Dichos datos del proceso son referentes a: objetivos, indicadores, responsable, autoridad, procesos relacionados, entradas, salidas, productos internos, roles involucrados, verificaciones, validaciones, actividades y tareas.

Datos de soporte:

- Los datos deben cubrir el rango de tareas que HeAP MoProSoft soporta o lleva a cabo para los roles RGPY y RAPE. Puesto que sólo se abarca ciertas tareas, la base de datos sólo soporta algunas sugerencias en cómo realizarlas; sin embargo puede ser almacenada más información conforme la herramienta sea utilizada, aumente las tareas para los usuarios RGPY y RAPE o se decida dar soporte a otro rol. En la tabla 5.3 se muestra las tareas comprendidas en HeAP MoProSoft correspondientes a los roles de los usuarios.

Identificador	**Descripción tarea**	**Rol**
A2.2.	Realizar actividades del Plan de Proyectos: • Generar Registro de Proyecto. • Generar Descripción del Proyecto. • Asignar Responsable de Administración del Proyecto Específico con base a la Asignación de Recursos.	RGPY
A1.3.	Definir conjuntamente con el Cliente el Protocolo de Entrega de cada uno de los entregables especificados en la Descripción del Proyecto.	RAPE
A1.4.	Identificar el número de ciclos y las actividades específicas que deben llevarse a cabo para producir los entregables y sus componentes identificados en la Descripción del Proyecto. Identificar las actividades específicas que deben llevarse a cabo para cumplir con los objetivos del proyecto, definir las actividades para llevar a cabo revisiones periódicas al producto o servicio que se está ofreciendo y para efectuar revisiones entre colegas. Identificar las actividades para llevar a cabo el Protocolo de Entrega. Documentar el resultado como Ciclos y Actividades.	RAPE
A1.6.	Establecer el *Tiempo Estimado* para desarrollar cada actividad.	RAPE

Identificador	Descripción tarea	Rol
A1.7.	Elaborar el *Plan de Adquisiciones y Capacitación*, definiendo las características y el calendario en cuanto a recursos humanos, materiales, equipo y herramientas, incluyendo la capacitación requerida para que el equipo de trabajo pueda desempeñar el proyecto.	RAPE
A1.8.	Conformar el *Equipo de Trabajo*, asignando roles y responsabilidades basándose en la *Descripción del Proyecto.*	RAPE
A1.10.	Documentar el Costo Estimado del proyecto.	RAPE
A1.11.	Identificar los riesgos que pueden afectar el proyecto, que contemple riesgos relacionados con el equipo de trabajo incluyendo al Cliente y a los usuarios, riesgos con la tecnología o la metodología, riesgos con la organización del proyecto (costo, tiempo, alcance y recursos) o riesgos externos al proyecto. Identificar la probabilidad e impacto de cada riesgo estimando sus implicaciones en los objetivos del proyecto. Priorizar los efectos de los riesgos sobre los objetivos del proyecto. Documentar en el *Plan de Manejo de Riesgos* o actualizarlo.	RAPE

Tabla 5.3: Tareas implementadas en HeAP MoProSoft para los usuarios RGPY y RAPE.

- Sobre dichas tareas, los datos deben ilustrar cómo llevar a cabo la tarea con métodos recomendables y conocidos. Si hay varias formas de realizar la labor, todas deben ser registradas.

Resumiendo, los datos deben contener al menos la información necesaria para su indexación de acuerdo a las situaciones en las que pueden proveer consulta o guía y contener suficiente información para ser entendidos y aprovechados por el usuario.

Para la extracción de la información anteriormente descrita, se identificó que se pueden utilizar como fuentes a la documentación del Modelo de Procesos para la Industria de Software versión 1.3, la guía del PMBoK [PMBOK2004] y el conocimiento de un experto humano (consultor que cuenta con experiencia en la implantación de la administración de proyectos de software).

5.6.3 Modelo lógico

La base de datos construida en el presente proyecto es relacional y contiene los datos que capturan la información de los procesos de MoProSoft y las sugerencias dadas por los expertos. La estructura de la base de datos se muestra en la figura 5.12 donde se define el modelo lógico con las entidades definidas.

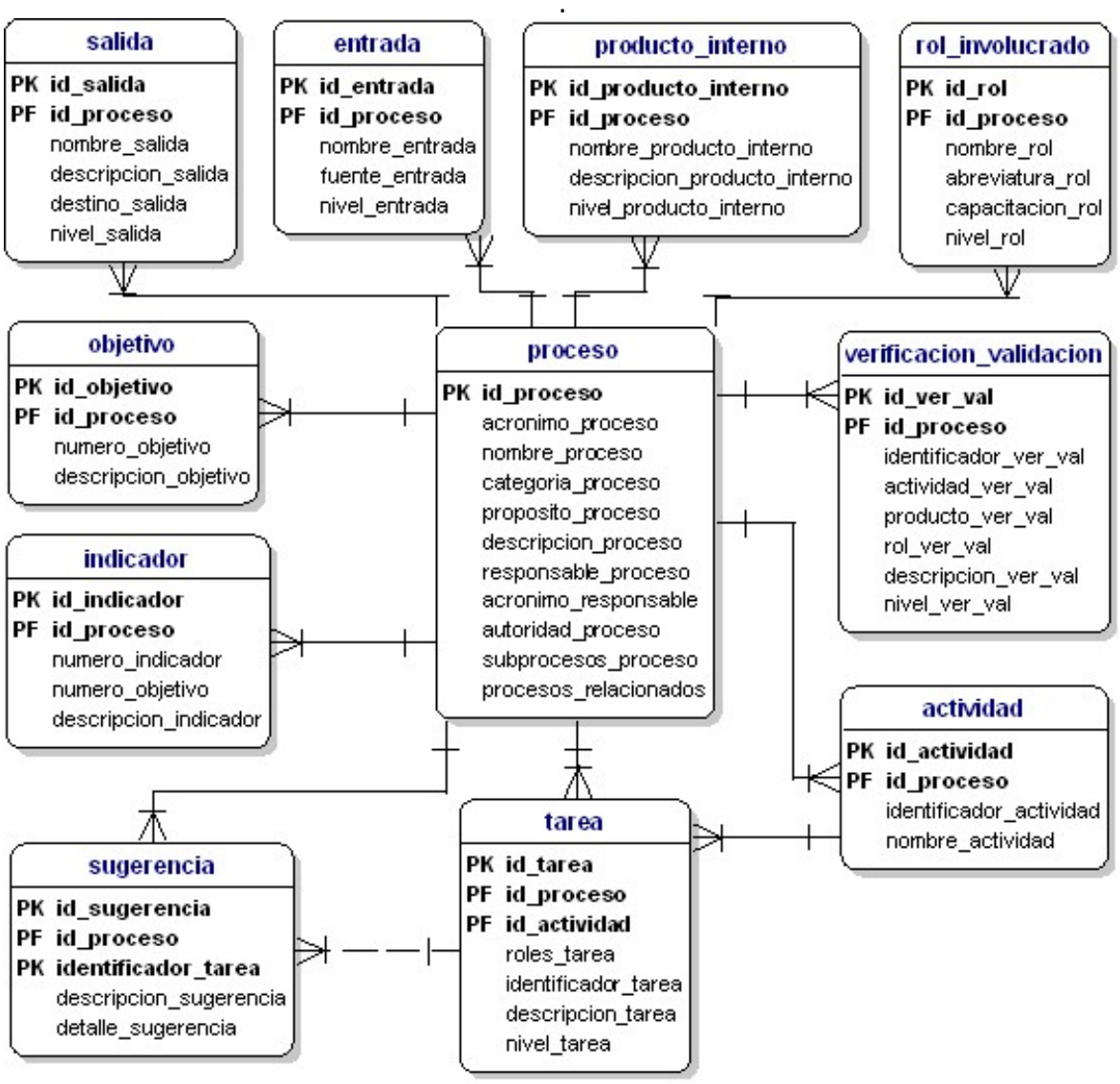


Figura 5.12: Diagrama del Modelo Lógico de la Base de Datos de Soporte.

En el apéndice C se describen las tablas que conforman la base de datos, así como sus atributos.

5.7 USO DEL PATRÓN DAO PARA EL ACCESO A LOS DATOS

La capa del modelo de la arquitectura de la herramienta HeAP MoProSoft se encarga de la persistencia de los datos, la consistencia de éstos y la creación de los objetos del negocio. La persistencia es implementada con PostgreSQL 8.1 a través de bases de datos relacionales. Para abstraer y encapsular todos los accesos a la fuente de datos, logrando así desacoplar la lógica de negocios de la lógica de acceso a datos, se hace uso del patrón DAO (Data Access Object).

En este patrón se tiene los siguientes componentes [ALUR2003]:

- *BussinessObject.* Es el objeto de negocio que representa al solicitante de datos que requiere acceder a la fuente de datos para realizar acciones de consulta y almacenamiento.

- *DataAccessObject:* Este objeto abstrae la implementación del acceso a datos a los objetos de negocio.

- *DataSource:* Representa la implementación de la fuente de datos.

- *TransferObject:* Es el objeto que representa al transportador de datos. Es creado y llenado por el objeto DAO y puede ser leído y modificado por el objeto de negocios.

Para el Sistema Multi-Agente, en la aplicación de este patrón (ver figura 5.13) se definen los componentes de la tabla 5.4.

Clase/interfaz	**Descripción**
InterfaceDAO	Interfaz que representa el comportamiento genérico de cualquier DAO, se define todas las operaciones de acción (insertar) y consulta.
DAOGeneral	Clase que contiene servicios comunes para el acceso a la fuente de datos, tales como cargar driver, identificación, obtener conexión y cerrar conexión.
DAONombreSMA	Clase que implementa *InterfaceDAO* y que además hereda de *DAOGeneral*.
VONombreSMA	Clase que representa el TransferObjet o ValueObject (beans) y que únicamente contiene datos y métodos de lectura y escritura. Existe uno por cada tabla que sea necesaria.

Tabla 5.4: Componentes de la implementación del patrón DAO por el SMA.

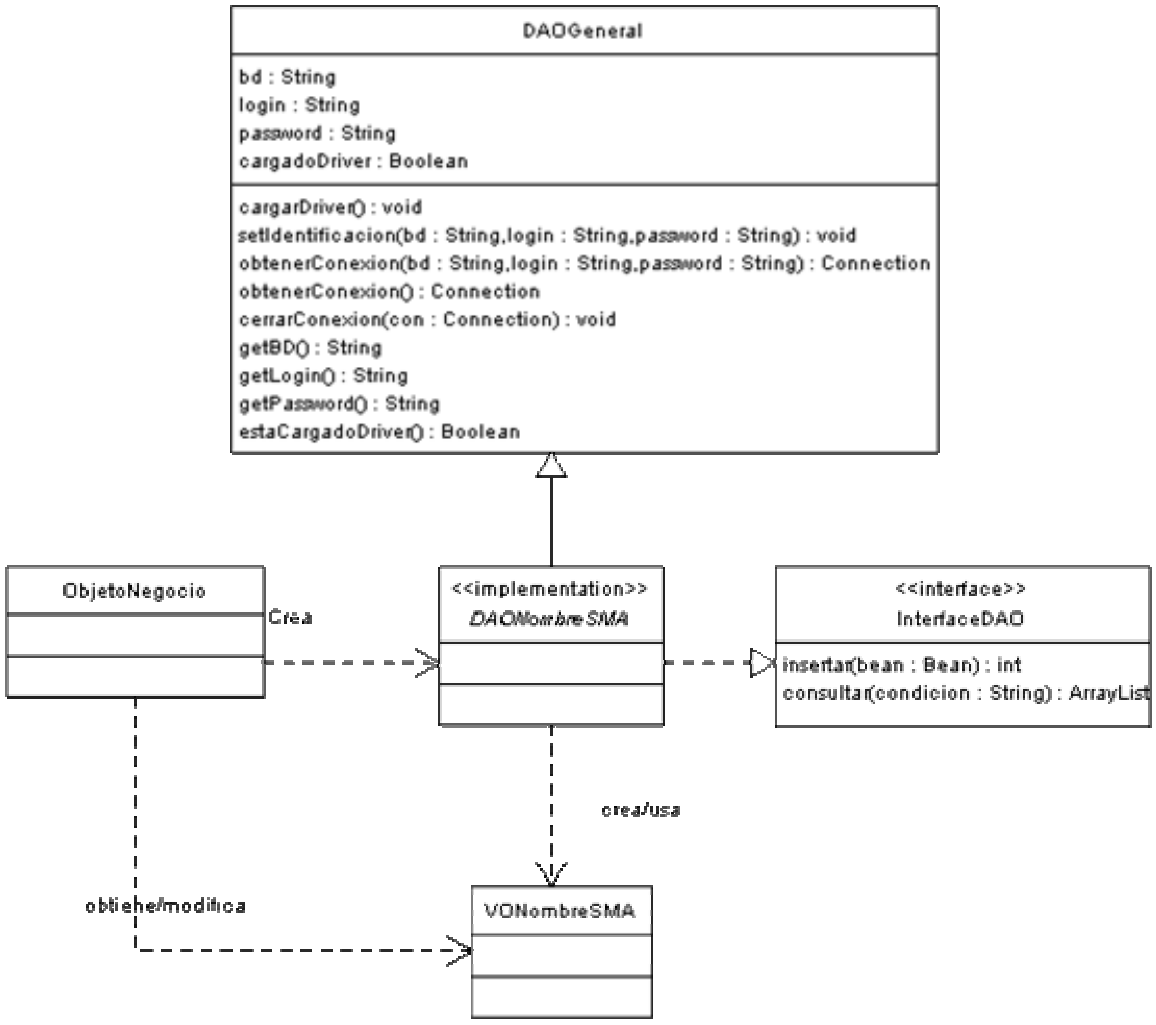


Figura 5.13: Diagrama de clases para la implementación del patrón DAO por el SMA.

En términos generales el procesamiento consiste en los siguientes pasos:

1. El objeto de negocio (agente Buscador) crea al objeto DAO y le solicita los datos que requiere.
2. El DAO creado responde a la llamada accediendo a la fuente de datos para recuperar los datos y los deposita en un objeto *ValueObject* que crea.
3. El DAO devuelve al objeto de negocio el ValueObject.
4. El objeto de negocio trabaja con estos datos. Si la acción que necesita es de guardar manda llamar al DAO para que actualice la fuente con los datos contenidos en el *ValueObject*.

Cabe señalar, que para agregar nuevas sugerencias a través del método insertar fue necesario hacer uso de la biblioteca de la clase MultipartRequest para guardar archivos en el servidor como parte de la descripción detallada de las sugerencias almacenadas. Estas clases permiten manejar peticiones del tipo multipart/form-data para subir archivos [HUNTER2007].

Saber no es suficiente, debemos aplicar. Desear no es suficiente, debemos hacer.

JOHANN W. VON GOETHE

Capítulo 6

Construcción del Sistema Multi-Agente

El presente capítulo aborda la construcción del Sistema Multi-Agente. Se describe la arquitectura general del sistema y la explicación del desarrollo de cada uno de los componentes definidos en la fase del diseño, así como la definición de las pruebas y los resultados obtenidos.

6.1 INTRODUCCIÓN

De lo expuesto en los capítulos precedentes se desprende la especificación, el análisis y diseño del Sistema Multi-Agente a desarrollar. Como resultado del proceso anterior y mediante varios modelos, se identificaron y definieron los agentes que conformarían al sistema, los mensajes que intercambiarían y la descripción de su comportamiento. De igual manera, se determinó el diseño de la Base de Datos de Soporte para que los agentes pudieran cumplir con sus objetivos identificados.

Lo anterior deriva en la etapa final del proyecto de tesis, que es la implementación del Sistema Multi-Agente, actividad que dado el diseño generado consiste en la traducción de módulos, interfaces, algoritmos y otras construcciones definidas en modelos, en estructuras de datos, procedimientos y secuencias de instrucciones directamente soportadas por el lenguaje de programación seleccionado. Como ya se ha apuntado, la programación del Sistema Multi-Agente se basa en gran medida en la utilización del marco de trabajo JADE, por lo que el lenguaje de programación es Java.

A continuación, se describe la arquitectura general del sistema, la implementación de cada agente y se describen las pruebas así como los resultados de las mismas.

6.2 ARQUITECTURA GENERAL DEL SISTEMA MULTI-AGENTE

Como se mencionó en la sección 2.2.2 del capítulo 2, existen varias arquitecturas de agentes de acuerdo a sus características. Dentro de una clasificación presentada [WOOLDRIDGE1995], se ubica al Sistema Multi-Agente en una arquitectura reactiva puesto que actúa siguiendo un modelo estímulo-respuesta, donde los estímulos recibidos del entorno (interfaz gráfica de HeAP MoProSoft) son procesados por capas especializadas que directamente responden con acciones a dichos estímulos.

Con respecto a esto último, cabe mencionar que la capacidad de reacción está dada por: algún procesamiento interno (por medio de comportamientos) o alguna interacción entre los agentes al enviar mensajes (utilizando el lenguaje de comunicación de agentes ACL). Además, a través de la plataforma JADE en cuanto a su servicio de mensajería, los agentes deciden qué y cuándo leer los mensajes al administrar su cola privada de mensajes.

Tomando en cuenta lo anterior y utilizando las arquitecturas de HeAP MoProSoft y JADE (vistas en los capítulos 1 y 3 respectivamente) se generó una arquitectura global (figura 6.1) donde se observa la composición del Sistema Multi-Agente y la manera en que interactúa con los demás componentes.

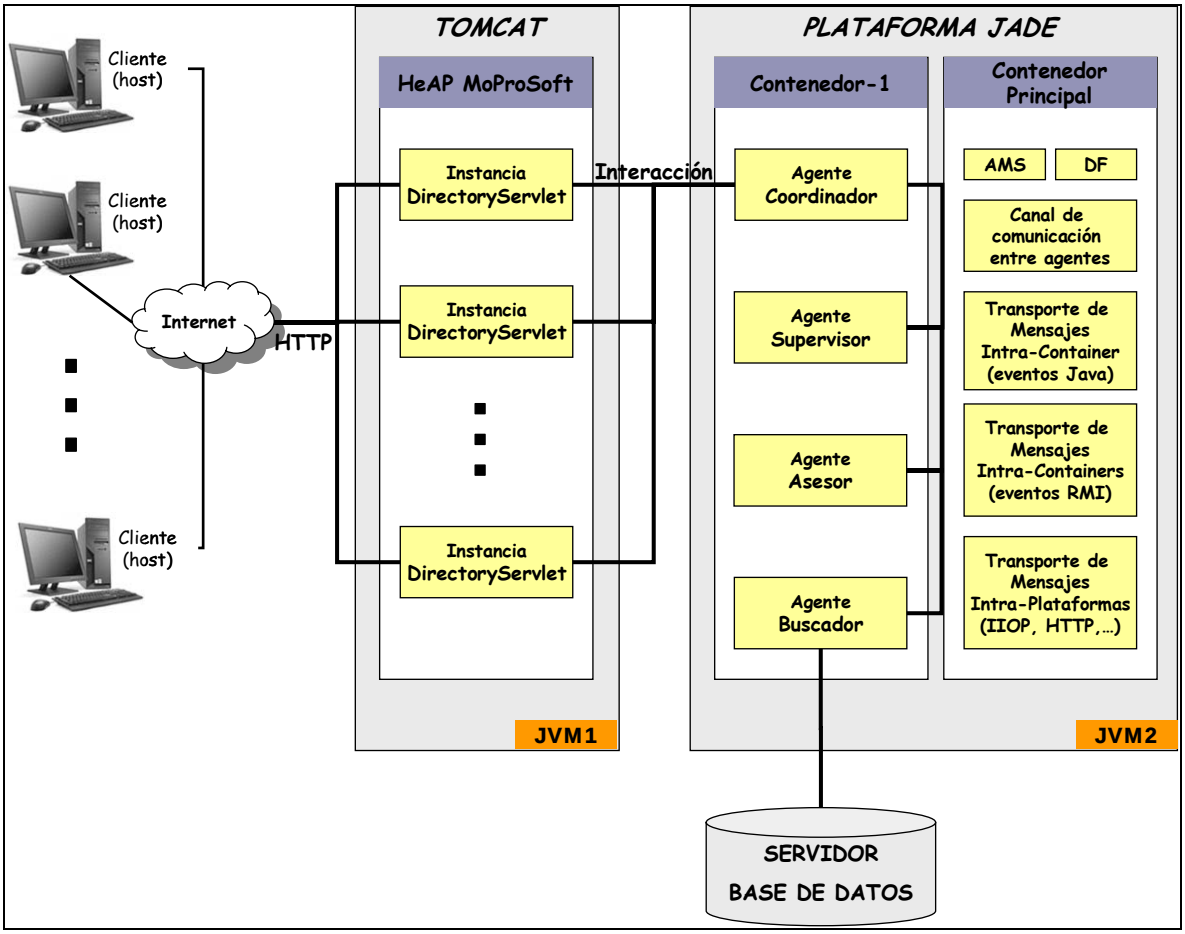


Figura 6.1: Arquitectura general del Sistema Multi-Agente.

Como se puede apreciar, se ejecutan dos JVM en el servidor: una para Tomcat (con HeAP MoProSoft funcionando) y la otra con la plataforma JADE, que a su vez contiene el contenedor principal con los componentes necesarios (agentes AMS, DMS, el DF y el registro RMI) y un secundario llamado *Contenedor-1* que alberga a los cuatro agentes definidos: Coordinador, Supervisor, Buscador y Asesor.

La arquitectura de JADE, está definida para poder ser distribuida entre diferentes clientes (*host*) incluso con sistemas operativos distintos, permite que cada cliente tenga una aplicación Java (por lo tanto una máquina virtual) ejecutándose en él. Así, cada una de las máquinas virtuales se convierte en un contenedor básico de agentes que provee el ambiente necesario para la ejecución de dichos agentes.

En cuanto al elemento *DirectoryServlet,* es el servlet que se implementó para ligar al agente Coordinador con la interfaz gráfica de la aplicación Web HeAP MoProSoft. Se ejecuta en el servidor Tomcat cuando el usuario realiza alguna acción en la interfaz relacionada con los servicios del SMA. Una vez que el

Sistema Multi-Agente generó la información, el *DirectoryServlet* contesta la petición *httpRequest* mostrando la información al usuario.

Para el acceso a la fuente de datos (Servidor de Base de Datos), el agente Buscador es el objeto de negocio encargado de ésta función mediante la utilización de las clases definidas bajo el patrón DAO.

El proceso para atender una solicitud del usuario es el siguiente:

1. En el navegador del usuario sucede un evento que genera un mensaje POST (usuario realiza una acción en la interfaz HeAP MoProSoft).
2. Se llama a una función para informar al DirectoryServlet que ha ocurrido una acción y se le mandan los parámetros. Verifica que los contenedores principal y secundario se han inicializado para los agentes, en caso de no estar inicializados, los crea juntos con los agentes (Coordinador, Supervisor, Asesor y Buscador).
3. El DirectoryServlet notifica al agente Coordinador del evento quien se encarga de realizar alguna acción para cumplir con sus objetivos (mandar mensajes a los demás agentes en solicitud de algún servicio o el procesamiento de algún dato interno).
4. Después del proceso de la solicitud, el agente correspondiente envía el resultado al agente Coordinador y éste a su vez al DirectoryServlet.
5. El DirectoryServlet contesta la petición *httpRequest* mostrando la información generada por el Sistema Multi-Agente.

Finalmente, puesto que el servidor del servlet levanta un hilo por cada petición en esta arquitectura, se puede manejar múltiples peticiones en paralelo.

6.3 DESARROLLO DE LOS COMPONENTES

Para la codificación de los elementos que conforman al Sistema Multi-agente se utilizó el entorno integrado de desarrollo Eclipse SDK 3.2.2 con Java JDK 1.6. Además, se anexaron las bibliotecas que proveen las clases necesarias.

La estructura principal de los cuatro agentes está dada por las clases *agent y behaviour* de Jade*.* Mediante estas clases se representa al agente y se determinan los comportamientos (ejecución de tareas en respuesta a eventos externos o que el mismo genera para lograr un estado deseado) que va a manifestar.

Para simplificar y comprender mejor esta estructura, los elementos principales que la conforman son:

- **Variables de inicialización.** Se define la lista de los agentes que conoce y las estructuras de datos para manipular los mensajes y su contenido.

- **Eventos.** Se define uno o varios comportamientos del tipo cíclico mediante los cuales se recibe los mensajes ACL. Estos comportamientos se implementan como una máquina de estados con bloqueos a la espera de recibir mensajes.
- **Métodos**. Se definen métodos para la interacción con la plataforma JADE: *takedown* (ejecutado cuando se muere el agente), *enviarMensaje* (envío de los mensajes ACL necesarios). También métodos que varían para cada agente puesto que son para lograr sus objetivos.

Por otro lado, a continuación se presentan ciertas particularidades entre los agentes debido a los objetivos que de manera particular persigue cada uno.

Coordinador

- Con ayuda del DirectoryServlet (descrito en el capítulo 5) implementa un comportamiento que recibe las acciones del usuario y mediante una estructura de decisión, envío y recepción de mensajes con los demás agentes pide el servicio.
- Procesa el resultado obtenido para que sea desplegado en una página Web.
- Si la acción corresponde al caso de agregar sugerencia y existe un archivo que detalle la sugerencia, se encarga del almacenamiento de dicho archivo en el servidor. Es importante mencionar que sólo se puede subir un archivo de tipo html por sugerencia y una imagen.

.

Buscador

- Mediante una estructura de recepción y decisión ejecuta métodos (infoProceso, infoSugerencia, guardarSugerencia, infoTareas) de búsqueda o almacenamiento en la base de datos adecuada perteneciente al entorno del SMA. Para el acceso a los datos hace uso de las clases definidas en la implementación del patrón de diseño DAO (descrito en el capítulo 5).
- Envía el resultado al agente que se lo solicitó.

Supervisor y Asesor

- Mediante una estructura de recepción y decisión determina la acción específica a realizar de acuerdo al contenido del mensaje recibido.
- Si el servicio que se va a proporcionar requiere de información de una base de datos, manda el mensaje apropiado al agente Buscador.
- Si el mensaje proviene del agente Buscador, entonces se analizan los datos del contenido del mensaje, se formatean de acuerdo al servicio brindado y se envía la información al agente Coordinador.

Por último, cabe mencionar que la construcción de las clases se realizó de manera satisfactoria y se utilizaron todos los elementos que se han generado a lo largo de la tesis. El código de cada componente se puede consultar en el disco compacto anexo a este documento.

6.4 INTEGRACIÓN CON HEAP MOPROSOFT

Para la integración del Sistema Multi-Agente con la aplicación HeAP MoProSoft es necesario primero listar los componentes desarrollados así como los que son necesarios para su funcionamiento. En la tabla 6.1 se describe cada elemento que conforma al Sistema Multi-Agente.

Elemento	**Descripción**
sma.Agentes	Es el paquete conformado por cuatro clases que implementan los agentes: Coordinador.java, Supervisor.java, Buscador.java y Razonador.java.
sma.ServletAgente	Es el paquete conformado por tres clases que hacen posible la integración del agente Coordinador en la interfaz gráfica de HeAP MoProSoft: el servlet DirectoryServlet y los componentes Interaction.java y Sinchronizer.java.
sma.BD_SMA	Es el paquete conformado por dos paquetes: Bean que tiene todas las clases correspondientes a los ValueObject (beans) y DAO que tiene todas las clases DAO necesarias que implementan a las clases genéricas *InterfaceDAO.java* y *DAOGeneral.java.*
Bibliotecas	Son las bibliotecas que contienen las clases necesarias para el funcionamiento y uso de JADE, MultipartRequest, y las bases de datos.
BaseDatosSoporte	Es la carpeta conformada por los archivos que detallan algunas de las sugerencias almacenadas en la Base de Datos de Soporte*.*
sma	Es la carpeta conformada por los archivos html, jsp, hoja de estilo e imágenes que implementan la parte gráfica para el usuario para las funcionalidades ofrecidas por el Sistema Multi-Agente.

Tabla 6.1: Componentes que conforman al Sistema Multi-Agente.

El SMA se integra mediante la realización de los siguientes pasos:

1. Agregar todos los paquetes y carpetas de la tabla 6.1 bajo los directorios (ver tabla 6.2) ubicados dentro del paquete principal que conforma a la aplicación HeAP MoProSoft.

Directorio	**Elemento**
Carpeta raíz	La carpeta *BaseDatosSoporte.*
web	La carpeta *sma*
src	El paquete *sma* conformado por los paquetes: *Agentes, Servlet* y *BD_SMA*
WEB-INF/lib	Los archivos jar correspondientes a las bibliotecas de: • *JADE*: jade.jar, jadeTools.jar, iiop.jar, http.jar y commons-codec-1.3 • *MultipartRequest*: cos.jar • *JDBC*: sqljdbc.jar

Tabla 6.2: Directorios para la integración de los componentes del Sistema Multi-Agente.

2. Agregar en el archivo web.xml de HeAP MoProSoft el DirectoryServlet. El código anexo necesario es incluido y descrito en el apéndice E.

3. Agregar la Base de Datos de Soporte en el Servidor de Base de Datos.

4. Finalmente, es necesario agregar en la interfaz gráfica de HeAP MoProSoft el código html y javaScript correspondiente al menú definido para hacer uso del Sistema Multi-Agente. El código es incluido y descrito en el apéndice E.

Es importante mencionar que hasta este paso se consideraba que la herramienta HeAP MoProSoft estaría lista para integrarle el sistema Multi-Agente. Sin embargo, al momento de redacción, no contaba con el desarrollo suficiente como para llevar a cabo esta fase de integración.

El hecho anterior genera un resultado no satisfactorio desde el punto de vista de que era un requerimiento el integrar ambas aplicaciones. No obstante, se puede encontrar justificación en que se deja todo los elementos necesarios para que una vez terminada HeAP MoProSoft se pueda realizar sin ningún problema debido a la estructura e independencia definida desde su diseño.

6.5 PRESENTACIÓN DEL PROTOTIPO

Se presenta el prototipo del Sistema Multi-Agente con la finalidad de mostrar su funcionalidad y la forma de utilizarlo dentro de HeAP MoProSoft. Para ello, se hace la demostración del sistema ejecutando la aplicación desde Eclipse y visualizando la herramienta a través de un navegador de Internet.

La interfaz de HeAP MoProSoft con el menú del Sistema Multi-Agente es la que se aprecia en la figura 6.2 y a continuación se muestra cada una de las funcionalidades desarrolladas.

Figura 6.2: Interfaz de HeAP MoProSoft con el menú del Sistema Multi-Agente.

Información del proceso

En el menú del Sistema Multi-Agente al dar clic sobre la opción *Información del proceso*, se despliega en una nueva ventana una lista desplegable con los temas que son parte de la descripción del proceso correspondiente a su rol. Al seleccionar el tema y presionar el botón *consultar* se lleva a cabo todo el procesamiento descrito en las secciones anteriores, y se despliega la información adecuada en la ventana emergente (figura 6.3).

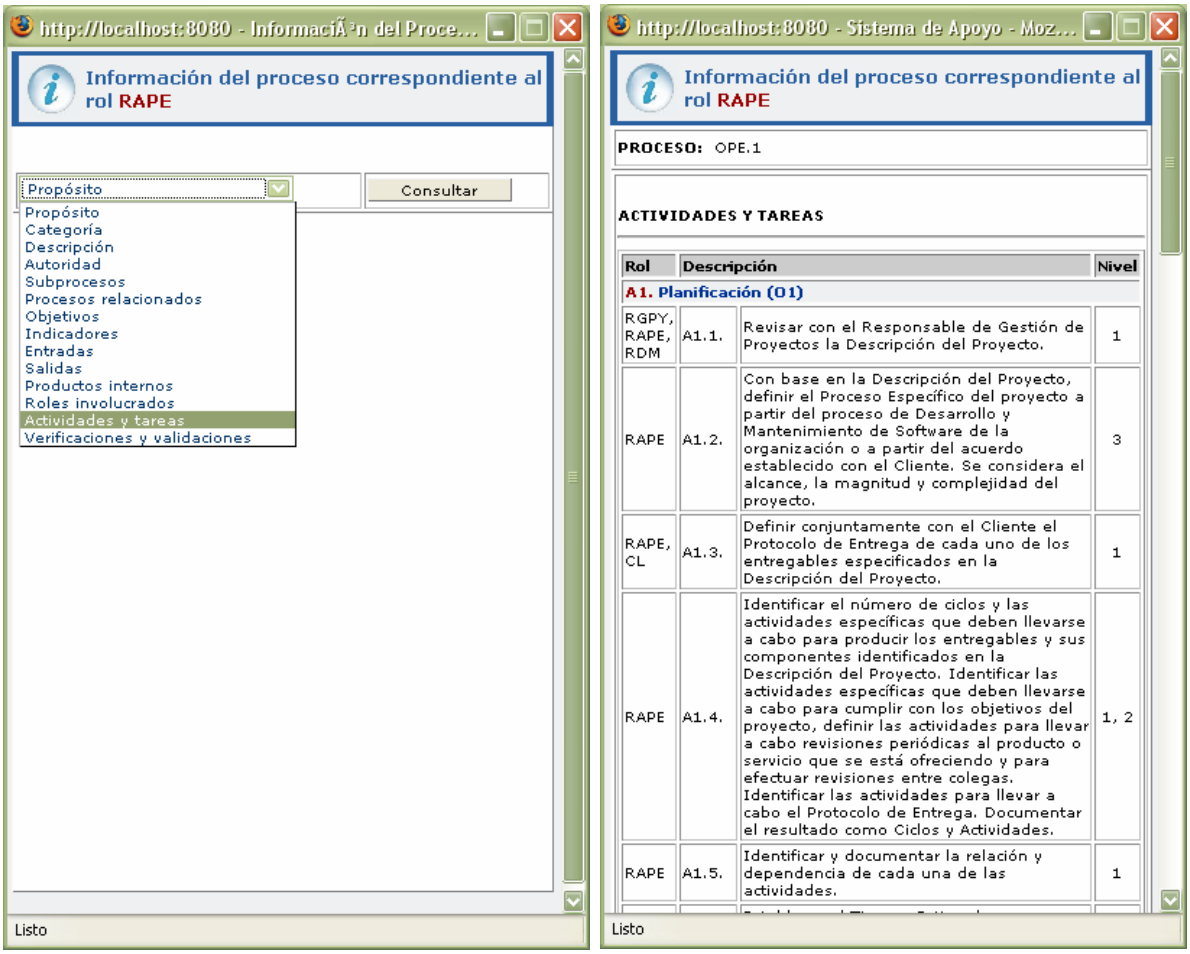


Figura 6.3: Página web generada a la petición del usuario Información del proceso.

Información de control

En el menú del Sistema Multi-Agente al dar clic sobre la opción *Información de control*, se lleva a cabo todo el procesamiento descrito en las secciones anteriores, y se despliega en una nueva ventana las tareas realizadas por el usuario y sus estados (figura 6.4).

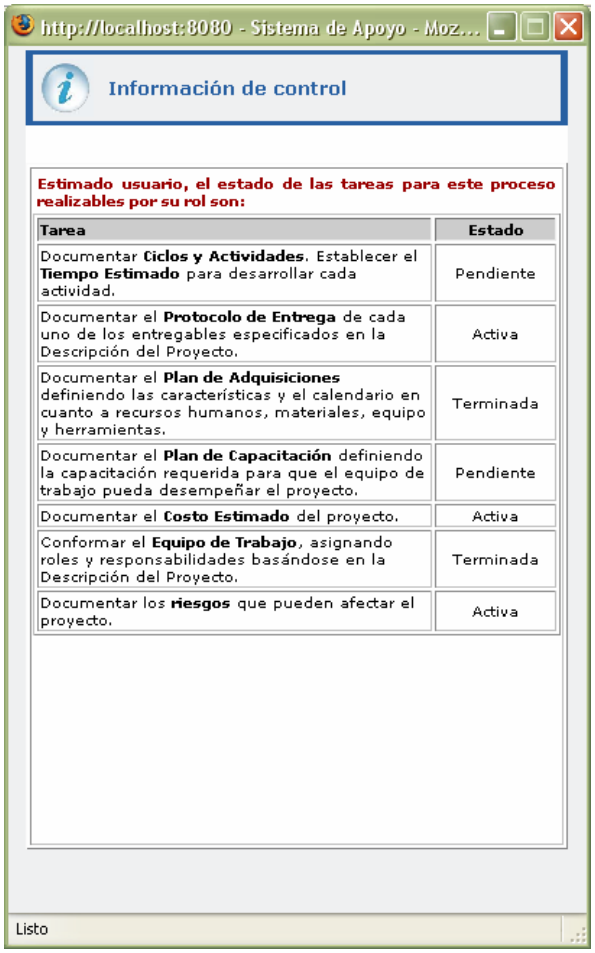


Figura 6.4: Página web generada a la petición del usuario Información de control.

Información de soporte

En el menú del Sistema Multi-Agente al dar clic sobre la opción *Información de soporte*, se despliega en una nueva ventana una lista desplegable con las tareas que son realizadas por el usuario en el sistema de acuerdo a su rol. Al seleccionar la tarea y presionar el botón *consultar,* se despliega una lista con las sugerencias registradas y que son correspondientes a la tarea que fue seleccionada. Al dar clic en la opción de *ver detalle* de una sugerencia, se despliega la información adecuada en la ventana emergente (figura 6.5).

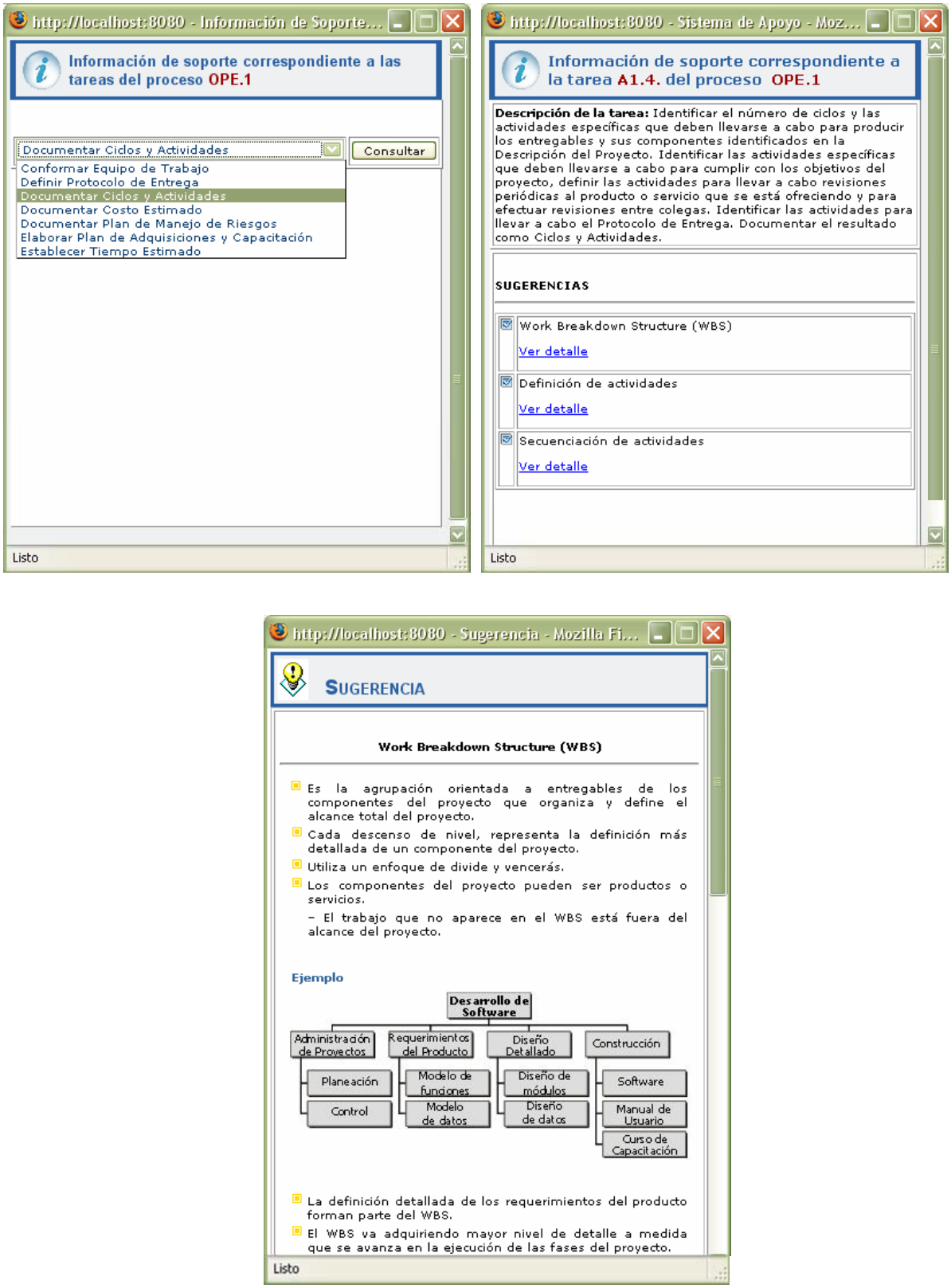


Figura 6.5: Página web generada a la petición del usuario Información de soporte.

Agregar sugerencia

En el menú del Sistema Multi-Agente al dar clic sobre la opción *agregar sugerencia*, se despliega en una nueva ventana una lista desplegable con las tareas que son realizadas por el usuario en el sistema de acuerdo a su rol, un campo para introducir la descripción de la sugerencia y las opciones para anexar un archivo html y una imagen para el detalle de la misma. Al introducir los datos y presionar el botón *guardar* se lleva a cabo todo el procesamiento descrito en las secciones anteriores, y se despliega un mensaje de éxito o fracaso de la transacción (figura 6.6).

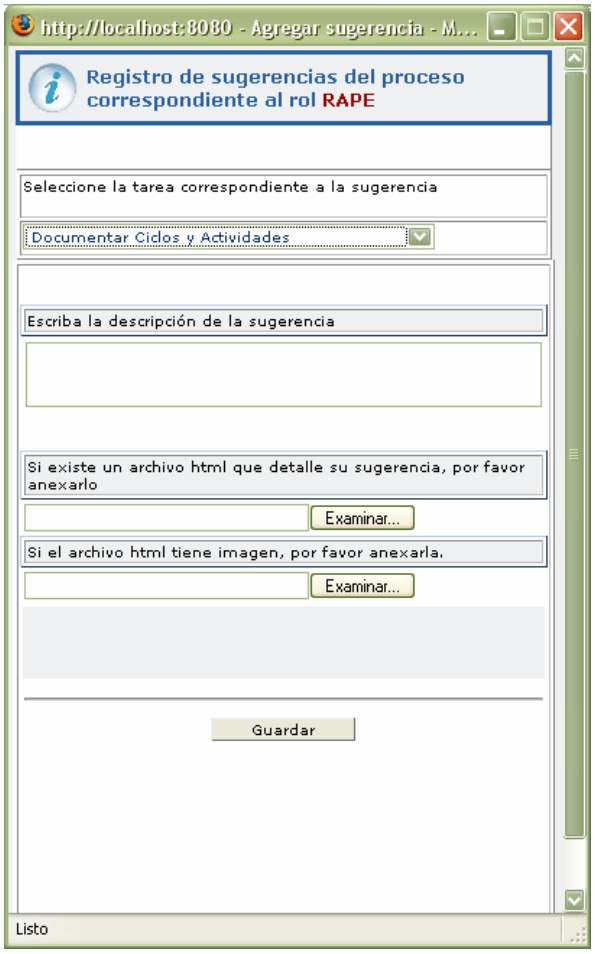


Figura 6.6: Página web generada a la petición del usuario Agregar sugerencia.

6.6 PRUEBAS DEL SISTEMA MULTI-AGENTE

En este apartado nos centraremos en la descripción, aplicación y evaluación de las pruebas para el Sistema Multi-Agente, con el objetivo de contar con un criterio que garantice el correcto funcionamiento de los componentes y determine si el comportamiento del sistema implementado satisface su especificación.

El nivel de cobertura de las pruebas, así como la explicación, su aplicación y resultados se describen a continuación.

6.6.1 Pruebas unitarias

Primero se probaron las clases, analizando cada uno de sus métodos y luego a la clase en general. Para ello, se llevaron a cabo al mismo tiempo que la codificación pruebas unitarias de caja blanca. Dichas pruebas consistieron en localizar y corregir los errores en aspectos como errores de sintaxis, las trayectorias que tienen que seguir los datos, inicio y terminación de los ciclos en forma normal y con datos dentro de los rangos permitidos, verificación de las terminaciones normales y anormales de recursiones y ciclos, manejo de errores, las estructuras de datos y el acceso y conexión a la base de datos.

Posteriormente, se realizaron pruebas de caja negra donde se introdujeron ciertos valores para verificar las salidas. Cabe señalar, que ambas pruebas se realizaron satisfactoriamente y que los resultados de las pruebas de caja negra están descritos en el apéndice D.

6.6.2 Pruebas de integración

Este proceso de pruebas se realizó con la finalidad de averiguar la corrección de las interfaces entre los módulos obtenidos. Para ello, la técnica de prueba elegida fue top-dow (descendente). Se comenzó analizando el funcionamiento del flujo de control del programa principal examinando cómo pasa el control y los datos a los distintos módulos, cómo estos los devuelven y cómo el programa de control pasa los datos al dispositivo de salida (interfaz gráfica del usuario).

Con respecto a esto último, se desarrolló un conjunto de instrucciones en cada módulo que permitiera la prueba puesto que aún no habían sido desarrollados. Estos módulos simulados se les denomina stubs.

Para terminar, el programa principal o de control que se probó en primer lugar fue el servlet auxiliar DirectoryServlet y el agente Coordinador. Posteriormente, se integraron los módulos correspondientes a los demás agentes y la parte del manejo de datos (clases DAO, ValueObjects y conexión con las bases de datos).

6.6.3 Pruebas del sistema

Los diferentes casos de prueba se diseñaron tomando como referencia los casos de uso definidos en el capítulo 4 (sección 4.2.1). La idea fundamental es mostrar la funcionalidad de la herramienta desarrollada. Estos casos de prueba del sistema se realizaron ejecutando la aplicación desde Eclipse y visualizando la herramienta a través de un navegador de Internet, utilizando las funciones que tienen que ver con el Sistema Multi-Agente.

Como precondición indispensable se tiene: haber ejecutado HeAP MoProSoft, se ingresó al sistema con un usuario determinado, se mostró la página principal y se desplegaron los elementos requeridos para las pruebas. Puesto que la aplicación aún no estaba en condiciones como para cumplir con la precondición, se utilizó un stub que permitiera simular dicha acción.

También cabe señalar, que no se realizaron pruebas con usuarios reales dado a la situación anteriormente mencionada en cuanto a que HeAP MoProSoft no cuenta con el nivel de desarrollo deseado.

Por último, los casos de prueba y resultados se muestran en el apéndice D.

Nuestra recompensa se encuentra en el esfuerzo y no en el resultado. Un esfuerzo total es una victoria completa.

MAHATMA GANDHI

Conclusiones

En este apartado se presentan los resultados y contribuciones obtenidos durante el desarrollo del trabajo. Adicionalmente, se exponen las conclusiones generales con respecto a los objetivos planteados, así como los trabajos a futuro identificados.

RESULTADOS

Evaluando los resultados obtenidos en el capítulo anterior con respecto a las pruebas, se puede concluir que el Sistema Multi-Agente cumple de manera satisfactoria con los requerimientos especificados.

Sin embargo, debido a la situación de que HeAP MoProSoft se encuentra en un estado de desarrollo, no se podrá cumplir con el requerimiento de integrar el Sistema Multi-Agente con esta aplicación. Sin duda, cuando se consiga una versión funcional y adecuada de HeAP MoProSoft para ofrecer a la Industria del Software, se podrá agregar el sistema desarrollado como originalmente fue requerido.

Otro aspecto importante en la evaluación de los resultados obtenidos con el desarrollo de este trabajo de investigación, es la satisfacción de los objetivos planteados al principio de la tesis (capítulo 3). Lo anterior deriva en la explicación del nivel de satisfacción alcanzado por cada objetivo que a continuación se ofrece.

- ***Aplicar técnicas de las áreas de Ingeniería de Software e Inteligencia Artificial***. Se puede afirmar que se ha desarrollado una herramienta que contribuye a la demostración práctica de la integración de ciertas técnicas de estas áreas para ofrecer mejores funcionalidades en sistemas de software.

- ***Desarrollar un sistema que al ser incorporado a HeAP MoProSoft permita aumentar la eficiencia y conocimiento del usuario***. En este sentido, se considera que aunque no se logró la incorporación del Sistema Multi-Agente a esta aplicación por el momento, se ofrece una integración que puede realizarse de manera relativamente sencilla (capítulo 6). En cuanto a aumentar la eficiencia y conocimiento del usuario que participa en la implementación de MoProSoft, en particular en la Administración de Proyectos, dentro de una empresa; el funcionamiento de proveer la información que el usuario necesita para realizar las tareas comprendidas en la aplicación HeAP MoProSoft contribuye a maximizar la efectividad de los usuarios participantes y con ello indudablemente se facilita el aprendizaje o adquisición de conocimiento.

- ***Ofrecer el servicio de monitoreo y tutoría de las tareas del usuario***. El sistema no cumple totalmente con la satisfacción de estos servicios debido a las limitaciones que se tienen tanto en HeAP MoProSoft como en el manejo de las interfaces Web con esta tecnología. Con respecto a la última limitante, se debe a que no se ha realizado avances por los desarrolladores de la comunidad JADE en este sentido, por lo que la integración de agentes en estas interfaces no es sencilla.

- ***Analizar los fundamentos teóricos de Agentes/Sistemas Multi-Agente***. Se aprendieron los fundamentos teóricos de Agentes/Sistemas Multi-Agente y a su vez se asentaron los necesarios para los lectores de la tesis mediante el estudio y análisis realizado.

CONTRIBUCIONES

Retomando los resultados obtenidos, se considera que las principales contribuciones del presente proyecto de tesis son las siguientes:

- La construcción del Sistema Multi-Agente integra tecnologías de distintas áreas de conocimiento: técnicas de *Ingeniería de Software* para estructurar el proceso de desarrollo y técnicas de *Inteligencia Artificial* para dotar a los programas con capacidad para tratar ciertas situaciones y tomar decisiones.

- Se generó otro ejemplo de utilización de la metodología INGENIAS para desarrollar Sistemas Multi-Agentes, que envuelve prácticas conocidas de la Ingeniería de Software con aspectos teóricos de la teoría de agentes de la Inteligencia Artificial.

- En base a su diseño, el sistema permite la implementación de comportamientos más complejos en todos los agentes, para poder adaptarlos después a HeAP MoProSoft u otro sistema, todo sin modificar su estructura inicial y funcional.

- Con este sistema se espera que sea más fácil el manejo de HeAP MoProSoft logrando así que el usuario ejecute y aprenda de una manera sencilla las tareas que comprende la administración de proyectos. El usuario puede tener información que en el momento no recuerda o desconoce para la ejecución de cierta tarea, así como su estado actual y las sugerencias o consejos de expertos para su realización. Con ello se brinda al usuario centrarse en tareas propias del proceso, en lugar de tareas de búsqueda de información.

- Finalmente, la herramienta desarrollada complementa los servicios ofrecidos por HeAP MoProSoft para llevar a cabo sus objetivos.

CONCLUSIONES

Con el desarrollo del Sistema Multi-Agente mediante el uso de la metodología INGENIAS se muestra la ventaja que representa seguir un método para realizar las actividades de construcción de un sistema de software de esta naturaleza.

Se documentó el procedimiento de desarrollo del sistema lo cual permite ofrecer cierta guía al construir este tipo de Sistemas Multi-agente y ser una referencia para trabajos futuros.

Es de destacar la aplicación y utilidad que brinda el sistema al integrarse dentro del sistema HeAP MoProSoft puesto que se contribuyó a mejorar un aspecto de este sistema. Además, se ha dejado una estructura adecuada para añadir nuevas capacidades incrementando la funcionalidad de la herramienta.

Una ventaja al utilizar el enfoque de Sistema Multi-Agente es la escalabilidad puesto que permite una mayor simplicidad y flexibilidad en la incorporación de nuevas funcionalidades que el enfoque tradicional. Esta característica es importante porque al implementar una nueva funcionalidad puede ser incorporada como una nueva interacción o un nuevo agente simplemente respetando la arquitectura definida.

Tomando en cuenta lo anterior y todo el trabajo de la tesis, se considera que se han alcanzado la mayoría de los objetivos del proyecto gracias a la convergencia de las áreas de Ingeniería de Software e Inteligencia Artificial y se dejan abiertas opciones para trabajos futuros.

TRABAJOS A FUTURO

Existen las cuestiones que pueden ser materia de trabajos futuros debido a que el proyecto de tesis desarrollado involucra varias áreas de la computación y a que hubo limitaciones en cuanto a funcionalidades no explotadas por no contar con la herramienta HeAP MoProSoft terminada. Los trabajos a futuro identificados son:

1. Agregar más funcionalidades a los agentes para proporcionar información de control en base a lo realizado por el usuario en HeAP MoProSoft y explotar la información almacenada. Por ejemplo: monitorear, recopilar e informar sobre avances, riesgos, costos, dependencias entre tareas, fechas de adquisiciones y capacitación, información histórica de proyectos, entre otros.

2. Para aumentar el desempeño de toda la herramienta, dotar a los agentes que componen al Sistema Multi-Agente con características de agentes inteligentes como aprender de experiencias pasadas para mejorar sus acciones, extraer por si mismos las conclusiones adecuadas de la información procesada, entre otras.

3. Mejorar la interfaz del Sistema Multi-Agente con la finalidad de que sea más usable y adecuado para el usuario mediante la integración de conocimiento del área de Interfaces Humano-Computadora.

BIBLIOGRAFÍA

[ALUR2003] Alur, Deepak; Crupi, John; MalksAlur, Dan. Core J2EE Patterns: Best Practices and Design Strategies. Págs. 462-495. Prentice Hall / Sun Microsystems Press, 2003.

[BELLIFEMINE2001] Bellifemine, F; Poggi, A; Rimassa, G. *Developing Multi-Agent Systems with JADE*. Págs 89-103. LNAI 1571, 2001.

[BROOKS1991] Brooks, Rodney A. *Intelligence without representation*. Artificial Intelligence, Págs. 47:139–159, 1991.

[CAIRE2001] Giovanni Caire, Francisco Leal, Paulo Chainho, Richard Evans, Francisco Garijo, Jorge, Gomez, Juan Pavón, Paul Kearney, Jamie Stark and Philippe Massonet. *Agent oriented analysis using MESSAGE/UML.* Proc. of 2nd International Workshop on Agent Oriented Software Engineering, Montreal Canada, 2001.

[CAVANESS2004] Cavaness, Chuck. *Programming Jakarta Struts*. 2nd Edition. Págs. 8-20. O'Reilly Media, 2004.

[CRUZ2007] Cruz Salas, Rafael. Tesis de maestría: *Sistema de tableros de control para gestión de proyectos para MoProSoft*. Universidad Nacional Autónoma de México, 2007.

[DELOACH2001] DeLoach, S. *Analysis and Design using MaSE and agentTool*. Actas de conferencia. Proc. 12th Midwest Artificial Intelligence and Cognitive Science Conferece (MAICS). 2001.

[DURFEE1994] Durfee, Edmund; Rosenschein, J. *Distributed Problem Solving and Multi-Agent Systems: Comparisons and Examples*. Págs 94-104. Proc. 13th Int. DAI Workshop, 1994.

[FERBER1999] Ferber, Jacques. *Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence*. Págs. 125-160. Addison-Wesley. Great Britain, 1999.

[FIPA2003] FIPA Abstract Architecture Specification (1996-2002) y FIPA Agent Discovery Service Specification. Version 1.2e (1996-2003). Disponible en: http://www.fipa.org/

[GERVAIS2003] Gervais, Marie-Pierre. *ODAC: An Agent-Oriented Methodology Based on ODP*. Págs. 199-228. Journal of Autonomous Agents and Multi-Agent Systems 7(3), 2003.

[GOB2005] Declaratoria completa en la página 40 del Diario Oficial de la Federación. Disponible en:

http://www.gobernacion.gob.mx/dof/2005/agosto/dof15-08-005.pdf

[GONZÁLEZ2006] González, Claudia N. *MoProSoft 16 veces: Lecciones Aprendidas*. Conferencia en: SG '06 Conferencia y Expo (2006: Ciudad de México). Consultado en febrero del 2007. Disponible en: http://www.softwareguru.com.mx/downloads/sg06/presentaciones/Mejora de procesos/P5 - MoProSoft 16 veces.pdf

[GRASIA2005] Sección Web de GRASIA para la metodología INGENIAS. Disponible en: http://grasia.fdi.ucm.es/ingenias/

[HAYES1995] Hayes-Roth, Barbara. *An Architecture for Adaptive Intelligent Systems.* Págs. 72, 329-365. *Artificial Intelligence: Special Issue on Agents and Interactivity*, 1995.

[HUNTER2007] Hunter, Jason. *Clase com.oreilly.servlet.MultipartRequest.* Versión 1.7. Consultado en mayo del 2007. Disponible en: http://www.stanford.edu/group/coursework/docsTech/oreilly/com.oreilly.servlet.MultipartRequest.html

[IGLESIAS1998] Iglesias, C. Tesis Doctoral: *Definición de una Metodología para el Desarrollo de Sistemas Multi-Agente*. Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid. España, 1998.

[INGENIAS2005] Ingenias Development Kit (IDK). Disponible en: http://ingenias.sourceforge.net/

[JACOBSON2000] Jacobson, Ivar; Booch, Grady; Rumbaugh, James. *El Proceso Unificado de Desarrollo de Software*. Pearson Education, 2000.

[JADE2005] Plataforma JADE. Disponible en: http://jade.tilab.com/

[JADEADM2005] Bellifemine, Fabio. Caire, Giovanni. Trucco, Tiziana. Rimassa, Giovanni. *Jade Administrator's guide*. TILAB. Enero del 2005. Disponible en: http://jade.tilab.com/doc/administratorsguide.pdf

[JADEPG2006] Bellifemine, Fabio. Caire, Giovanni. Trucco, Tiziana. Rimassa, Giovanni. *Jade Programmer's guide*. TILAB, 2006. Disponible en: http://jade.tilab.com/doc/programmersguide.pdf

[JENNINGS1998] Jennings, Nicholás R; Sycara, Katia; Wooldridge, Michael. *A roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems*. Págs. 1, 7-38. Kluwer Academia Publishers. Boston, 1998.

[JENNINGS2000] Jennings, Nicholas R. Wooldridge, Michael. *Agent-Oriented Software Engineering.* Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi- Agent World: Multi-Agent System Engineering, 2000.

[JULIAN2001] Julián, V; Rebollo, M; Carrascosa, C. *Agentes de Información*. Págs. 56-62. Revista Base. ISSN 1135-0695, 2001.

[KINNY1996] Kinny, D; Georgeff, M.; Rao, A. *A Methodology and Modelling Technique for Systems of BDI Agents*. Págs. 56–71. Proc. 7th European Workshop on Modelling Autonomous Agents in a MultiAgent World, (LNAI Volume 1038). Springer-Verlag: Berlin, Alemania, 1996.

[KLUSCH2001] Klusch, M. *Information agent technology for the Internet: a survey*. Págs. 337-372. Data and Knowledge Engineering, Volumen. 36 (3). 2001.

[LEONTIEV1978] Leontev, Aleksei N. *Activity, Conciousness, and Personality*. Prentice Hall. 1978.

[LIND2000] Lind, Jurgen. *Iterative Software Engineering for Multiagent Systems : The MASSIVE Method.* Proc. 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, 2000.

[LOPEZ2005] López Jaquero, Víctor Manuel. Tesis de doctorado: *Interfaces de Usuario Adaptativas Basadas en Modelos Y Agentes Software*. Universidad de Castilla-La Mancha, Departamento de sistemas informáticos, 2005.

[LYYTINEN1999] Lyytinen, K. S*. METAEDIT+ A fully configurable Multi-User and Multi-tool CASE and CAME Environment*. Actas de conferencia. Springer Verlag, 1999.

[MAES1995] Maes, Pattie. *Artificial Life Meets Entertainment: Life like Autonomous Agents*. Págs. 38, 11, 108-114. Communications of the ACM, 1995.

[MAS2005] Mas, Ana. *Agentes Software y Sistemas Multiagente*. *Conceptos, Arquitecturas y Aplicaciones*. Págs 145-252. Pearson Education, S.A. Madrid, 2005.

[NWANA1996] Nwana, Hyacinth. S. *Software Agents: An overview. Knowledge Engineering Review*. Págs. 1-40. Cambridge University Press, 1996.

[OKTABA2005] Oktaba, Hanna. *Introducción a la Ingeniería de Software.* Presentación *en:* Maestría en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México (2005: Ciudad de México). Consultado en febrero del 2007.

[PAVÓN2003] Pavón, J; Gómez-Sanz, J. J. *Agent Oriented Software Engineering with INGENIAS*. Págs. 394-403. In Multi-Agent Systems and Applications III, 3rd International Central and Eastern European

Conference on Multi-Agent Systems, CEEMAS 2003. Lecture Notes in Computer Science 2691. Springer-Verlag.

[PMBOK2004] Project Management Institute. *Guía de los Fundamentos de la Dirección de Proyectos (Guía del PMBoK)*. 3ra. Edición, Newtown Square, Pennsylvania, USA, 2004.

[PRUEBAS2005] *Pruebas Controladas*. Presentación en: MoProSoft, Alto desempeño a bajo costo para empresas Mexicanas de software (2005: Ciudad de México). Consultado en julio del 2007.

[RICORDEL2001] Ricordel, P. M. Tesis doctoral: *Programmation Orientée Multi-Agents, Développement et Déploiement de Systèmes Multi-Agents Voyelles*. Institut National Polytechnique de Grenoble. 2001.

[RUSSELL1995] Russell, S; Norvig, P. *Artificial Intelligence, A modern Approach*. Págs. 31-35. Prentice Hall International, 1995.

[SE2001] Secretaría de Economía. *Programa para el desarrollo de la industria de software*. Secretaría de Economía, 2001. Disponible en: http://www.software.net.mx/desarrolladores/prosoft/Programa/prosoft.htm

[STAN1996] Stan, Franklin; Art, Graesser. *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents.* Págs. 1-9. Institute for Intelligent Systems, University of Memphis. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

[SYCARA2003] Sycara, K; Paolucci, M; Vav Velsen, M; Giampapa, J. *The RETSINA MAS Infrastructure*. Págs. 29-48. Autonomous Agents and Multi-Agent Systems, 2003.

[TORRES2007] Torres Castillo, Brenda Daniela. Tesis de maestría*: Herramienta de Administración de Proyectos para MoProSoft*. Universidad Nacional Autónoma de México, 2007.

[WOOLDRIDGE1995] Wooldridge, Michael; Jennings, Nicholas R. *Intelligent Agents: Theory and Practice*. Págs. 23-40. In Knowledge Engineering Review 10(2), 1995.

[WOOLDRIDGE2000] Wooldridge, M; Jennings, N. R; Kinny, D. *The Gaia Methodology for Agent-Oriented Analysis and Design*. Journal of Autonomous Agents and Multi-Agent Systems, vol.15. 2000.

[WOOLDRIDGE2001] Wooldridge, M; *Ciancarini,* P. *Agent-Oriented Software Engineering: The State of the Art*. Págs. 75:116–118. Handbook of Software Engineering and Knowledge Engineering. World Scientific Publishing Co., 2001.

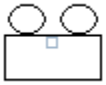
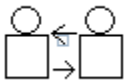
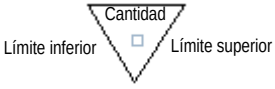
Apéndice A

Notación INGENIAS

Esta sección describe la notación específica de la metodología INGENIAS para la representación de los modelos.

ENTIDADES BÁSICAS

La metodología INGENIAS proporciona una jerarquía de conceptos básicos para el desarrollo de un Sistema Multi-Agente así como una notación para representar estos conceptos. A continuación se presenta en la tabla A.1 la notación de estos conceptos.

 **Objetivo**	Se etiqueta con el nombre del objetivo.
 **Tarea**	Se etiqueta con el nombre de la tarea.
 **Rol**	Se etiqueta con el nombre del rol.
 **Agente**	Se etiqueta con el nombre del agente.
 **Grupo**	Se etiqueta con el nombre del grupo.
 **Organización**	Se etiqueta con el nombre de la organización.
 **Interacción**	Se etiqueta con el nombre de la interacción y su naturaleza, como coordinación, planificación o negociación.
 **Flujo de trabajo**	Se etiqueta con el nombre del flujo.
 **Consulta de entidades autónomas**	Se etiqueta con nombres concretos de agentes existentes o expresiones que denotan agentes existentes.
 **Recurso**	Se etiqueta con el nombre del recurso, la cantidad disponible del mismo, el límite inferior y superior admisibles. Por debajo o encima de estos límites, el recurso se deshabilita.

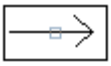
 **Aplicación de entorno**	Se etiqueta con el nombre de la aplicación y las operaciones soportadas.
 **Aplicación interna**	Se etiqueta con el nombre de la aplicación y las operaciones soportadas.
 **Hecho**	Se etiqueta con el nombre del hecho y los nombres de los slots identificados.
 **Evento**	Se etiqueta con el nombre del evento y los nombres de los slots identificados.
 **Unidad de interacción**	Se etiqueta con el nombre de la unidad y el acto del habla al que hace referencia, como *request*, *inform*, o *not-understood*.
 **Creencia**	Se etiqueta con el nombre de la evidencia e información acerca de qué es lo que se está aceptando como cierto.
 **Procesador de estado mental**	Se etiqueta con el nombre del procesador.
 **Gestor de estado mental**	Se etiqueta con el nombre del gestor.

Tabla A.1: Notación de conceptos de INGENIAS.

RELACIONES BÁSICAS

La nomenclatura de las relaciones obedece a unas reglas nemotécnicas donde el nombre de la relación es precedido por un conjunto de letras que denote su procedencia, como el flujo de trabajo (*WF*), meta-modelo de agente (A), interacción (I), unidad de interacción (UI), modelos de tareas y objetivos (GT), relaciones sociales (AGO), organización (O) o el entorno (E). A continuación, se describen las meta-relaciones básicas.

GTAffects	Indica que se actúa sobre una entidad mental, por lo que se requiere de su existencia.
GTCreates	Indica la alteración del estado mental al crear una o varias entidades.
GTDescomposes	Representa la descomposición de objetivos y tareas. Da lugar a grafos acíclicos donde se distinguen tareas, objetivos, sub-tareas y sub-objetivos. La descomposición de objetivos da lugar a árboles Y/O soportados por instancias de

	GTDependeY (asume que cuando todos los sub-objetivos han sido satisfechos, el objetivo padre puede darse por alcanzado) y GTDependeO (requiere que al menos uno de los objetivos referenciados haya sido satisfecho).
GTDestroys	Indica la alteración del estado mental al destruir una o varias entidades.
GTFails	Es una especialización que concreta la forma de modificar un objetivo al hacerlo fallar con la existencia de ciertas evidencias que lo indiquen (propiedad *Evidencia*).
GTModifies	Indica la alteración del estado mental al modificar una o varias entidades.
GTPursues	Meta-relación entre objetivos y agente u organización para representar que estas dos últimas entidades persiguen ciertos objetivos.
GTSatisfies	Es una especialización que concreta la forma de modificar un objetivo al hacerlo satisfacer con la existencia de ciertas evidencias que lo indiquen (propiedad *Evidencia*).
WFConnects	Corresponde con la conexión de las salidas de una tarea con las entradas de otra.
WFConsumes	Define un tipo de precondición que indica que para que se ejecute la tarea se necesita que existan determinadas entidades mentales (*objetivos*, *creencias*, *hechos,* etc.).
WFDecomposes	Representa la descomposición de objetivos y tareas.
WFPlays	Meta-relación entre roles y agentes que indica al diseñador que el agente adquiere todos los objetivos asociados al rol, todas sus responsabilidades y sus capacidades.
WFProduces	Define un tipo de postcondición que indica la creación de entidades mentales, interacciones o reposición de recursos en el seno de un flujo de trabajo.
WFResponsable	Indica el agente o agentes responsables de las tareas.
WFSpecifiesExecution	Indica que la especificación concreta de las condiciones de ejecución de una tarea se presenta dentro de una interacción.
WFUses	Define un tipo de precondición que indica que se necesitan algunos de los recursos que se identifican en el meta-modelo de entorno.
IHasSpec	Indica la ejecución y representación de las *unidades de interacción*. Existen dos tipos de especificación: Diagramas de colaboración UML y lo especializados GRASIA.
UIColaborates	Representa la colaboración de una unidad de interacción.
UIInitiates	Representa el inicio de una unidad de interacción.
UIPrecedes	Representa el orden de ejecución secuencial de las unidades de interacción.
OHasGroup, OHasMember	Cada grupo contiene agentes, recursos, aplicaciones o roles. La asignación obedece a propósitos organizativos.
AGOInconditionalSubordinationRelationshipGroup	Consiste en obedecer todas y cada una de las órdenes.

Tabla A.2: Nomenclatura de las relaciones básicas de INGENIAS.

Apéndice B

Modelos del Análisis y Diseño del Sistema Multi-Agente

Esta sección contiene los modelos complementarios de las fases de análisis y diseño del Sistema Multi-agente desarrolladas en los capítulos cuarto y quinto respectivamente.

1. ANÁLISIS

1.1 Descripción de tareas

1.1.1 *Tarea: buscar_dato_proceso*

Esta tarea toma como entrada los datos obtenidos del agente *Coordinador* (hecho *infoProceso*). Para atender la petición hace uso de la Base de Datos de HeAP MoProSoft. Finalmente, produce el resultado obtenido de la búsqueda de la información solicitada (hecho *resultadoProceso*).

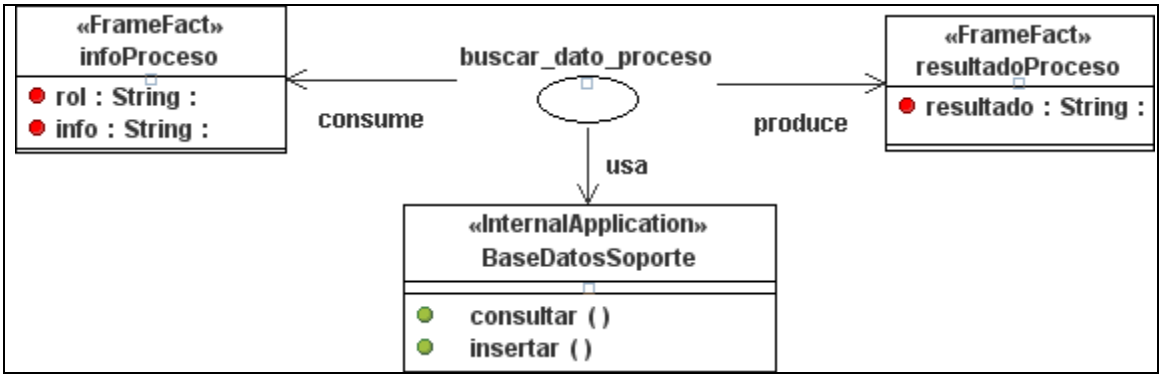


Figura B.1: Descripción de la tarea *buscar_dato_proceso.*

1.1.2 *Tarea: solicitar_estado_tareas*

Esta tarea toma como entrada los datos obtenidos del usuario a través de la herramienta HeAP MoProSoft (hecho *DatosUsuario*). Para atender la solicitud hace uso del elemento ServletAgente. El procesamiento de la solicitud del estado de las tareas se realiza dentro de la interacción *solicitar_estado_tareas*.

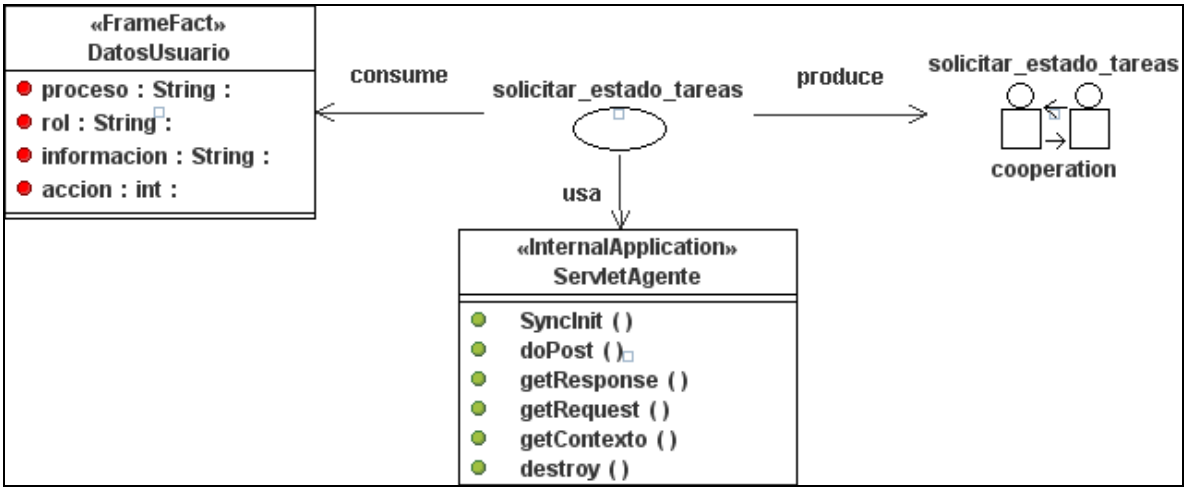


Figura B.2: Descripción de la tarea *solicitar_estado_tareas.*

1.1.3 *Tarea: buscar_info_tareas*

Esta tarea toma como entrada los datos obtenidos del agente *Coordinador* (hecho *infoTareas*). Para atender la petición hace uso de la Base de Datos de HeAP MoProSoft. Finalmente, produce el resultado obtenido de la búsqueda de la información solicitada (hecho *resultadoTareas*).

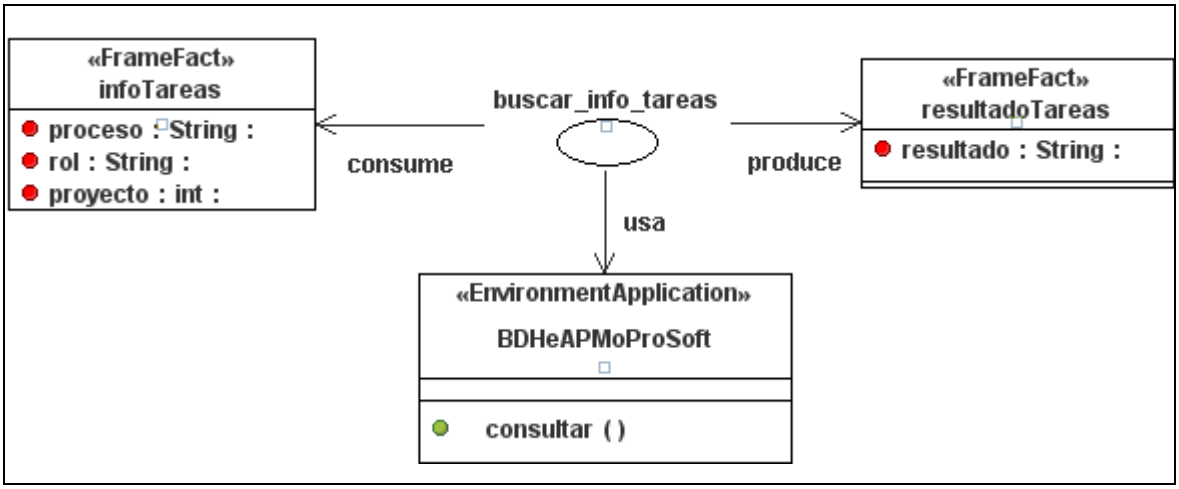


Figura B.3: Descripción de la tarea *buscar_info_tareas.*

1.1.4 *Tarea: solicitar_sugerencias*

Esta tarea toma como entrada los datos obtenidos del usuario a través de la herramienta HeAP MoProSoft (hecho *DatosUsuario*). Para atender la solicitud hace uso del elemento ServletAgente. El procesamiento de la solicitud de sugerencias de una tarea se realiza dentro de la interacción *solicitar_sugerencias*.

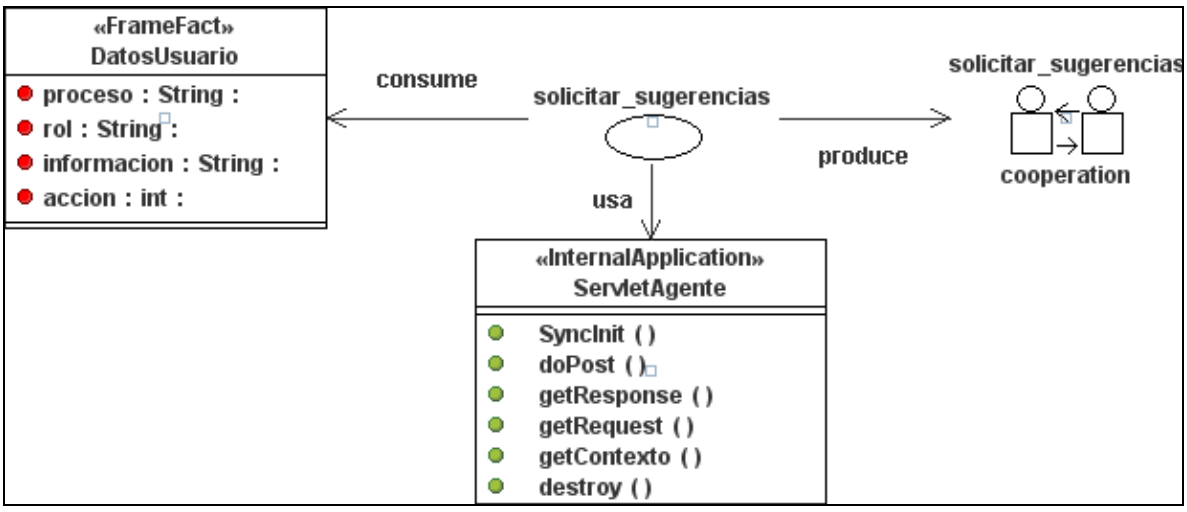


Figura B.4: Descripción de la tarea *solicitar_sugerencias.*

1.1.5 *Tarea: buscar_info_sugerencias*

Esta tarea toma como entrada los datos obtenidos del agente *Coordinador* (hecho *infoSugerencia*). Para atender la petición hace uso de la Base de Datos Soporte. Finalmente, produce el resultado obtenido de la búsqueda de la información solicitada (hecho *resultadoSugerencia*).

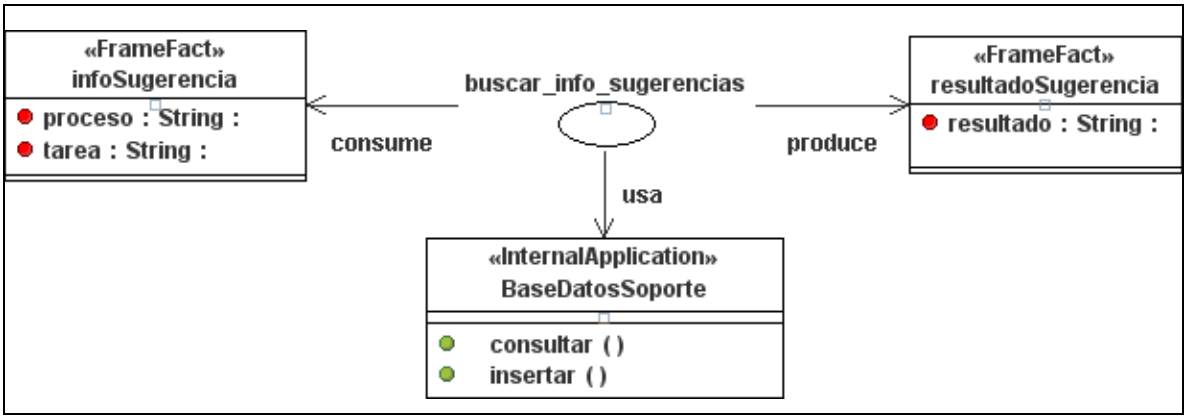


Figura B.5: Descripción de la tarea *buscar_info_sugerencias.*

1.1.6 *Tarea: solicitar_incorporacion_sugerencia*

Esta tarea toma como entrada los datos obtenidos del usuario a través de la herramienta HeAP MoProSoft (hecho *DatosUsuario*). Para atender la solicitud hace uso del elemento ServletAgente. El procesamiento de la solicitud de incorporar una nueva sugerencia de una tarea se realiza dentro de la interacción *anexar_sugerencias*.

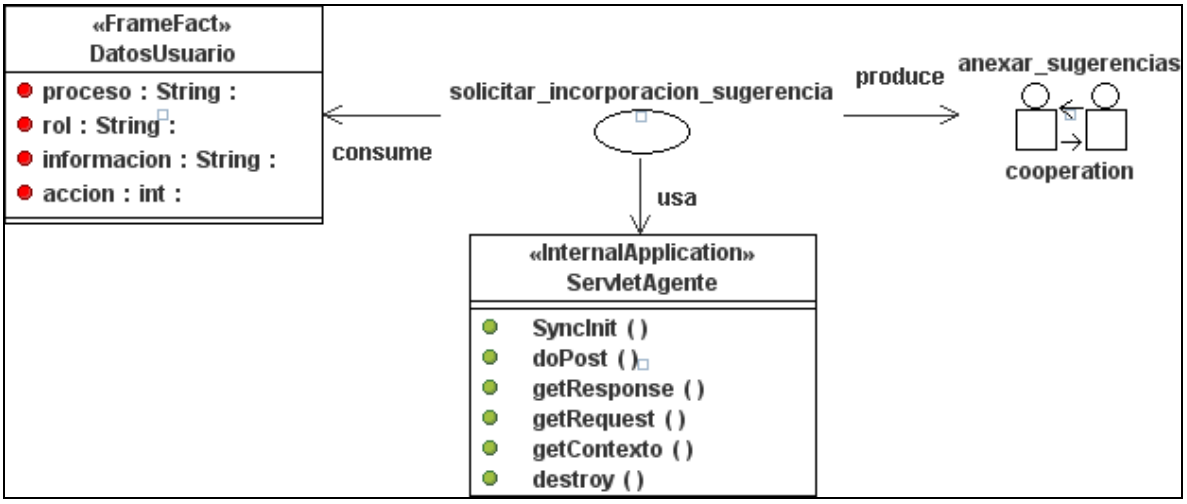


Figura B.6: Descripción de la tarea *solicitar_incorporacion_sugerencia.*

1.1.7 *Tarea: guardar_sugerencia*

Esta tarea toma como entrada los datos obtenidos del agente *Coordinador* (hecho *guardarSugerencia*). Para atender la petición hace uso de la Base de Datos Soporte. Finalmente, produce el resultado obtenido al realizar la transacción solicitada (hecho *resultadoGuardarSugerencia*).

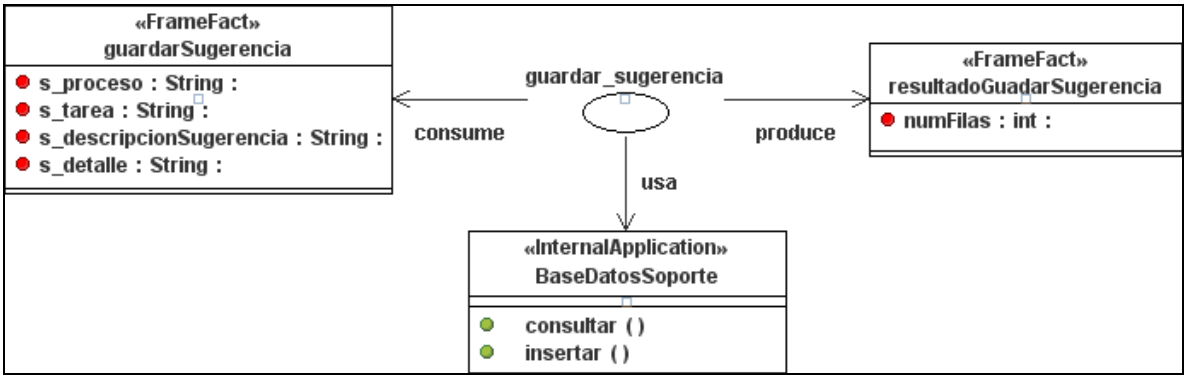


Figura B.7: Descripción de la tarea *guardar_sugerencia.*

1.1.8 *Tarea: visualizar_resultado*

Esta tarea toma como entrada los datos obtenidos del agente que proceso la petición del usuario (hecho *datosRespuesta*). Produce mediante el elemento *ServletAgente* la respuesta http. La respuesta http visualiza al usuario el resultado obtenido.

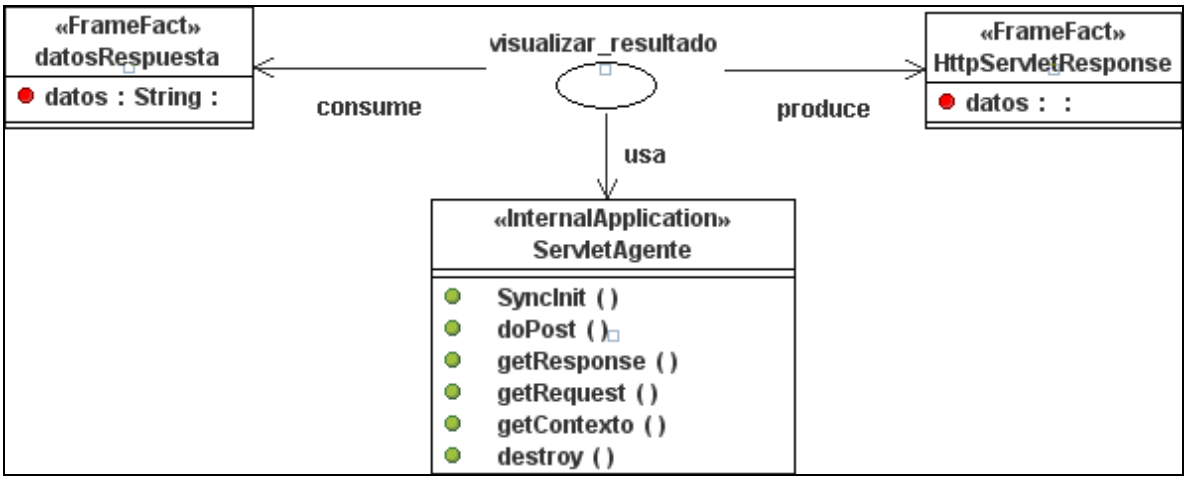


Figura B.8: Descripción de la tarea *solicitar_incorporacion_sugerencia.*

1.2 Descripción de interacciones

1.2.1 *Interacción: anexar_sugerencias*

Solicitar sugerencias (Figura B.9) consiste en ofrecer al usuario un listado de las sugerencias pertenecientes a la tarea solicitada. Las sugerencias que puede consultar sólo son aquellas que pertenezcan a las tareas que realiza el usuario dentro de la herramienta HeAP MoProSoft.

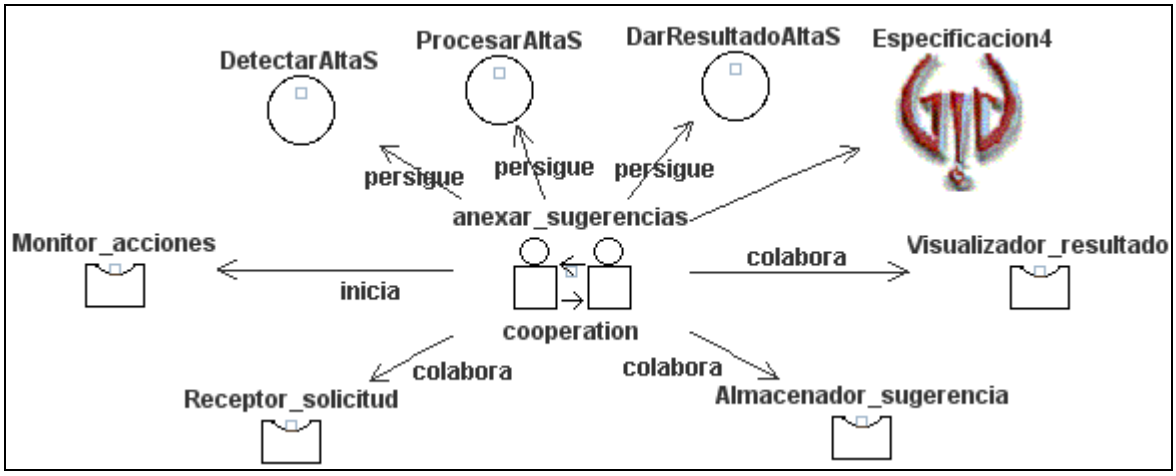


Figura B.9: Meta-modelo de la interacción *anexar_sugerencias*.

A continuación se muestra en la figura B.10 el diagrama de colaboración para esta interacción.

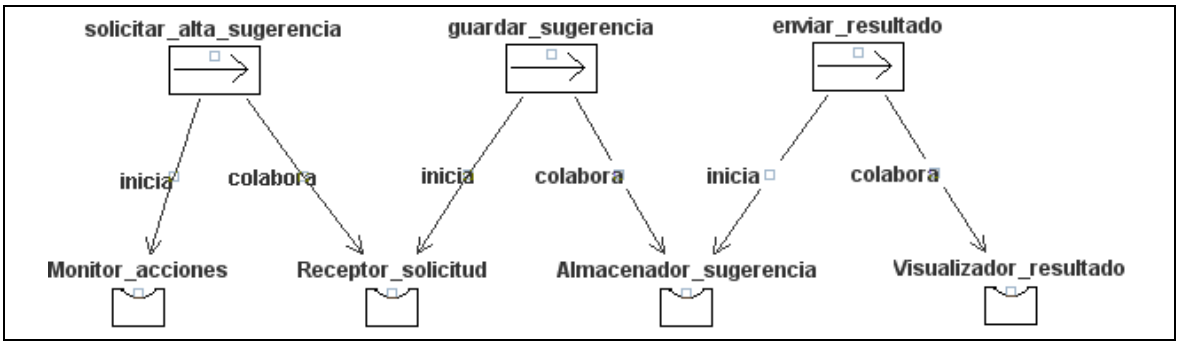


Figura B.10: Diagrama de colaboración para la interacción *anexar_sugerencias*.

1.2.2 *Interacción: solicitar_estado_tareas*

Solicitar el estado de las tareas (Figura B.11) consiste en ofrecer al usuario un listado de las tareas con su estado actual. Estas tareas sólo comprenden a las que realiza el usuario dentro de la herramienta HeAP MoProSoft.

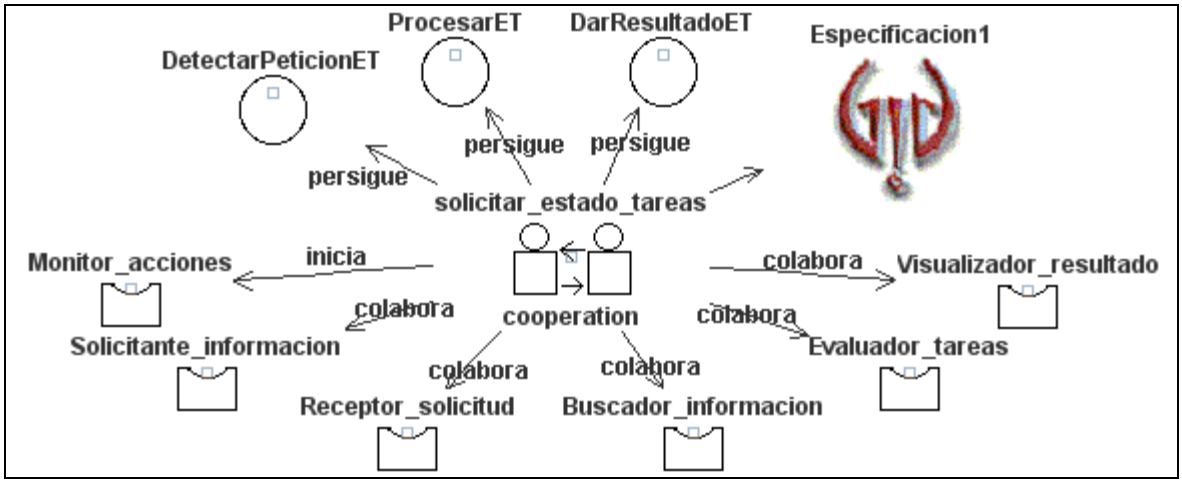


Figura B.11: Meta-modelo de la interacción *solicitar_estado_tareas*.

A continuación se muestra en la figura B.12 el diagrama de colaboración para esta interacción.

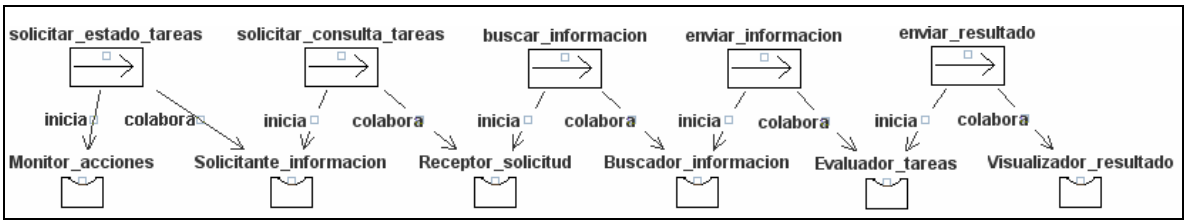


Figura B.12: Diagrama de colaboración para la interacción *solicitar_estado_tareas*.

1.2.3 *Interacción: solicitar_sugerencias*

Solicitar sugerencias (Figura B.13) consiste en ofrecer al usuario un listado de las sugerencias pertenecientes a la tarea solicitada. Las sugerencias que puede consultar sólo son aquellas que pertenezcan a las tareas que realiza el usuario dentro de la herramienta HeAP MoProSoft.

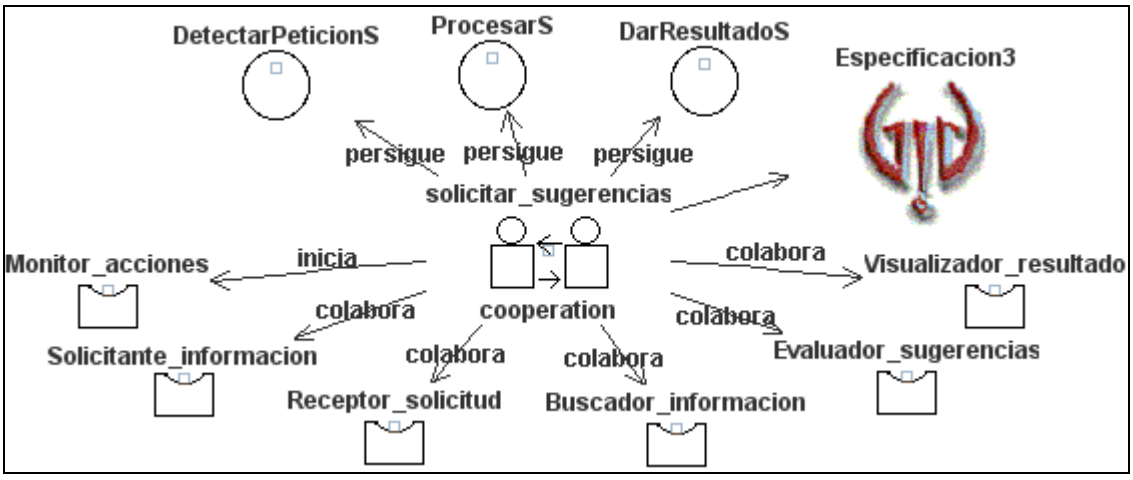


Figura B.13: Meta-modelo de la interacción *solicitar_sugerencias*.

A continuación se muestra en la figura B.14 el diagrama de colaboración para esta interacción.

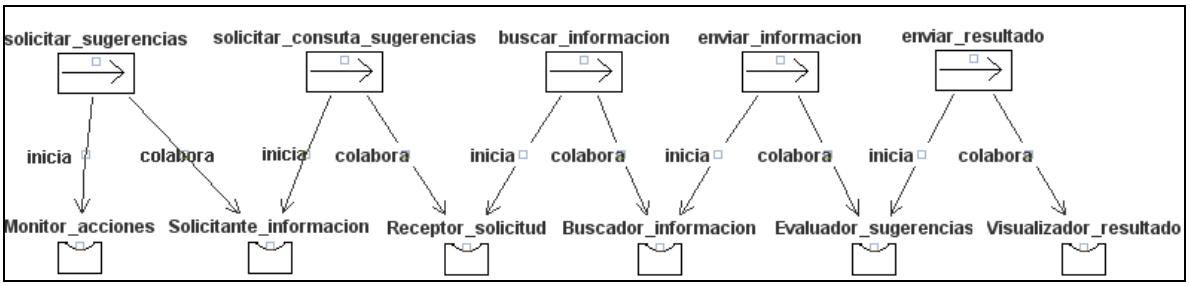


Figura B.14: Diagrama de colaboración para la interacción *solicitar_sugerencias*.

2. DISEÑO

2.1 Condiciones mentales para la satisfacción de objetivos

2.1.1 *Satisface_detectarPeticionDP1*: el objetivo *DetectarPeticionDP* se satisface cuando se produce el hecho *datosCaso2*. La satisfacción se indica con el slot *accion* en 2.

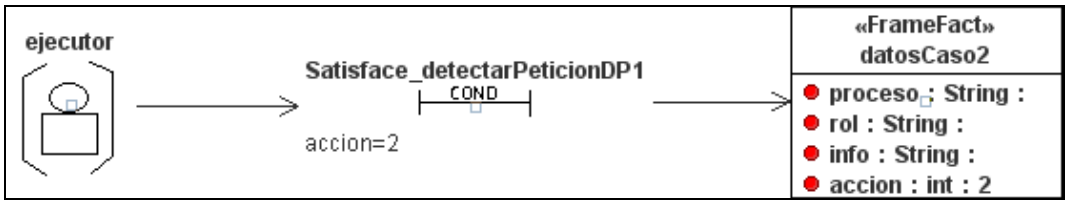


Figura B.15: Condiciones para satisfacer el objetivo *DetectarPeticionDP*.

2.1.2 *Fracaso_detectarPeticionDP1*: el objetivo *DetectarPeticionDP* falla cuando no es posible obtener un valor para el slot *accion*. El comportamiento del agente es bloqueado hasta que lleguen más mensajes.

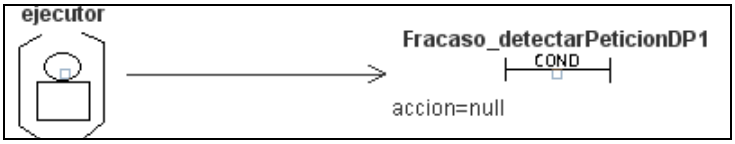


Figura B.16: Condiciones para dar por fallido el objetivo *DetectarPeticionDP*.

2.1.3 *Satisface_detectarPeticionDP2*: el objetivo *DetectarPeticionDP* se satisface cuando se produce el hecho *infoProceso*. La satisfacción se indica con la *accion* en 2.

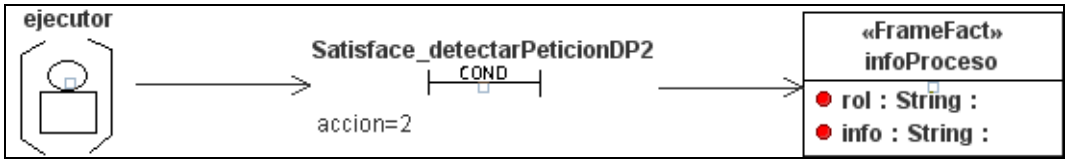


Figura B.17: Condiciones para satisfacer el objetivo *DetectarPeticionDP*.

2.1.4 ***Fracaso_detectarPeticionDP2*:** el objetivo *DetectarPeticionDP* falla cuando no es posible obtener un valor para *accion*. El comportamiento del agente es bloqueado hasta que lleguen más mensajes.

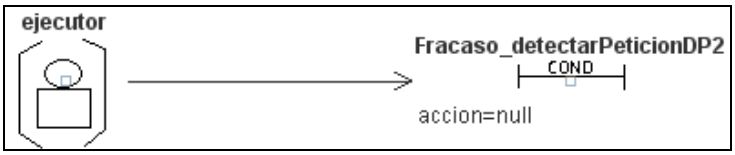


Figura B.18: Condiciones para dar por fallido el objetivo *DetectarPeticionDP*.

2.1.5 ***Satisface_procesarDP1***: el objetivo *ProcesarDP* se satisface cuando se produce el hecho *resultadoProceso*. La satisfacción se indica con el slot *resultado* cuyo valor sea diferente a una cadena vacía.

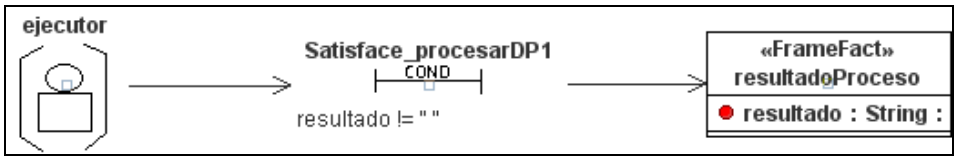


Figura B.19: Condiciones para satisfacer el objetivo *ProcesarDP*.

2.1.6 ***Fracaso_procesarDP1*:** el objetivo *ProcesarDP* falla cuando el valor obtenido en el slot *resultado* es una cadena vacía.

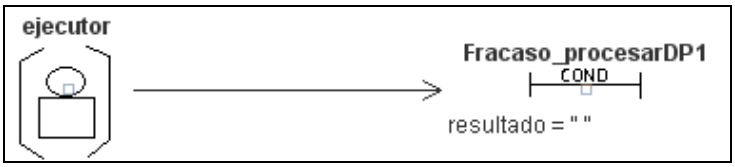


Figura B.20: Condiciones para dar por fallido el objetivo *ProcesarDP*.

2.1.7 ***Satisface_procesarDP2***: el objetivo *ProcesarDP* se satisface cuando se produce el hecho *datosRespuesta*. La satisfacción se indica con el slot *datos* cuyo valor es diferente a null.

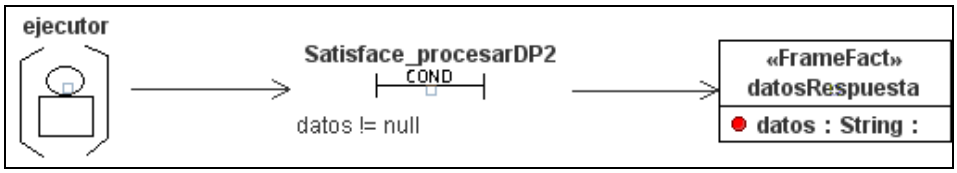


Figura B.21: Condiciones para satisfacer el objetivo *ProcesarDP*.

2.1.8 ***Fracaso_procesarDP2*:** el objetivo *ProcesarDP* falla cuando el valor obtenido en el slot *datos* es null.

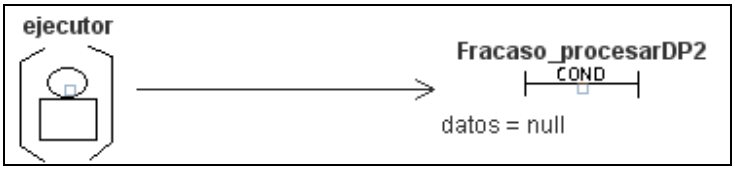


Figura B.22: Condiciones para dar por fallido el objetivo *ProcesarDP*.

2.1.9 ***Satisface_darResultadoDP***: el objetivo *DarResultadoDP* se satisface cuando se produce el hecho *HttpServletResponse*. La satisfacción se indica con el slot *datos* cuya longitud sea mayor a 0.

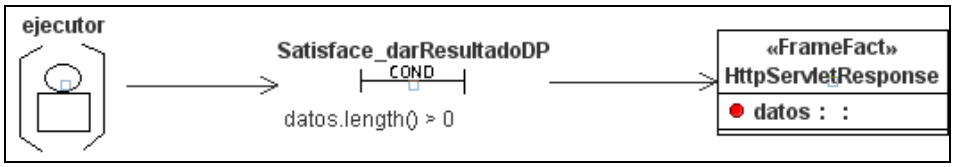


Figura B.23: Condiciones para satisfacer el objetivo *DarResultadoDP.*

2.1.10 ***Fracaso_darResultadoDP*:** el objetivo *DarResultadoDP* falla cuando el valor obtenido en el slot *datos* tiene una longitud de 0.

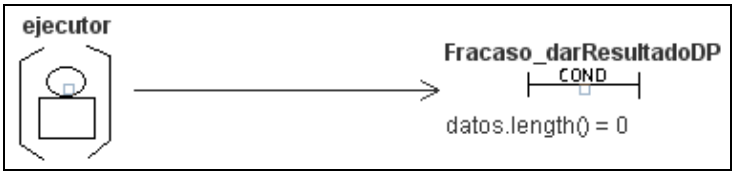


Figura B.24: Condiciones para dar por fallido el objetivo *DarResultadoDP*.

2.1.11 ***Satisface_detectarPeticionET1*:** el objetivo *DetectarPeticionET* se satisface cuando se produce el hecho *datosCaso1*. La satisfacción se indica con el slot *accion* en 1.

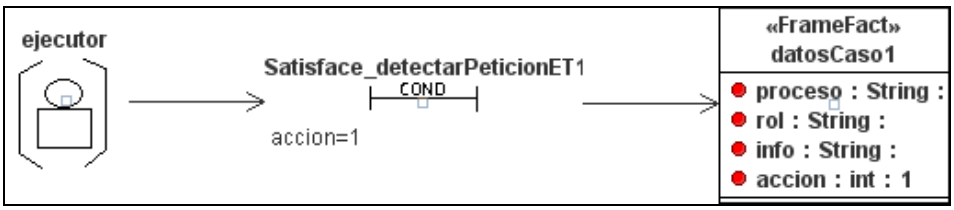


Figura B.25: Condiciones para satisfacer el objetivo *DetectarPeticionET*.

2.1.12 ***Fracaso_detectarPeticionET1*:** el objetivo *DetectarPeticionET* falla cuando no es posible obtener un valor para el slot *accion*. El comportamiento del agente es bloqueado hasta que lleguen más mensajes.

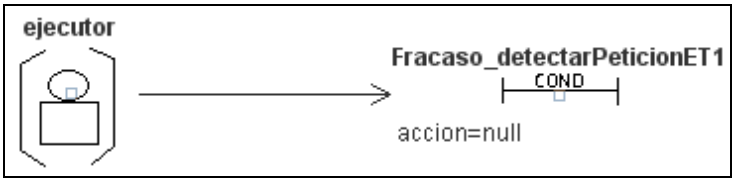


Figura B.26: Condiciones para dar por fallido el objetivo *DetectarPeticionET*.

2.1.13 ***Satisface_detectarPeticionET2***: el objetivo *DetectarPeticionET* se satisface cuando se produce el hecho *datosBuscador*. La satisfacción se indica con la comprobación del que el mensaje recibido del agente *Coordinador* no es null.

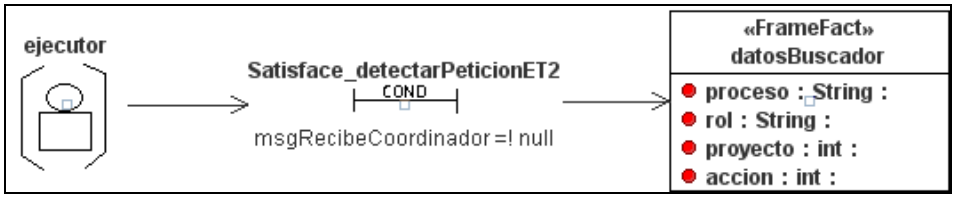


Figura B.27: Condiciones para satisfacer el objetivo *DetectarPeticionET*.

2.1.14 *Fracaso_detectarPeticionET2*: el objetivo *DetectarPeticionET* falla cuando el mensaje recibido del agente *Coordinador* es null. El comportamiento del agente es bloqueado hasta que lleguen más mensajes.

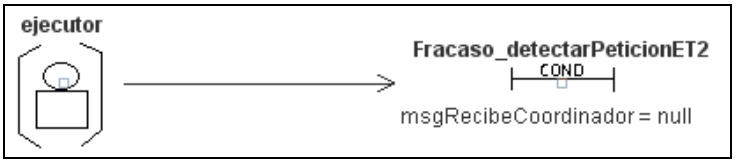


Figura B.28: Condiciones para dar por fallido el objetivo *DetectarPeticionET*.

2.1.15 *Satisface_detectarPeticionET3*: el objetivo *DetectarPeticionET* se satisface cuando se produce el hecho *infoTareas*. La satisfacción se indica con *accion* en 1.

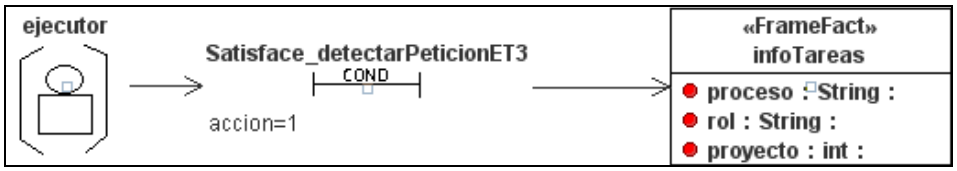


Figura B.29: Condiciones para satisfacer el objetivo *DetectarPeticionET*.

2.1.16 *Fracaso_detectarPeticionET3*: el objetivo *DetectarPeticionET* falla cuando no es posible obtener un valor para *accion*. El comportamiento del agente es bloqueado hasta que lleguen más mensajes.

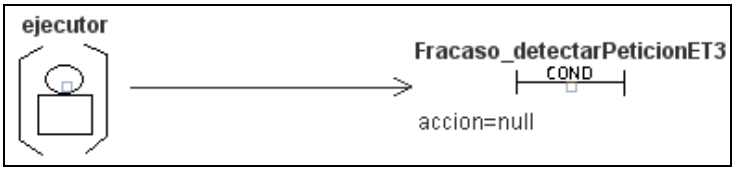


Figura B.30: Condiciones para dar por fallido el objetivo *DetectarPeticionET*.

2.1.17 *Satisface_procesarET1*: el objetivo *ProcesarET* se satisface cuando se produce el hecho *resultadoTareas*. La satisfacción se indica con el slot *resultado* cuyo valor sea diferente a una cadena vacía.

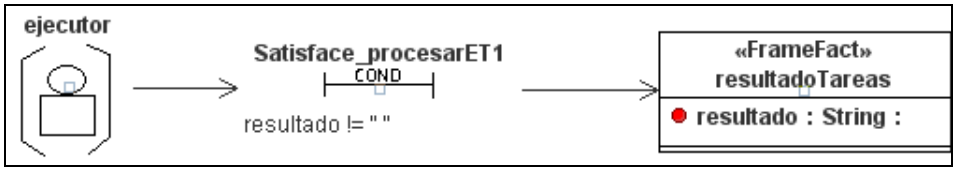


Figura B.31: Condiciones para satisfacer el objetivo *ProcesarET*.

2.1.18 ***Fracaso_procesarET1*:** el objetivo *ProcesarET* falla cuando el valor obtenido en el slot *resultado* es una cadena vacía.

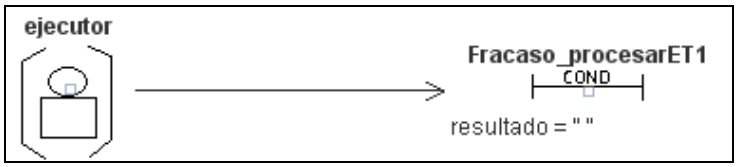


Figura B.32: Condiciones para dar por fallido el objetivo *ProcesarET*.

2.1.19 ***Satisface_procesarET2***: el objetivo *ProcesarET* se satisface cuando se produce el hecho *datosRespuesta*. La satisfacción se indica con el slot *datos* cuyo valor es diferente a null.

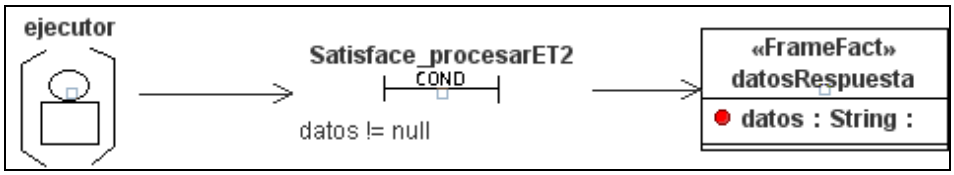


Figura B.33: Condiciones para satisfacer el objetivo *ProcesarET*.

2.1.20 ***Fracaso_procesarET2*:** el objetivo *ProcesarET* falla cuando el valor obtenido en el slot *datos* es null.

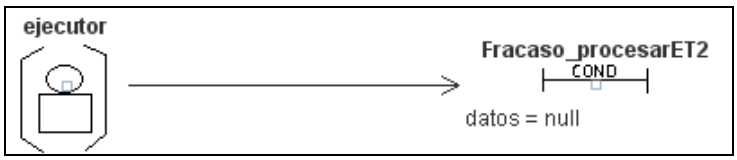


Figura B.34: Condiciones para dar por fallido el objetivo *ProcesarET*.

2.1.21 ***Satisface_procesarET3***: el objetivo *ProcesarET* se satisface cuando se produce el hecho *datosRespuestaCoordinador*. La satisfacción se indica con la comprobación del que el mensaje recibido del agente *Buscador* no es null.

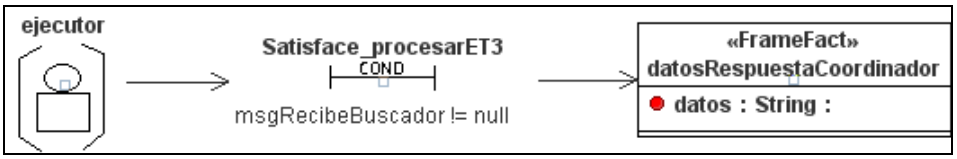


Figura B.35: Condiciones para satisfacer el objetivo *ProcesarET*.

2.1.22 ***Fracaso_procesarET3*:** el objetivo *ProcesarET* falla cuando el mensaje recibido del agente *Buscador* es null. El comportamiento del agente es bloqueado hasta que lleguen más mensajes.

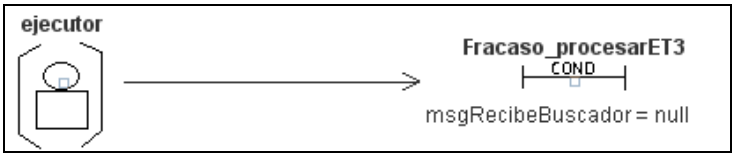


Figura B.36: Condiciones para dar por fallido el objetivo *ProcesarET*.

2.1.23 ***Satisface_darResultadoET***: el objetivo *DarResultadoET* se satisface cuando se produce el hecho *HttpServletResponse*. La satisfacción se indica con el slot *datos* cuya longitud sea mayor a 0.

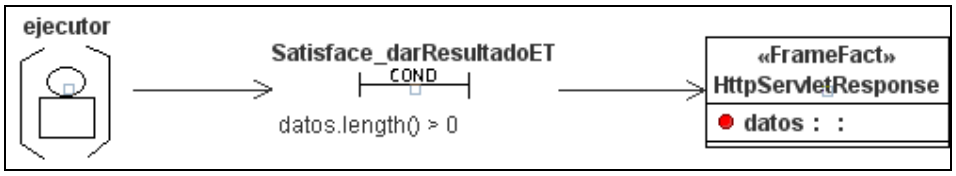


Figura B.37: Condiciones para satisfacer el objetivo *DarResultadoET.*

2.1.24 ***Fracaso_darResultadoET*:** el objetivo *DarResultadoET* falla cuando el valor obtenido en el slot *datos* tiene una longitud de 0.

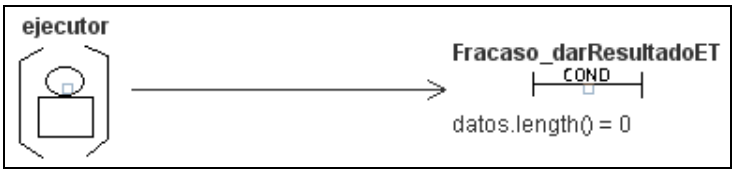


Figura B.38: Condiciones para dar por fallido el objetivo *DarResultadoET*.

2.1.25 ***Satisface_detectarPeticionS1*:** el objetivo *DetectarPeticionS* se satisface cuando se produce el hecho *datosCaso3*. La satisfacción se indica con el slot *accion* en 3.

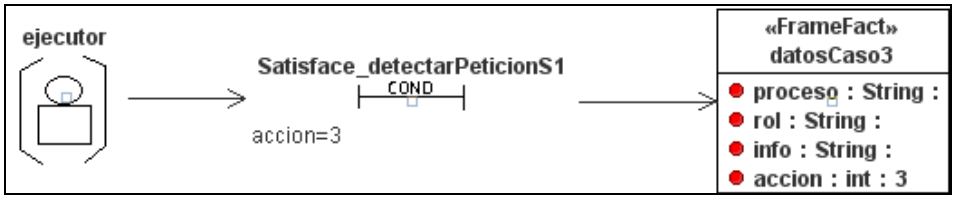


Figura B.39: Condiciones para satisfacer el objetivo *DetectarPeticionS*.

2.1.26 ***Fracaso_detectarPeticionS1*:** el objetivo *DetectarPeticionS* falla cuando no es posible obtener un valor para el slot *accion*. El comportamiento del agente es bloqueado hasta que lleguen más mensajes.

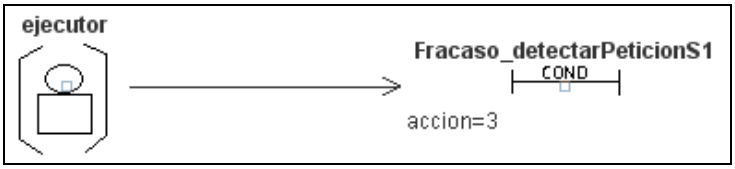


Figura B.40: Condiciones para dar por fallido el objetivo *DetectarPeticionS*.

2.1.27 ***Satisface_detectarPeticionS2***: el objetivo *DetectarPeticionS* se satisface cuando se produce el hecho *datosBuscar*. La satisfacción se indica con la comprobación del que el mensaje recibido del agente *Coordinador* no es null.

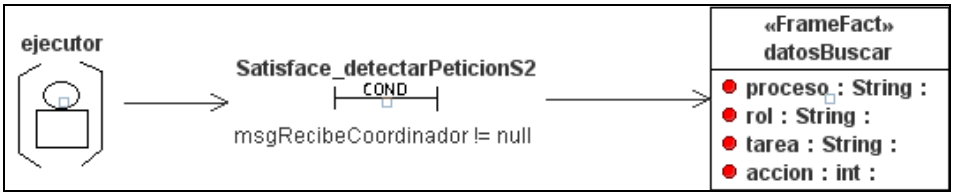


Figura B.41: Condiciones para satisfacer el objetivo *DetectarPeticionS*.

2.1.28 *Fracaso_detectarPeticionS2*: el objetivo *DetectarPeticionS* falla cuando el mensaje recibido del agente *Coordinador* es null. El comportamiento del agente es bloqueado hasta que lleguen más mensajes.

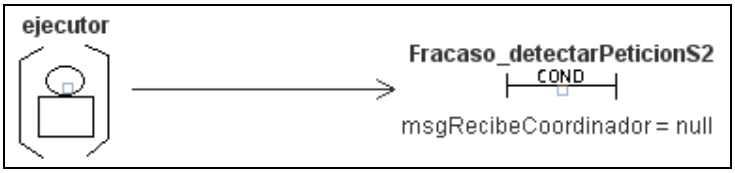


Figura B.42: Condiciones para dar por fallido el objetivo *DetectarPeticionS*.

2.1.29 *Satisface_detectarPeticionS3*: el objetivo *DetectarPeticionS* se satisface cuando se produce el hecho *infoSugerencia*. La satisfacción se indica con *accion* en 3.

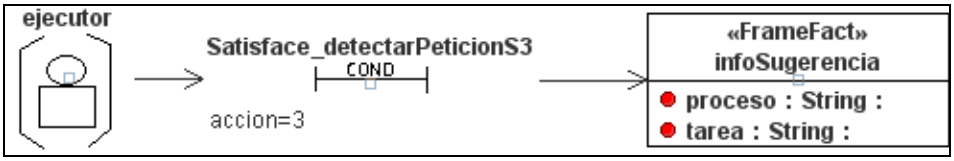


Figura B.43: Condiciones para satisfacer el objetivo *DetectarPeticionS*.

2.1.30 *Fracaso_detectarPeticionS3*: el objetivo *DetectarPeticionS* falla cuando no es posible obtener un valor para *accion*. El comportamiento del agente es bloqueado hasta que lleguen más mensajes.

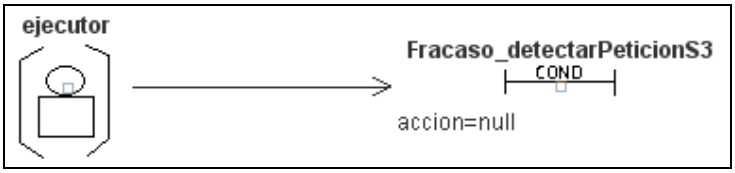


Figura B.44: Condiciones para dar por fallido el objetivo *DetectarPeticionS*.

2.1.31 *Satisface_procesarS1*: el objetivo *ProcesarS* se satisface cuando se produce el hecho *resultadoSugerencia*. La satisfacción se indica con el slot *resultado* cuyo valor sea diferente a una cadena vacía.

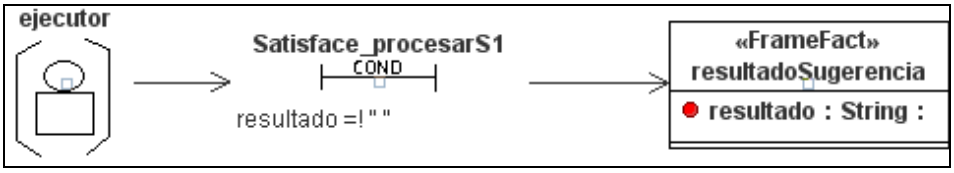


Figura B.45: Condiciones para satisfacer el objetivo *ProcesarS*.

2.1.32 ***Fracaso_procesarS1*:** el objetivo *ProcesarS* falla cuando el valor obtenido en el slot *resultado* es una cadena vacía.

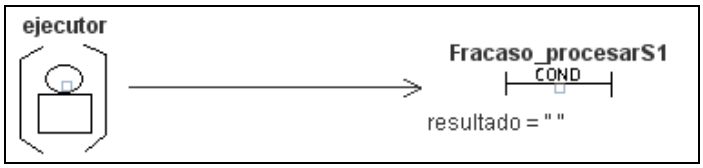


Figura B.46: Condiciones para dar por fallido el objetivo *ProcesarS*.

2.1.33 ***Satisface_procesarS2***: el objetivo *ProcesarS* se satisface cuando se produce el hecho *datosRespuesta*. La satisfacción se indica con el slot *datos* cuyo valor es diferente a null.

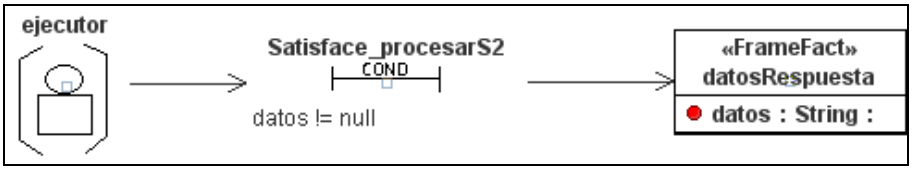


Figura B.47: Condiciones para satisfacer el objetivo *ProcesarS*.

2.1.34 ***Fracaso_procesarS2*:** el objetivo *ProcesarS* falla cuando el valor obtenido en el slot *datos* es null.

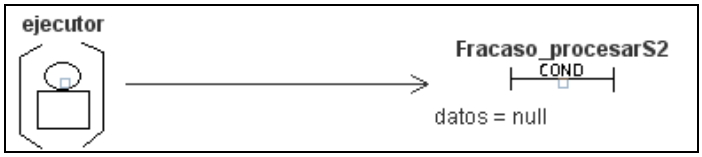


Figura B.48: Condiciones para dar por fallido el objetivo *ProcesarS*.

2.1.35 ***Satisface_procesarS3***: el objetivo *ProcesarS* se satisface cuando se produce el hecho *datosRespuestaCoordinador*. La satisfacción se indica con la comprobación del que el mensaje recibido del agente *Buscador* no es null.

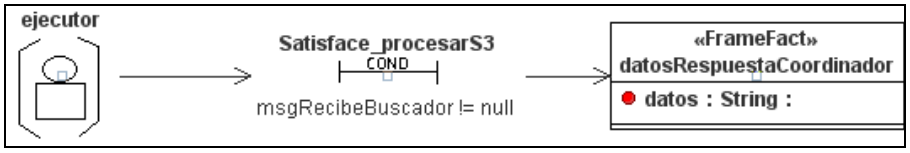


Figura B.49: Condiciones para satisfacer el objetivo *ProcesarS*.

2.1.36 ***Fracaso_procesarS3*:** el objetivo *ProcesarS* falla cuando el mensaje recibido del agente *Buscador* es null. El comportamiento del agente es bloqueado hasta que lleguen más mensajes.

Figura B.50: Condiciones para dar por fallido el objetivo *ProcesarS*.

2.1.37 *Satisface_darResultadoS*: el objetivo *DarResultadoS* se satisface cuando se produce el hecho *HttpServletResponse*. La satisfacción se indica con el slot *datos* cuya longitud sea mayor a 0.

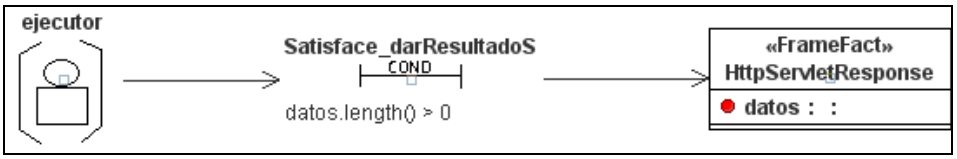


Figura B.51: Condiciones para satisfacer el objetivo *DarResultadoS.*

2.1.38 *Fracaso_darResultadoS*: el objetivo *DarResultadoS* falla cuando el valor obtenido en el slot *datos* tiene una longitud de 0.

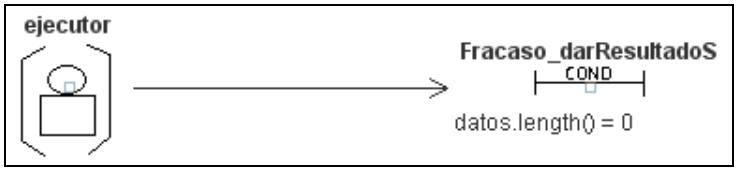


Figura B.52: Condiciones para dar por fallido el objetivo *DarResultadoS*.

2.1.39 *Satisface_detectarAltaS1*: el objetivo *DetectarAltaS* se satisface cuando se produce el hecho *datosCaso4*. La satisfacción se indica con el slot *accion* en 4.

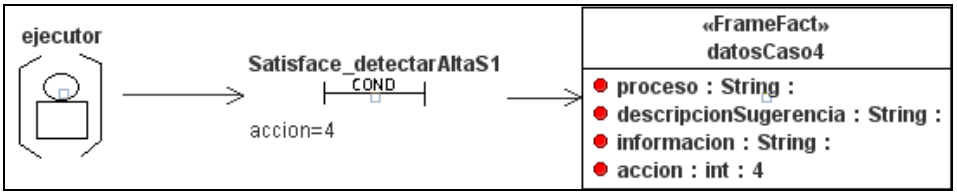


Figura B.53: Condiciones para satisfacer el objetivo *DetectarAltaS*.

2.1.40 *Fracaso_detectarAltaS1*: el objetivo *DetectarAltaS* falla cuando no es posible obtener un valor para el slot *accion*. El comportamiento del agente es bloqueado hasta que lleguen más mensajes.

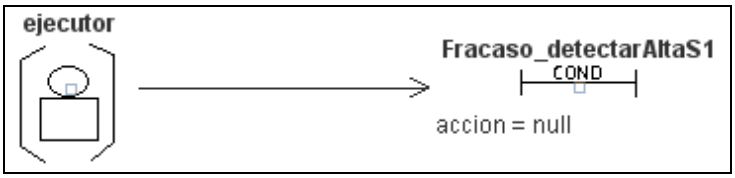


Figura B.54: Condiciones para dar por fallido el objetivo *DetectarAltaS*.

2.1.41 *Satisface_detectarAltaS2*: el objetivo *DetectarAltaS* se satisface cuando se produce el hecho *guardarSugerencia*. La satisfacción se indica con la *accion* en 4.

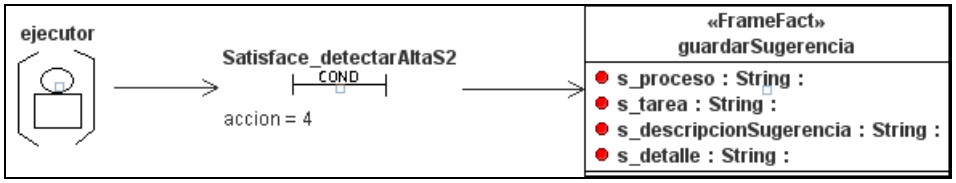


Figura B.55: Condiciones para satisfacer el objetivo *DetectarAltaS*.

2.1.42 *Fracaso_detectarAltaS2*: el objetivo *DetectarAltaS* falla cuando no es posible obtener un valor para *accion*. El comportamiento del agente es bloqueado hasta que lleguen más mensajes.

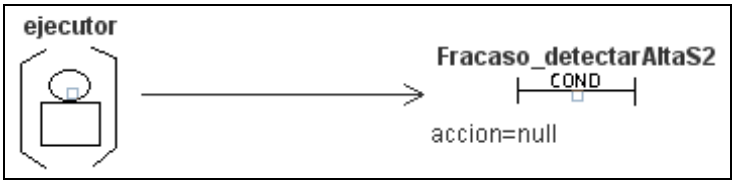


Figura B.56: Condiciones para dar por fallido el objetivo *DetectarAltaS*.

2.1.43 *Satisface_procesarAltaS1*: el objetivo *ProcesarAltaS* se satisface cuando se produce el hecho *resultadoGuardarSugerencia*. La satisfacción se indica con el slot *numFilas* cuyo valor sea diferente a 0.

Figura B.57: Condiciones para satisfacer el objetivo *ProcesarAltaS*.

2.1.44 *Fracaso_procesarAltaS1*: el objetivo *ProcesarAltaS* falla cuando el valor obtenido en el slot *numFilas* es 0.

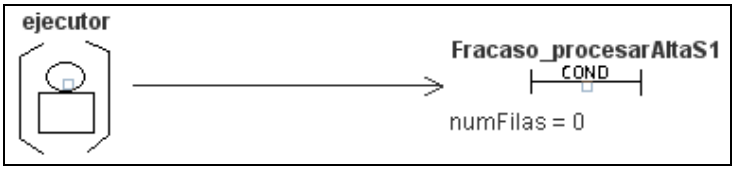


Figura B.58: Condiciones para dar por fallido el objetivo *ProcesarAltaS*.

2.1.45 *Satisface_procesarAltaS2*: el objetivo *ProcesarAltaS* se satisface cuando se produce el hecho *datosRespuesta*. La satisfacción se indica con el slot *datos* cuyo valor es diferente a null.

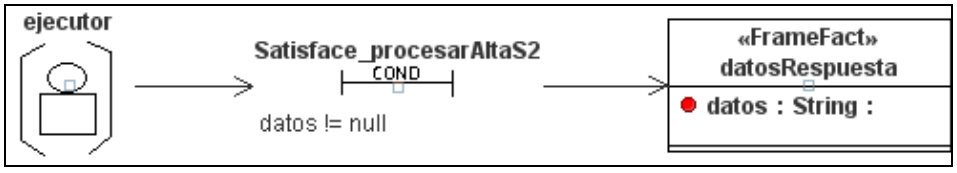


Figura B.59: Condiciones para satisfacer el objetivo *ProcesarAltaS*.

2.1.46 *Fracaso_procesarAltaS2*: el objetivo *ProcesarAltaS* falla cuando el valor obtenido en el slot *datos* es null.

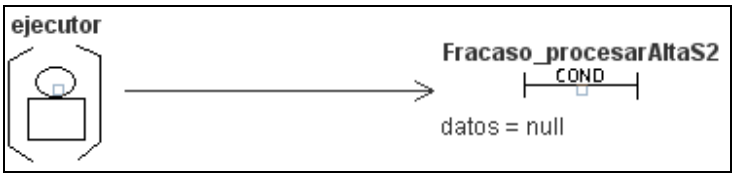


Figura B.60: Condiciones para dar por fallido el objetivo *ProcesarAltaS*.

2.1.47 *Satisface_darResultadoAltaS*: el objetivo *DarResultadoAltaS* se satisface cuando se produce el hecho *HttpServletResponse*. La satisfacción se indica con el slot *datos* cuya longitud sea mayor a 0.

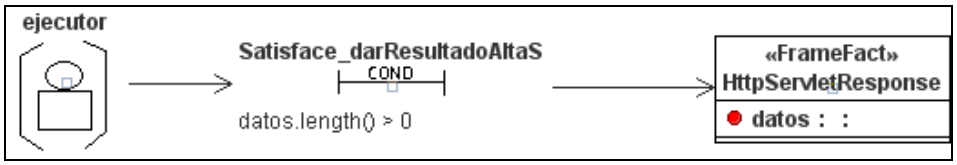


Figura B.61: Condiciones para satisfacer el objetivo *DarResultadoAltaS.*

2.1.48 *Fracaso_darResultadoAltaS*: el objetivo *DarResultadoAltaS* falla cuando el valor obtenido en el slot *datos* tiene una longitud de 0.

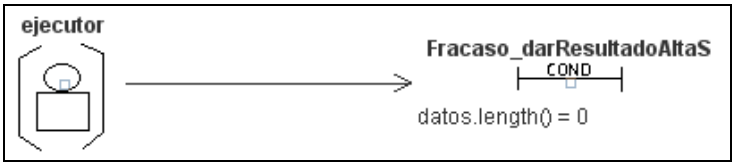


Figura B.62: Condiciones para dar por fallido el objetivo *DarResultadoAltaS*.

2.2 Descripción de los flujos de trabajo

2.2.1 *Flujo de trabajo: Agregar_sugerencias*

Define las tareas necesarias para proporcionar al usuario la posibilidad de anexar nuevas sugerencias sobre cierta tarea que realiza en HeAP MoProSoft. Se inicia cuando el usuario solicita este servicio al seleccionar en la interfaz de HeAP MoProsoft la opción correspondiente y ha elegido la tarea que desea anexarle la sugerencia. En la tabla B.1 se presentan por orden alfabético a las tareas que conforman a este flujo de trabajo.

Tarea	Descripción
enviar_resultado	Envía el resultado obtenido al realizar la transacción solicitada en la fuente de datos.
guardar_sugerencia	Procesa el almacenamiento de la sugerencia en la fuente de datos adecuada.
obtener_solicitud_alta	Obtener la solicitud de dar de alta a una nueva sugerencia para una tarea que realiza el usuario en HeAP MoProSoft.
solicitar_incorporacion_su	Solicita la incorporación de una sugerencia. Esta tarea

gerencia	dispara una interacción que persigue el proporcionar al usuario la posibilidad de anexar nuevas sugerencias sobre cierta tarea que realiza en HeAP MoProSoft.
visualizar_resultado	Procesa la visualización del resultado obtenido para el usuario.

Tabla B.1: Tareas que conforman al flujo de trabajo *Agregar_sugerencias*.

A continuación se presentan las cinco instancias del meta-modelo de Organización para la descripción detallada del flujo de trabajo.

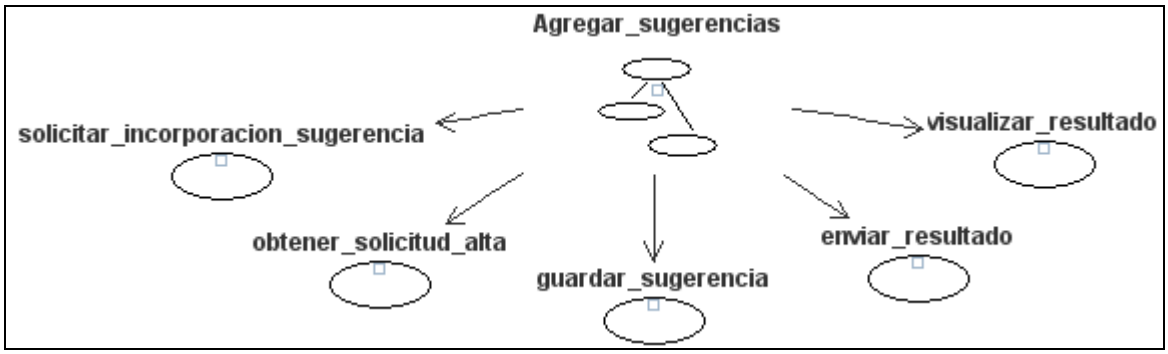


Figura B.63: Tareas asociadas al flujo de trabajo *Agregar_sugerencias*.

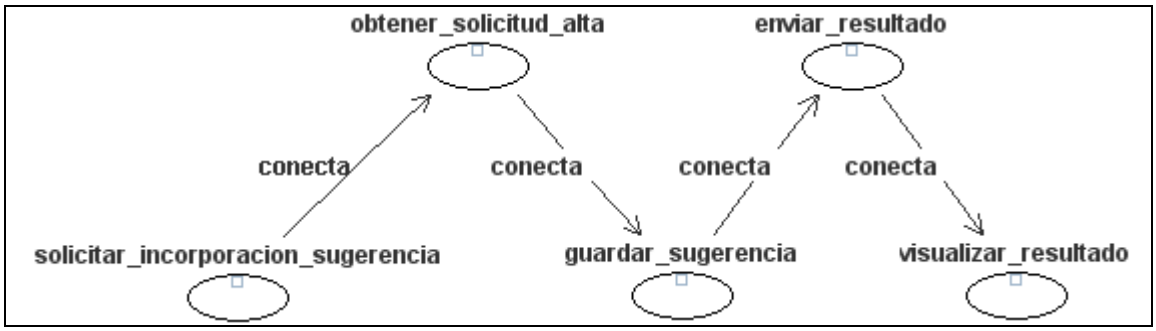


Figura B.64: Dependencias entre las tareas en el flujo de trabajo *Agregar_sugerencias*.

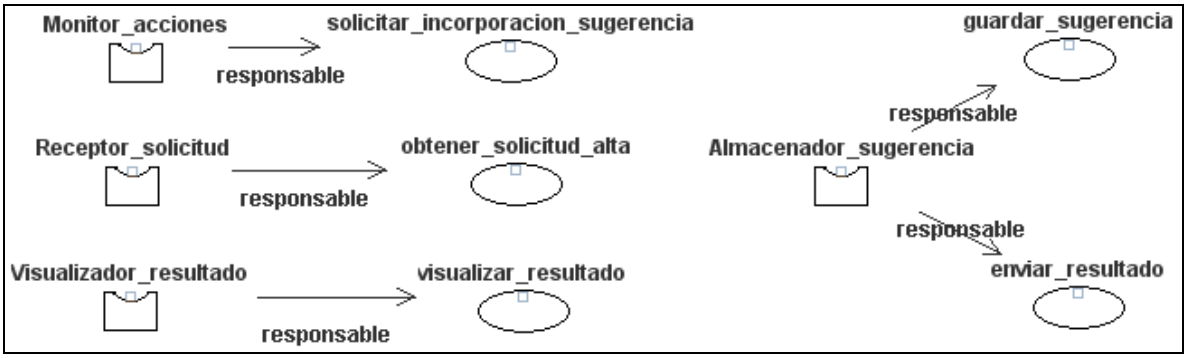


Figura B.65: Roles responsables de las tareas del flujo de trabajo *Agregar_sugerencias*.

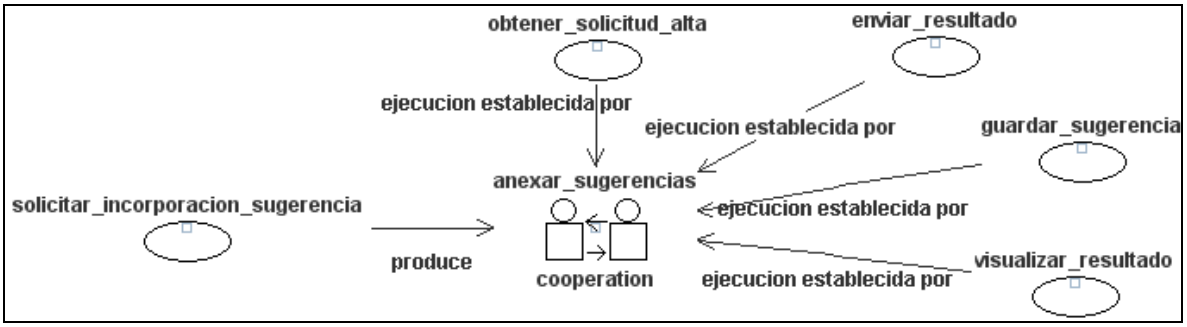


Figura B.66: Relación de las tareas del flujo de trabajo *Agregar_sugerencias* con la interacción *anexar_sugerencias*.

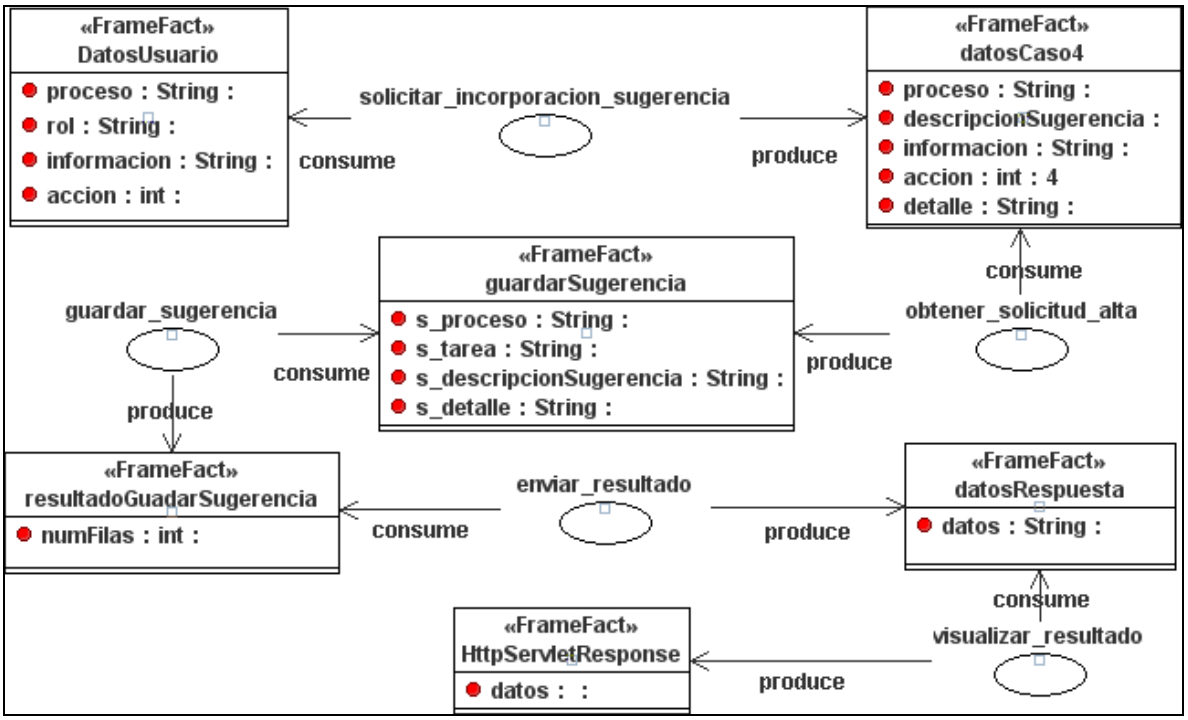


Figura B.67: Descripción detallada del flujo de trabajo *Agregar_sugerencias*.

2.2.2 *Flujo de trabajo: Obtener_estado_tareas*

Define las tareas necesarias para proporcionar al usuario la consulta del estado de las tareas que realiza en HeAP MoProSoft. Se inicia cuando el usuario solicita este servicio al seleccionar en la interfaz de HeAP MoProsoft la opción correspondiente. En la tabla B.2 se presentan las tareas que conforman a este flujo de trabajo.

Tarea	Descripción
buscar_info_tareas	Procesa la búsqueda solicitada en la fuente de datos adecuada.
enviar_informacion	Envía la información obtenida al consultar en la fuente de datos.

evaluar_tareas	Evalúa la información recibida de las tareas y envía el resultado.
obtener_solicitud_buscar	Obtener la solicitud de búsqueda y los datos necesarios para llevarla a cabo.
solicitar_estado_tareas	Solicita el estado de las tareas. Esta tarea dispara una interacción que persigue el proporcionar al usuario los estados de las tareas que realiza en HeAP MoProSoft.
solicitar_info_tareas	Envía una solicitud para obtener información de las tareas.
visualizar_resultado	Procesa la visualización del resultado obtenido para el usuario.

Tabla B.2: Tareas que conforman al flujo de trabajo *Obtener_estado_tareas*.

A continuación se presentan las cinco instancias del meta-modelo de Organización para la descripción detallada del flujo de trabajo.

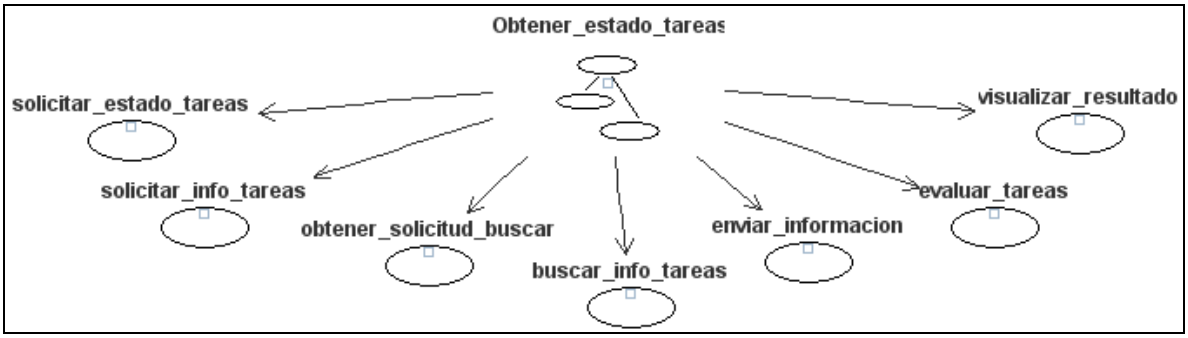


Figura B.68: Tareas asociadas al flujo de trabajo *Obtener_estado_tareas*.

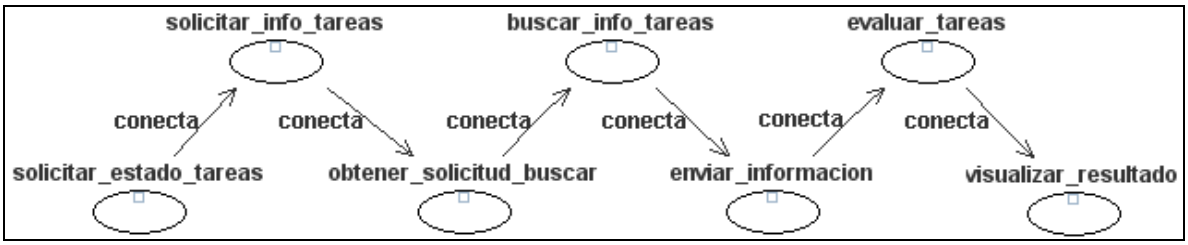


Figura B.69: Dependencias entre las tareas en el flujo de trabajo *Obtener_estado_tareas*.

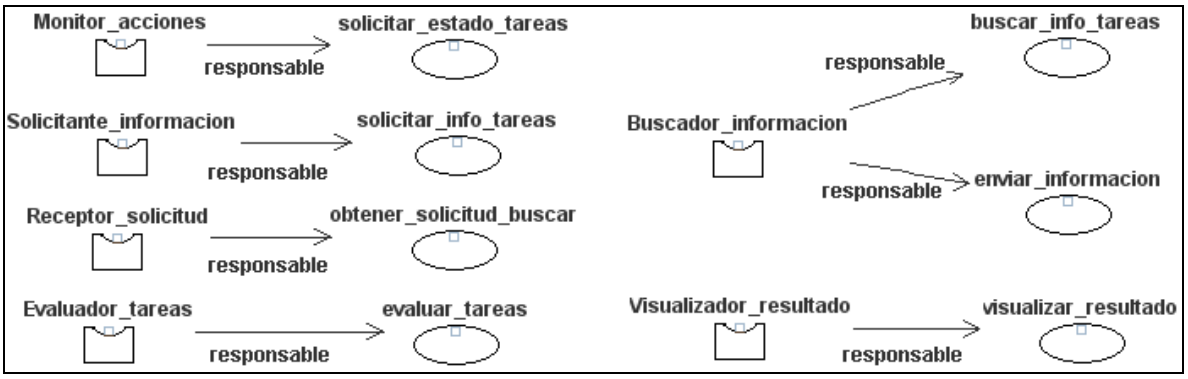


Figura B.70: Roles responsables de las tareas del flujo de trabajo *Obtener_estado_tareas*.

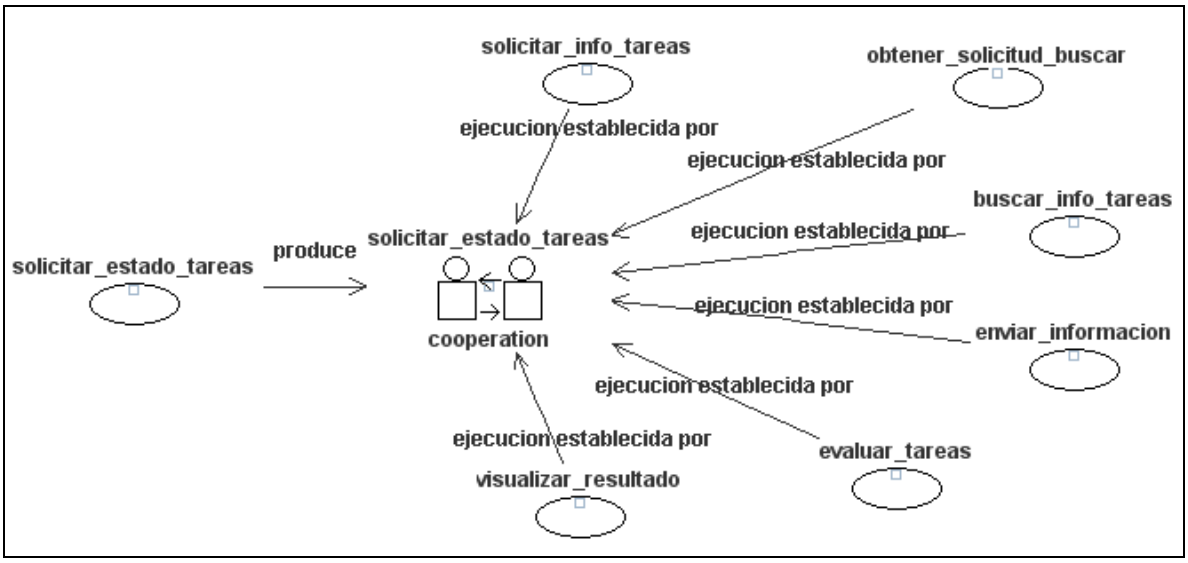


Figura B.71: Relación de las tareas del flujo de trabajo *Obtener_estado_tareas* con la interacción *solicitar_estado_tareas*.

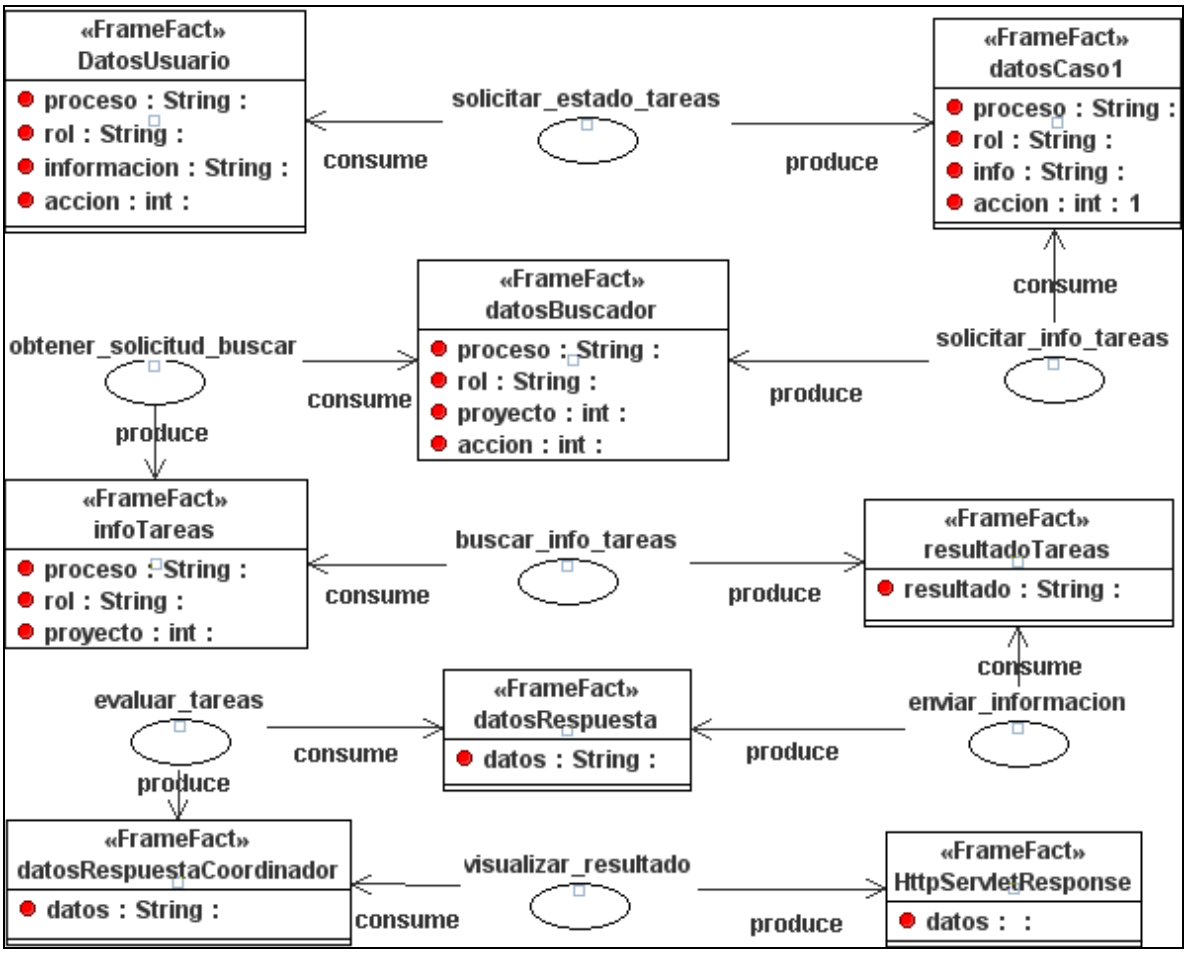


Figura B.72: Descripción detallada del flujo de trabajo *Obtener_estado_tareas*.

2.2.3 *Flujo de trabajo: Obtener_sugerencias*

Define las tareas necesarias para proporcionar al usuario la consulta de sugerencias sobre cómo llevar a cabo cierta tarea que realiza en HeAP MoProSoft. Se inicia cuando el usuario solicita este servicio al seleccionar en la interfaz de HeAP MoProsoft la opción correspondiente y ha elegido la tarea que desea consultar sus sugerencias. En la tabla B.3 se presentan las tareas que conforman a este flujo de trabajo.

Tarea	Descripción
buscar_info_sugerencias	Procesa la búsqueda solicitada en la fuente de datos adecuada.
enviar_informacion	Envía la información obtenida al consultar en la fuente de datos.
evaluar_sugerencias	Evalúa la información recibida de las sugerencias y envía el resultado.
obtener_solicitud_buscar	Obtener la solicitud de búsqueda y los datos necesarios para llevarla a cabo.
solicitar_sugerencias	Solicita las sugerencias de una tarea. Esta tarea dispara una interacción que persigue el proporcionar al usuario las sugerencias sobre cómo llevar a cabo cierta tarea que realiza en HeAP MoProSoft.
solicitar_info_sugerencias	Envía una solicitud para obtener información de las sugerencias.
visualizar_resultado	Procesa la visualización del resultado obtenido para el usuario.

Tabla B.3: Tareas que conforman al flujo de trabajo *Obtener_sugerencias*.

A continuación se presentan las cinco instancias del meta-modelo de Organización para la descripción detallada del flujo de trabajo.

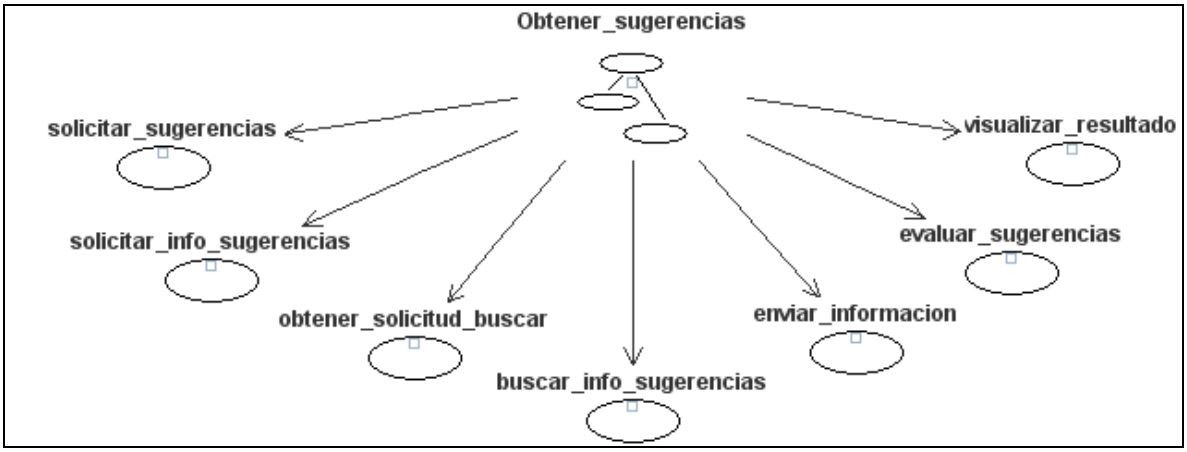


Figura B.73: Tareas asociadas al flujo de trabajo *Obtener_sugerencias*.

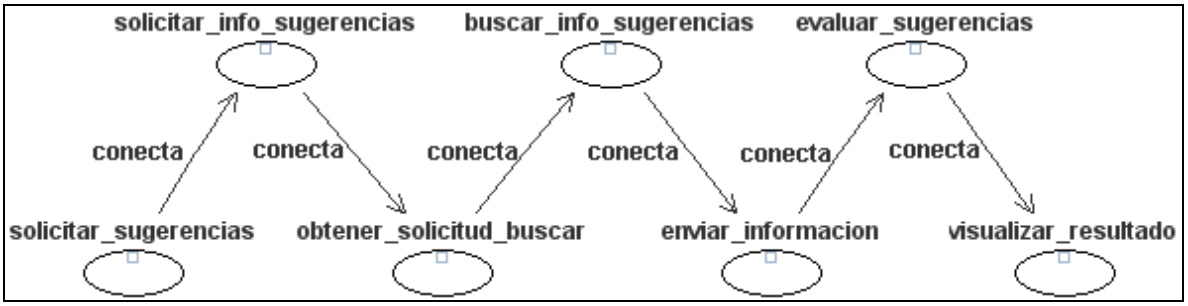


Figura B.74: Dependencias entre las tareas en el flujo de trabajo *Obtener_sugerencias*.

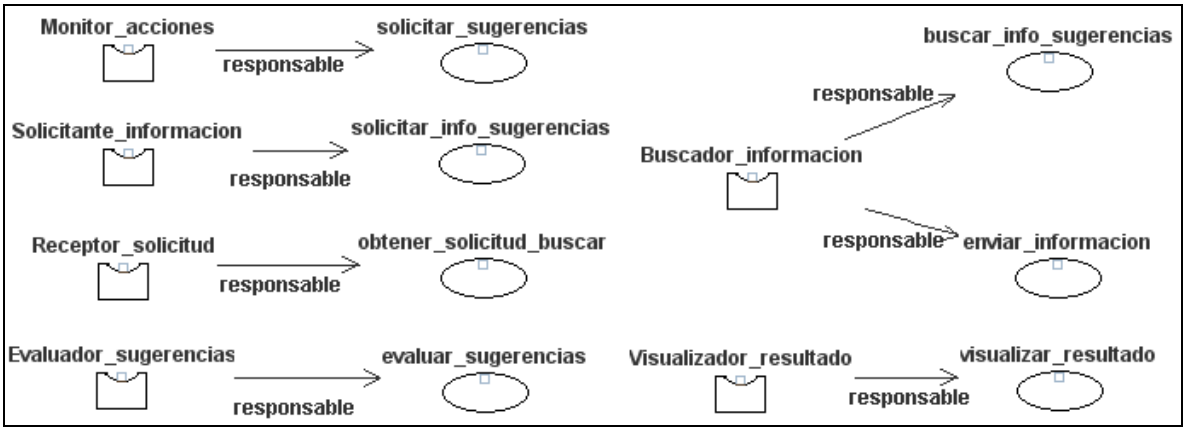


Figura B.75: Roles responsables de las tareas del flujo de trabajo *Obtener_sugerencias*.

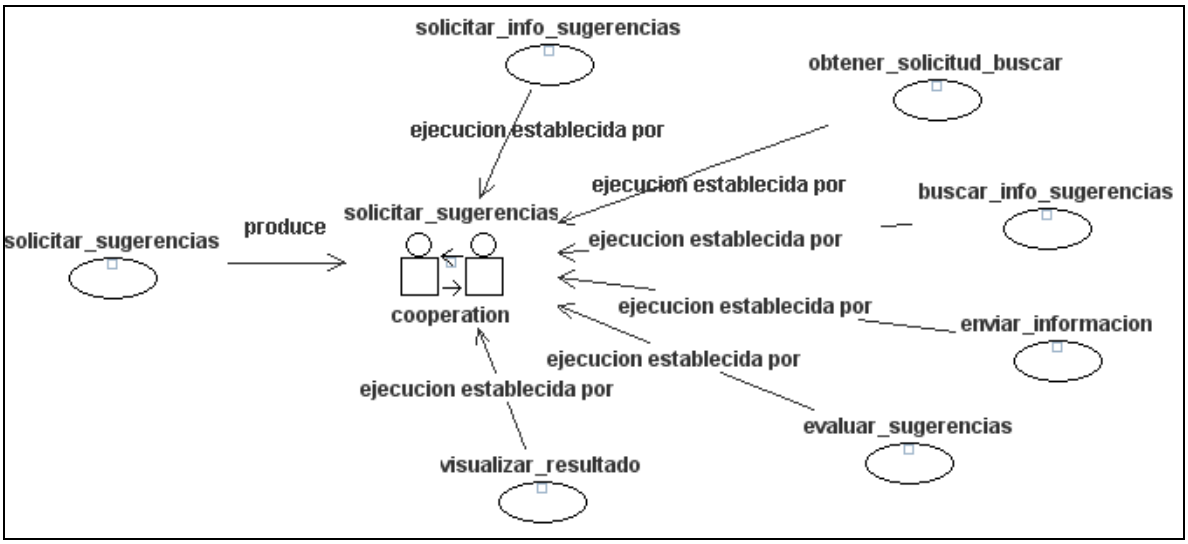


Figura B.76: Relación de las tareas del flujo de trabajo *Obtener_sugerencias* con la interacción *solicitar_sugerencias*.

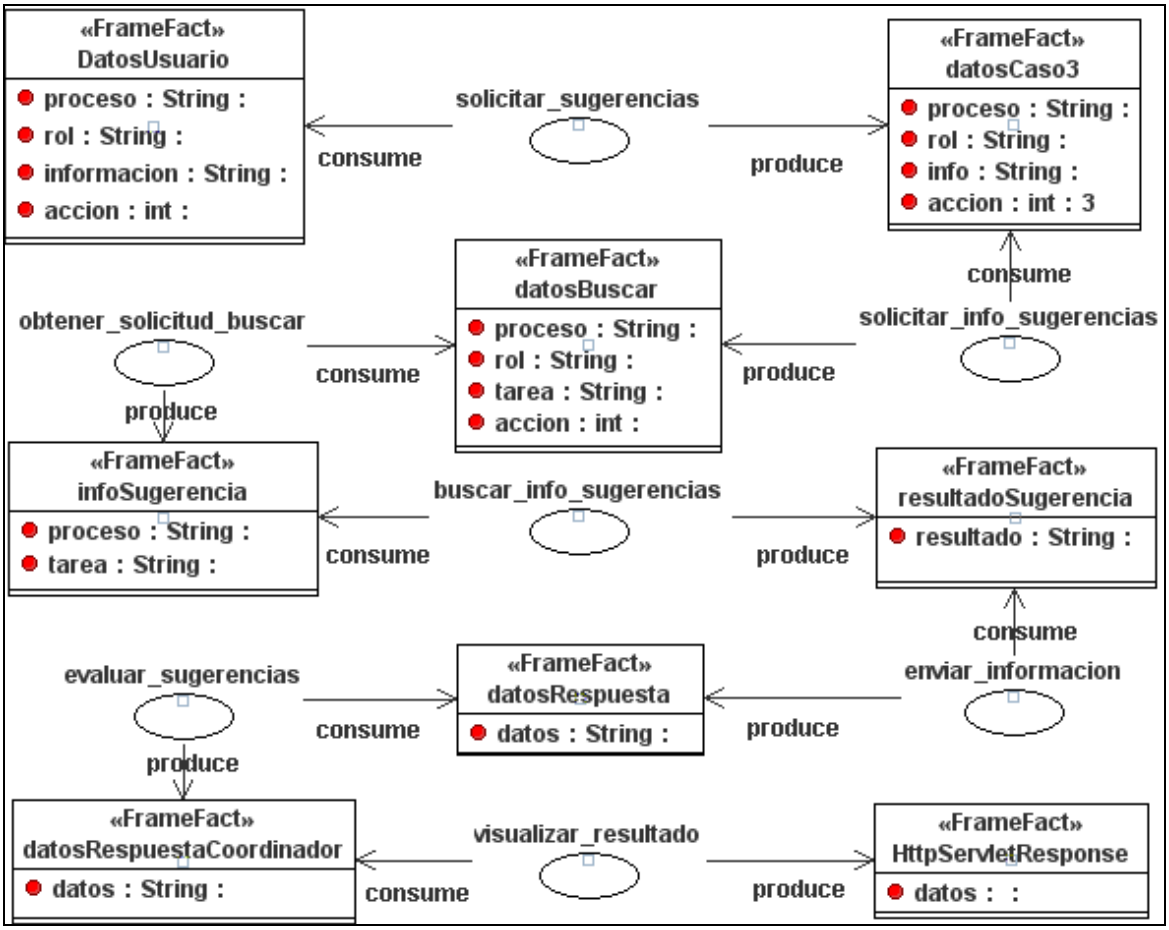


Figura B.77: Descripción detallada del flujo de trabajo *Obtener_sugerencias*.

2.3 Diagramas de colaboración detallados de las interacciones

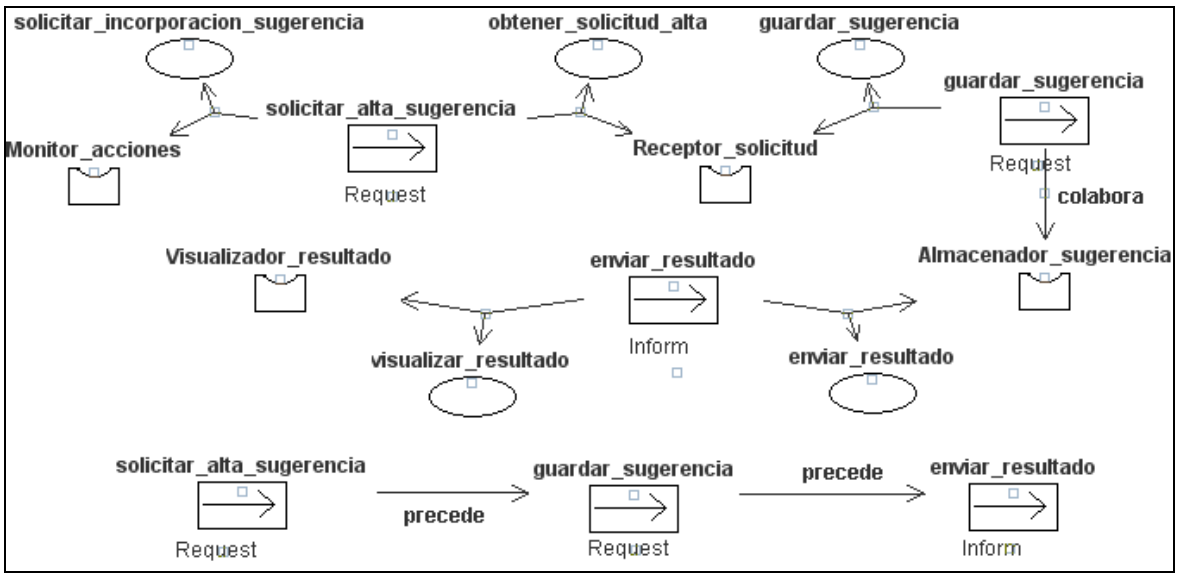


Figura B.78: Diagrama de colaboración detallado para la interacción *anexar_sugerencias*.

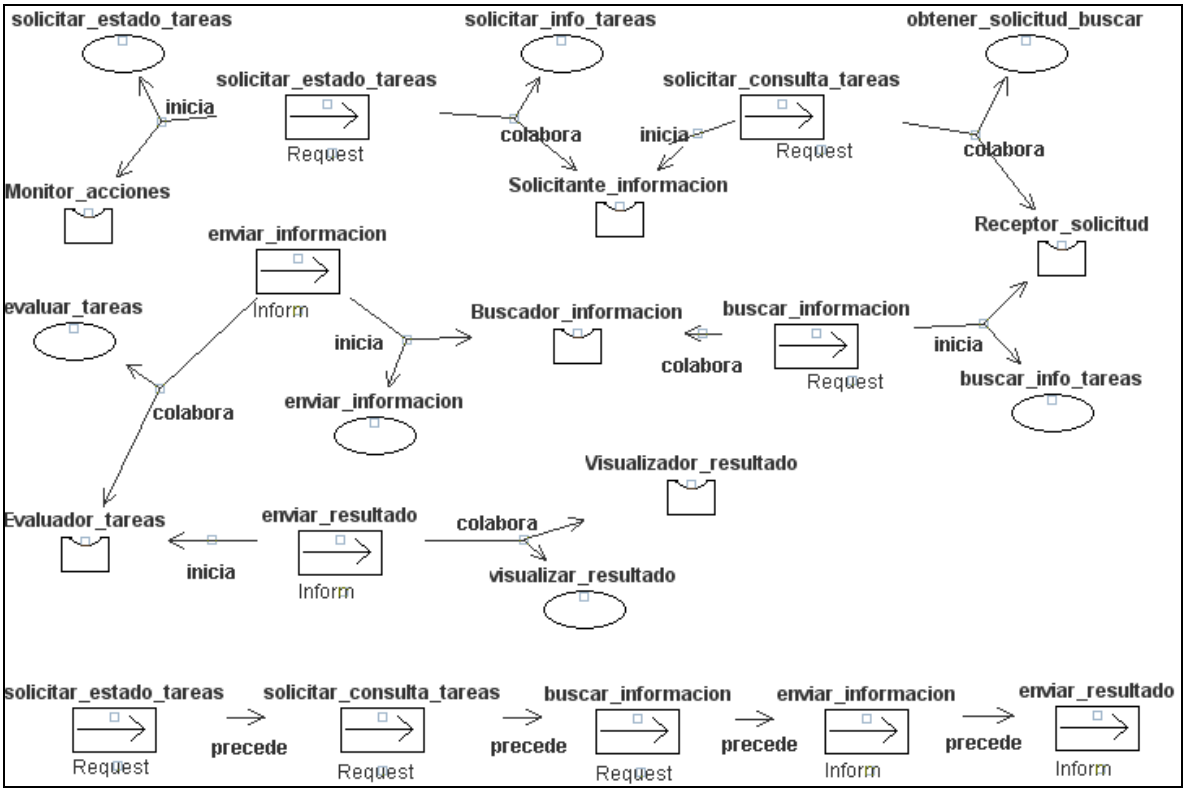


Figura B.79: Diagrama de colaboración detallado para la interacción *solicitar_estado_tareas.*

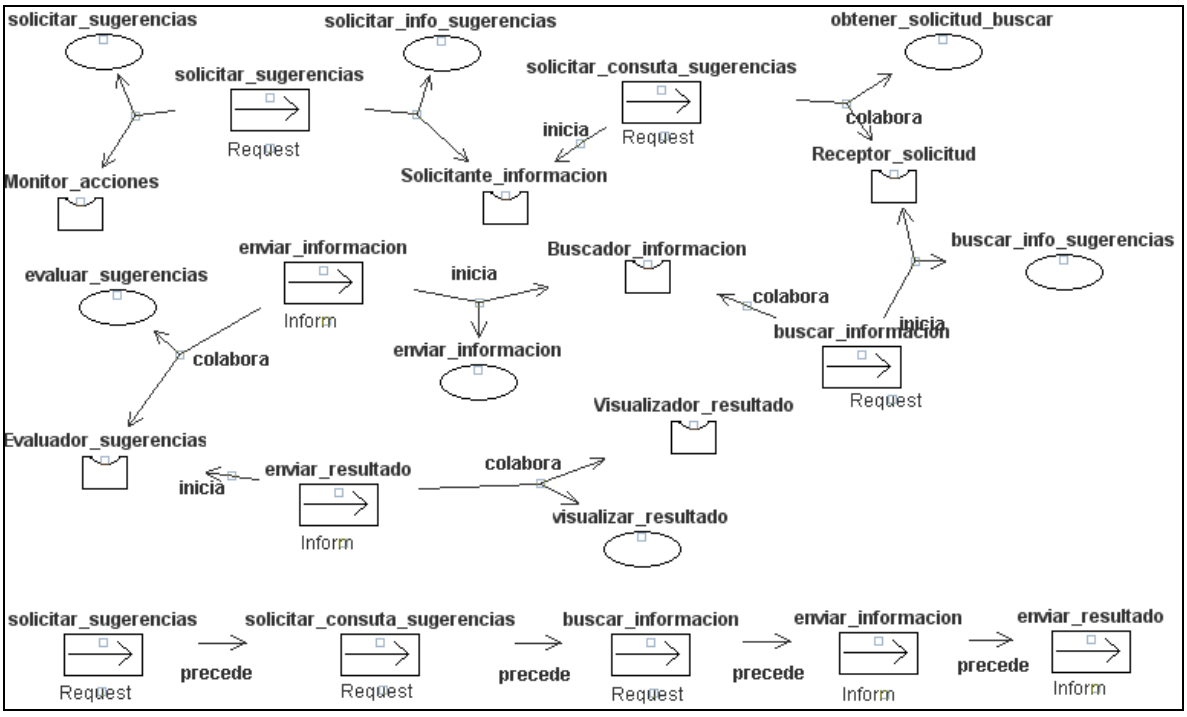


Figura B.80: Diagrama de colaboración detallado para la interacción *solicitar_sugerencias.*

Apéndice C

Base de Datos de Soporte

Esta sección contiene la descripción de las tablas que conforman a la Base de Datos de Soporte descrita en el quinto capítulo.

DESCRIPCIÓN DE LAS TABLAS

Tabla proceso

Tabla que contiene la definición general de los procesos descrita en MoProSoft. Un proceso es definido en MoProSoft como un conjunto de prácticas relacionadas entre si, llevadas a cabo a través de roles y por elementos automatizados, que utilizando recursos y a partir de insumos producen un satisfactor de negocio para el cliente. A continuación se presenta en la tabla C.1 la información de los atributos que la conforman.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
id_proceso	Entero	Número secuencial asignado sólo para identificar a los procesos especificados en MoProSoft.	En el intervalo de 1 a N. No puede quedar vacío.
acronimo_proceso	Carácter	Acrónimo establecido en la definición de los elementos de la estructura del modelo de procesos.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
nombre_proceso	Carácter	Nombre del proceso perteneciente a MoProSoft.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
categoria_proceso	Carácter	Nombre de la categoría a la que pertenece el proceso y el acrónimo entre paréntesis.	Alfabeto, signos de puntuación y números.
proposito_proceso	Carácter	Objetivos generales medibles y resultados esperados de la implantación efectiva del proceso.	Alfabeto, signos de puntuación y números.
descripcion_proceso	Carácter	Descripción general de las actividades y productos que componen el flujo de trabajo del proceso.	Alfabeto, signos de puntuación y números.
responsable_proceso	Carácter	Rol principal responsable por la ejecución del proceso.	Alfabeto, signos de puntuación y números.
acronimo_responsable	Carácter	Acrónimo establecido para el rol principal responsable por la ejecución del proceso.	Alfabeto, signos de puntuación y números.
autoridad_proceso	Carácter	Rol responsable por validar la ejecución del proceso y el cumplimiento de su propósito.	Alfabeto, signos de puntuación y números.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
subprocesos_proceso	Carácter	Subprocesos del proceso.	Alfabeto, signos de puntuación y números.
procesos_relacionados	Carácter	Son los nombres de los procesos relacionados.	Alfabeto, signos de puntuación y números.

Tabla C.1: Información de los atributos de la tabla *proceso*.

Tabla objetivo

Tabla que contiene los objetivos específicos cuya finalidad es asegurar el cumplimiento del propósito del proceso. Un objetivo es definido en MoProSoft como un fin a que se dirige o encamina una acción u operación. A continuación se presenta en la tabla C.2 la información de los atributos que la conforman.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
id_objetivo	Entero	Número secuencial asignado para identificar a los objetivos de los procesos.	En el intervalo de 1 a N. No puede quedar vacío.
id_proceso	Entero	Número secuencial asignado para identificar a los procesos especificados en MoProSoft.	En el intervalo de 1 a N. No puede quedar vacío.
numero_objetivo	Entero	Número asignado al objetivo como parte de su identificación en el proceso.	En el intervalo de 1 a N. No puede quedar vacío.
descripcion_objetivo	Carácter	Descripción general del objetivo.	Alfabeto, signos de puntuación y números. No puede quedar vacío.

Tabla C.2: Información de los atributos de la tabla *objetivo*.

Tabla indicador

Tabla que contiene los indicadores para evaluar la efectividad del cumplimiento de los objetivos del proceso. Un indicador es definido en MoProSoft como un mecanismo que sirve para mostrar o significar una cosa con evidencias y hechos. A continuación se presenta en la tabla C.3 la información de los atributos que la conforman.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
id_indicador	Entero	Número secuencial asignado para identificar a	En el intervalo de 1 a N. No puede

Nombre atributo	Tipo dato	Descripción	Dominio de valores
		los indicadores de los procesos.	quedar vacío.
id_proceso	Entero	Número secuencial asignado para identificar a los procesos especificados en MoProSoft.	En el intervalo de 1 a N. No puede quedar vacío.
numero_indicador	Entero	Número asignado al indicador como parte de su identificación en el proceso.	En el intervalo de 1 a N. No puede quedar vacío.
numero_objetivo	Entero	Número del objetivo al que da respuesta el indicador.	En el intervalo de 1 a N. No puede quedar vacío.
descripcion_indicador	Carácter	Descripción general del indicador.	Alfabeto, signos de puntuación y números. No puede quedar vacío.

Tabla C.3: Información de los atributos de la tabla *indicador*.

Tabla entrada

Tabla que contiene las entradas que son necesarias para el desarrollo del proceso. A continuación se presenta en la tabla C.4 la información de los atributos que la conforman.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
id_entrada	Entero	Número secuencial asignado para identificar a las entradas de los procesos.	En el intervalo de 1 a N. No puede quedar vacío.
id_proceso	Entero	Número secuencial asignado para identificar a los procesos especificados en MoProSoft.	En el intervalo de 1 a N. No puede quedar vacío.
nombre_entrada	Carácter	Nombre del producto o recurso.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
fuente_entrada	Carácter	Referencia al origen del producto o recurso.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
nivel_entrada	Carácter	Número(s) del nivel de capacidad de la entrada.	Alfabeto, signos de puntuación y números. No puede quedar vacío.

Tabla C.4: Información de los atributos de la tabla *entrada*.

Tabla salida

Tabla que contiene las salidas que son generadas al implementar el proceso. A continuación se presenta en la tabla C.5 la información de los atributos que la conforman.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
id_salida	Entero	Número secuencial asignado para identificar a las salidas de los procesos.	En el intervalo de 1 a N. No puede quedar vacío.
id_proceso	Entero	Número secuencial asignado para identificar a los procesos especificados en MoProSoft.	En el intervalo de 1 a N. No puede quedar vacío.
nombre_salida	Carácter	Nombre del producto o recurso.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
descripcion_salida	Carácter	Descripción y características del producto o recurso.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
destino_salida	Carácter	Referencia al destinatario del producto o recurso.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
nivel_salida	Carácter	Número(s) del nivel de capacidad de la salida.	Alfabeto, signos de puntuación y números. No puede quedar vacío.

Tabla C.5: Información de los atributos de la tabla *salida*.

Tabla producto_interno

Tabla que contiene los productos internos que son generados y utilizados en el propio proceso. A continuación se presenta en la tabla C.6 la información de los atributos que la conforman.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
id_producto_interno	Entero	Número secuencial asignado para identificar a los productos internos de los procesos.	En el intervalo de 1 a N. No puede quedar vacío.
id_proceso	Entero	Número secuencial asignado para identificar a los procesos especificados en MoProSoft.	En el intervalo de 1 a N. No puede quedar vacío.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
nombre_producto_interno	Carácter	Nombre del producto generado y utilizado en el propio proceso.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
descripcion_producto_interno	Carácter	Descripción y características del producto interno.	Alfabeto, signos de puntuación y números.
nivel_producto_interno	Carácter	Número(s) del nivel de capacidad del producto.	Alfabeto, signos de puntuación y números. No puede quedar vacío. No puede quedar vacío.

Tabla C.6: Información de los atributos de la tabla *producto_interno*.

Tabla rol_involucrado

Tabla que contiene los roles involucrados y su capacitación requerida para el proceso. A continuación se presenta en la tabla C.7 la información de los atributos que la conforman.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
id_rol	Entero	Número secuencial asignado para identificar a los roles involucrados de los procesos.	En el intervalo de 1 a N. No puede quedar vacío.
id_proceso	Entero	Número secuencial asignado para identificar a los procesos especificados en MoProSoft.	En el intervalo de 1 a N. No puede quedar vacío.
nombre_rol	Carácter	Nombre del rol involucrado en el proceso.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
abreviatura_rol	Carácter	Abreviatura del rol involucrado en el proceso.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
capacitacion_rol	Carácter	Capacitación requerida por el rol para poder ejecutar el proceso.	Alfabeto, signos de puntuación y números.
nivel_rol	Carácter	Número(s) del nivel de capacidad del rol.	Alfabeto, signos de puntuación y números. No puede quedar vacío. No puede quedar vacío.

Tabla C.7: Información de los atributos de la tabla *rol_involucrado*.

Tabla actividad

Tabla que contiene las actividades implicadas en el proceso. Una actividad es definida en MoProSoft como un conjunto de tareas específicas asignadas para su realización a uno o más roles. A continuación se presenta en la tabla C.8 la información de los atributos que la conforman.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
id_actividad	Entero	Número secuencial asignado para identificar a las actividades de los procesos.	En el intervalo de 1 a N. No puede quedar vacío.
id_proceso	Entero	Número secuencial asignado para identificar a los procesos especificados en MoProSoft.	En el intervalo de 1 a N. No puede quedar vacío.
identificador_actividad	Carácter	Identificador asignado a la actividad.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
nombre_actividad	Carácter	Nombre de la actividad del proceso.	Alfabeto, signos de puntuación y números. No puede quedar vacío.

Tabla C.8: Información de los atributos de la tabla *actividad*.

Tabla tarea

Tabla que contiene las tareas asociadas a las actividades implicadas en el proceso. A continuación se presenta en la tabla C.9 la información de los atributos que la conforman.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
id_tarea	Entero	Número secuencial asignado para identificar a las tareas de las actividades de los procesos.	En el intervalo de 1 a N. No puede quedar vacío.
id_proceso	Entero	Número secuencial asignado para identificar a los procesos especificados en MoProSoft.	En el intervalo de 1 a N. No puede quedar vacío.
id_actividad	Entero	Número secuencial asignado para identificar a las actividades de los procesos.	En el intervalo de 1 a N. No puede quedar vacío.
roles_tarea	Carácter	Abreviaturas de los roles responsables de la tarea.	Alfabeto, signos de puntuación y números. No puede quedar vacío.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
identificador_tarea	Carácter	Identificador asignado a la tarea.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
descripcion_tarea	Carácter	Descripción de la tarea.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
nivel_tarea	Carácter	Número(s) del nivel de capacidad de la tarea.	Alfabeto, signos de puntuación y números. No puede quedar vacío.

Tabla C.9: Información de los atributos de la tabla *tarea*.

Tabla verificacion_validacion

Tabla que contiene las verificaciones y validaciones asociadas a los productos generados en las actividades del proceso. A continuación se presenta en la tabla C.10 la información de los atributos que la conforman.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
id_ver_val	Entero	Número secuencial asignado para identificar a las verificaciones o validaciones de las actividades de los procesos.	En el intervalo de 1 a N. No puede quedar vacío.
id_proceso	Entero	Número secuencial asignado para identificar a los procesos especificados en MoProSoft.	En el intervalo de 1 a N. No puede quedar vacío.
identificador_ver_val	Carácter	Identificador asignado a la verificación o validación.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
actividad_ver_val	Carácter	Identificación de la tarea para la verificación o validación.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
producto_ver_val	Carácter	Nombre del producto para la verificación o validación.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
rol_ver_val	Carácter	Abreviatura del rol responsable de realizar la verificación o validación.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
descripcion_ver_val	Carácter	Descripción de la verificación	Alfabeto, signos de

Nombre atributo	Tipo dato	Descripción	Dominio de valores
		o validación que se hará al producto.	puntuación y números. No puede quedar vacío.
nivel_ver_val	Carácter	Número(s) del nivel de capacidad de la verificación o validación.	Alfabeto, signos de puntuación y números. No puede quedar vacío.

Tabla C.10: Información de los atributos de la tabla *verificacion_validacion*.

Tabla sugerencia

Tabla que contiene las sugerencias de cómo llevar a cabo las tareas con métodos recomendables y conocidos. A continuación se presenta en la tabla C.11 la información de los atributos que la conforman.

Nombre atributo	Tipo dato	Descripción	Dominio de valores
id_sugerencia	Entero	Número secuencial asignado para identificar a las sugerencias de las tareas comprendidas en las actividades de los procesos.	En el intervalo de 1 a N. No puede quedar vacío.
id_proceso	Entero	Número secuencial asignado sólo para identificar a los procesos especificados en MoProSoft.	En el intervalo de 1 a N. No puede quedar vacío.
identificador_tarea	Carácter	Identificador asignado a la tarea.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
descripcion_sugerencia	Carácter	Descripción de la sugerencia para la tarea.	Alfabeto, signos de puntuación y números. No puede quedar vacío.
detalle_sugerencia	Carácter	Es la dirección del archivo html que contiene la descripción detallada de la sugerencia para la tarea.	Alfabeto, signos de puntuación y números.

Tabla C.11: Información de los atributos de la tabla *sugerencia*.

Apéndice D

Pruebas

Esta sección contiene los resultados obtenidos en las pruebas unitarias de caja negra y las pruebas del sistema.

PRUEBAS UNITARIAS (CAJA NEGRA)

Clase Coordinador

Caso de prueba	Resultado esperado	Resultado obtenido
Método: action ()		
Obtener parámetros correctos del objeto Interaction	Se obtiene los valores correctos de todos los parámetros.	Se obtiene los valores correctos de todos los parámetros.
Obtener valores nulos de los parámetros del objeto Interaction	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes y se visualiza un aviso indicando que “Se produjo un error en la transacción”.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes y se visualiza un aviso indicando que “Se produjo un error en la transacción”.
Almacenar el archivo que detalla la sugerencia almacenada en la base de datos.	Es almacenado el archivo e imagen (en caso de tenerla) en el directorio adecuado en el servidor.	Es almacenado el archivo e imagen (en caso de tenerla) en el directorio adecuado en el servidor.
No se efectúo el almacenamiento del archivo que detalla la sugerencia almacenada en la base de datos.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes y se visualiza un mensaje de que se produjo un error.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes y se visualiza un mensaje de que se produjo un error.
Método: enviarMensaje ()		
Envío correcto del mensaje	Se ha enviado al agente adecuado el mensaje correspondiente.	Se ha enviado al agente adecuado el mensaje correspondiente.
Envío incorrecto del mensaje	No se envía el mensaje y se visualiza el mensaje correspondiente al error.	No se envía el mensaje y se visualiza el mensaje correspondiente al error.
Método: recibirMensaje ()		
Obtener los datos correctos del mensaje enviado.	Se obtiene los datos correctos del mensaje enviado.	Se obtiene los datos correctos del mensaje enviado.
Obtener valores nulos en los datos del mensaje enviado.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes y se visualiza un aviso indicando que “Se produjo un error en la transacción”.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes y se visualiza un aviso indicando que “Se produjo un error en la transacción”.
Método: takeDown ()		
takeDown ()	Se ha Inhabilitado el canal de comunicación objeto-agente.	Se ha Inhabilitado el canal de comunicación objeto-agente.

Tabla D.1: Resultados de la prueba unitaria de la clase Coordinador.

Clase Asesor

Caso de prueba	Resultado esperado	Resultado obtenido
Método: recibirMensajeCoordinador ()		
Obtener los datos correctos del mensaje enviado.	Se obtiene los datos correctos del mensaje enviado.	Se obtiene los datos correctos del mensaje enviado.
Obtener valores nulos en los datos del mensaje enviado.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes.
Método: recibirMensajeBuscador ()		
Obtener los datos correctos del mensaje enviado.	Obtener los datos correctos del mensaje enviado.	Obtener los datos correctos del mensaje enviado.
Obtener valores nulos en los datos del mensaje enviado.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes.
Método: enviarMensaje ()		
Envío correcto del mensaje	Se ha enviado al agente adecuado el mensaje correspondiente.	Se ha enviado al agente adecuado el mensaje correspondiente.
Envío incorrecto del mensaje	No se envía el mensaje y se visualiza el mensaje correspondiente al error.	No se envía el mensaje y se visualiza el mensaje correspondiente al error.
Método: takeDown ()		
takeDown ()	Se ha Inhabilitado el canal de comunicación objeto-agente.	Se ha Inhabilitado el canal de comunicación objeto-agente.

Tabla D.2: Resultados de la prueba unitaria de la clase Asesor.

Clase Supervisor

Caso de prueba	Resultado esperado	Resultado obtenido
Método: recibirMensajeCoordinador ()		
Obtener los datos correctos del mensaje enviado.	Se obtiene los datos correctos del mensaje enviado.	Se obtiene los datos correctos del mensaje enviado.
Obtener valores nulos en los datos del mensaje enviado.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes.
Método: recibirMensajeBuscador ()		
Obtener los datos correctos del mensaje enviado.	Obtener los datos correctos del mensaje enviado.	Obtener los datos correctos del mensaje enviado.
Obtener valores nulos en los datos del mensaje enviado.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes.

Método: enviarMensaje ()		
Envío correcto del mensaje	Se ha enviado al agente adecuado el mensaje correspondiente.	Se ha enviado al agente adecuado el mensaje correspondiente.
Envío incorrecto del mensaje	No se envía el mensaje y se visualiza el mensaje correspondiente al error.	No se envía el mensaje y se visualiza el mensaje correspondiente al error.
Método: takeDown ()		
takeDown ()	Se ha Inhabilitado el canal de comunicación objeto-agente.	Se ha Inhabilitado el canal de comunicación objeto-agente.

Tabla D.3: Resultados de la prueba unitaria de la clase Supervisor.

Clase Buscador

Caso de prueba	Resultado esperado	Resultado obtenido
Método: recibirMensaje ()		
Obtener los datos correctos del mensaje enviado.	Se obtiene los datos correctos del mensaje enviado.	Se obtiene los datos correctos del mensaje enviado.
Obtener valores nulos en los datos del mensaje enviado.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes.	El agente obtiene un comportamiento bloqueado hasta que le lleguen más mensajes.
Método: enviarMensaje ()		
Envío correcto del mensaje	Se ha enviado al agente adecuado el mensaje correspondiente.	Se ha enviado al agente adecuado el mensaje correspondiente.
Envío incorrecto del mensaje	No se envía el mensaje y se visualiza el mensaje correspondiente al error.	No se envía el mensaje y se visualiza el mensaje correspondiente al error.
Método: limpiarDatosRespuesta ()		
limpiarDatosRespuesta ()	Se han asignado los valores iniciales por defecto al arreglo de datos respuesta.	Se han asignado los valores iniciales por defecto al arreglo de datos respuesta.
Método: infoProceso(String rol, String dato)		
infoProceso(“RAPE”, “Objetivos”): datos correctos.	Se obtiene la información correcta.	Se obtiene la información correcta.
infoProceso(“APE”, “Metas”): datos incorrectos.	No se obtiene información.	No se obtiene información.
Método: infoSugerencia(String proceso, String tarea)		
infoSugerencia(“OPE.1”, “A1.3.”): datos correctos.	Se obtiene la información correcta.	Se obtiene la información correcta.
infoSugerencia(“SPE.1”, “A1.50.”): datos incorrectos.	No se obtiene información.	No se obtiene información.
Método: guardarSugerencia(String s_proceso, String s_tarea, String s_descripcionSugerencia, String s_detalle)		
guardarSugerencia (“OPE.1”, “A1.3.”, “Guía para la formación	Se almacena en la base de datos.	Se almacena en la base de datos.

de equipos efectivos", "BaseDatosSoporte/detalles/detalle1.htm"): datos correctos.		
guardarSugerencia ("OPI.1", "A30.3.", "Guía para la formación de equipos efectivos", "BaseDatosSoporte/detalles/detalle1.htm"): datos incorrectos.	No se almacena en la base de datos y se visualiza el mensaje correspondiente al error.	No se almacena en la base de datos y se visualiza el mensaje correspondiente al error.
Método: infoTareas (String proceso, String rol, String proyecto)		
infoTareas ("OPE.1", "RAPE", "3"): datos correctos.	Se obtiene la información correcta.	Se obtiene la información correcta.
infoTareas ("OVE.1", "REPE", "0"): datos incorrectos.	No se obtiene información.	No se obtiene información.
Método: takeDown ()		
takeDown ()	Se ha Inhabilitado el canal de comunicación objeto-agente.	Se ha Inhabilitado el canal de comunicación objeto-agente.

Tabla D.4: Resultados de la prueba unitaria de la clase Buscador.

Clase DAOGeneral

Caso de prueba	Resultado esperado	Resultado obtenido
Método: cargarDriver ()		
cargarDriver ()	Se obtiene el valor true indicando que se ha cargado el driver.	Se obtiene el valor true.
Método: obtenerConexion ()		
obtenerConexion ()	Conexión establecida con la BD.	Conexión establecida con la BD.
Método: cerrarConexion(Connection con)		
cerrarConexion(con)	Conexión cerrada con la BD.	Conexión cerrada con la BD.

Tabla D.5: Resultados de la prueba unitaria de la clase DAOGeneral.

Clase DAOSugerenciaSMA

Caso de prueba	Resultado esperado	Resultado obtenido
Método: consultar(String condicion)		
consultar("Sugerencia.idProceso = '1')AND(Sugerencia.identificadorTarea = 'A1.4')AND (Tarea.idProceso = '1') AND (Tarea.identificadorTarea = 'A1.4')"): datos correctos	Se obtiene la información correcta.	Se obtiene la información correcta.
consultar("Sugerencia.idProceso = '1')AND(Sugerencia.identificadorTarea = 'B1.4')AND (Tarea.idProceso = '1') AND (Tarea.identificadorTarea = 'A1.4')"): datos incorrectos	No se obtiene información.	No se obtiene información.
Método: insertar(Bean bean)		
insertar (VOSugerenciaSMA): datos correctos	Se inserta el registro en la base de datos y	Se inserta el registro en la base de datos y

	se devuelve el número de filas afectadas.	se devuelve el número de filas afectadas.
insertar (VOSugerenciaSMA): datos incorrectos	No se inserta el registro en la base de datos y se devuelve 0 número de filas afectadas.	No se inserta el registro en la base de datos y se devuelve 0 número de filas afectadas.

Tabla D.6: Resultados de la prueba unitaria de la clase DAOSugerenciaSMA.

Cabe señalar que las demás clases DAO no se listaron debido a que la ejecución de las pruebas es prácticamente la misma, sólo con la variación de atributos y que sólo implementan el método *consultar().* En cuanto a las clases de los ValueObject (beans), se comprobó que los métodos gets y sets se realizan correctamente.

Clase DirectoryServlet

Caso de prueba	Resultado esperado	Resultado obtenido
Método: SyncInit ()		
SyncInit ()	Se ha creado el contenedor principal, secundario y los agentes Coordinador, Supervisor, Asesor y Buscador.	Se ha creado el contenedor principal, secundario y los agentes Coordinador, Supervisor, Asesor y Buscador.
Método: destroy()		
destroy()	Se ha terminado el contenedor creado para los agentes, el contenedor principal y el contenedor JADE.	Se ha terminado el contenedor creado para los agentes, el contenedor principal y el contenedor JADE.

Tabla D.7: Resultados de la prueba unitaria de la clase DirectoryServlet.

PRUEBAS DEL SISTEMA

Caso de uso: 1.1 Consultar información de control

Precondiciones: haber entrado al sistema con usuario y contraseña correctos.

Caso de prueba	Resultado obtenido
Presionar en la opción "Información de control"	El sistema muestra en la pantalla las tareas realizadas por el usuario y sus estados.
No existe información.	El sistema muestra en la pantalla un mensaje indicando que “No se encontró información”.
No se obtienen los datos sobre el rol, proyecto o proceso.	El sistema despliega un aviso explicando que “Se produjo un error en la transacción”.

Tabla D.8: Resultados de los casos de prueba del sistema del caso de uso 1.1 Consultar información de control.

Caso de uso: 1.2 Consultar información del proceso

Precondiciones: haber entrado al sistema con usuario y contraseña correctos.

Caso de prueba	Resultado obtenido
Presionar en la opción "Información del proceso"	El sistema muestra en la pantalla una lista desplegable con los temas que son parte de la descripción del proceso correspondiente a su rol.
No se obtienen los datos sobre el rol o proceso.	El sistema despliega un aviso explicando que “Se produjo un error en la transacción”.
Seleccionar el tema a consultar sobre el proceso y presionar el botón "Consultar"	El sistema muestra en la pantalla la información referente al tema elegido del proceso que le corresponde en base a su rol.
No existe información.	El sistema muestra en la pantalla un mensaje indicando que “No se encontró información”.

Tabla D.9: Resultados de los casos de prueba del sistema del caso de uso 1.2 Consultar información del proceso.

Caso de uso: 1.3 Consultar información de soporte

Precondiciones: haber entrado al sistema con usuario y contraseña correctos.

Caso de prueba	Resultado obtenido
Presionar en la opción "Información de soporte"	El sistema muestra en la pantalla una lista desplegable con las tareas que son realizadas por el usuario en el sistema de acuerdo a su rol.
No se obtienen los datos sobre el rol o proceso.	El sistema despliega un aviso explicando que “Se produjo un error en la transacción”.
Seleccionar la tarea cuyas sugerencias se desean consultar y presionar el botón "Consultar"	El sistema muestra en la pantalla una lista con las sugerencias registradas y que son correspondientes a la tarea que fue seleccionada.
No existe información.	El sistema muestra en la pantalla un mensaje indicando que “No se encontró información”.
Presionar en la opción " Ver detalle" de alguna sugerencia listada.	El sistema muestra en la pantalla la información correspondiente al detalle de la sugerencia elegida.

Tabla D.10: Resultados de los casos de prueba del sistema del caso de uso 1.3 Consultar información de soporte.

Caso de uso: 2. Agregar información de soporte

Precondiciones: haber entrado al sistema con usuario y contraseña correctos.

Caso de prueba	Resultado obtenido
Presionar en la opción "Agregar sugerencia"	El sistema muestra en la pantalla una lista desplegable con las tareas que son realizadas por el usuario en el sistema de acuerdo a su rol.

Caso de prueba	Resultado obtenido
No se obtienen los datos sobre el rol o proceso.	El sistema despliega un aviso explicando que “Se produjo un error en la transacción”.
Seleccionar la tarea a la que pertenecerá la sugerencia y presionar el botón "Guardar” sin introducir el texto correspondiente a la descripción de la sugerencia.	El sistema muestra en la pantalla un mensaje indicando que “El campo referente a la descripción de la sugerencia es un dato necesario.”.
Seleccionar la tarea a la que pertenecerá la sugerencia y presionar el botón "Guardar” sin haber examinado un archivo e imagen validos.	El sistema guarda el registro en la base de datos sin detalle para la sugerencia, además no guarda ningún archivo de detalle ni imagen y muestra en la pantalla un mensaje indicando que “La sugerencia fue almacenada exitosamente”.
Seleccionar la tarea a la que pertenecerá la sugerencia y presionar el botón "Guardar” con un archivo e imagen validos.	El sistema guarda el registro en la base de datos con detalle para la sugerencia, además guarda el archivo de detalle e imagen, y muestra en la pantalla un mensaje indicando que “La sugerencia fue almacenada exitosamente”.
Seleccionar la tarea a la que pertenecerá la sugerencia y presionar el botón "Guardar” sin haber examinado un archivo html pero si se tiene una imagen válida.	El sistema no guarda el registro en la base de datos y muestra en la pantalla un mensaje indicando que “No puede subir una imagen sino especifica el archivo correspondiente”.
Seleccionar la tarea a la que pertenecerá la sugerencia y presionar el botón "Guardar” sin haber examinado una imagen pero si se tiene un archivo válido.	El sistema guarda el registro en la base de datos con detalle para la sugerencia pero no almacena ninguna imagen relacionada al archivo del detalle y muestra en la pantalla un mensaje indicando que “La sugerencia fue almacenada exitosamente”.
Seleccionar la tarea a la que pertenecerá la sugerencia y presionar el botón "Guardar” con un archivo válido y una imagen inválida.	El sistema no guarda el registro en la base de datos y muestra en la pantalla un mensaje indicando que “Comprueba el tipo de la imagen a subir. Sólo se pueden subir imágenes con extensiones: .gif, .jpg, .jpeg, .bmp, .png”.
Seleccionar la tarea a la que pertenecerá la sugerencia y presionar el botón "Guardar” con una imagen válida y un archivo inválido.	El sistema no guarda el registro en la base de datos y muestra en la pantalla un mensaje indicando que “Comprueba el tipo del archivo a subir. Sólo se pueden subir archivos con extensiones:: .html, .htm”

Tabla D.11: Resultados de los casos de prueba del sistema del caso de uso 2. Agregar información de soporte.

Apéndice E

Código para la integración

Esta sección contiene el código necesario para la integración del Sistema Multi-Agente con HeAP MoProSoft. Para ello, se describe el código anexo al archivo web.xml y el código correspondiente al menú definido para hacer uso del Sistema Multi-Agente.

CÓDIGO ANEXO AL ARCHIVO WEB.XML PARA EL DIRECTORYSERVLET

En la parte de *Standard Action Servlet Configuration* anexar el siguiente código:

```
<!-- Standard Action Servlet Configuration -->

<!-- ***************** Servlet para el SMA ************** -->
<servlet>
    <servlet-name>DirectoryServlet</servlet-name>
    <display-name>DirectoryServlet</display-name>
    <description> servlet para ligar al  agente Coordinador con
    la interfaz gráfica de la aplicación Web HeAP MoProSoft
    </description>
    <servlet-class> sma.ServletAgente.DirectoryServlet</servlet-
class>
  </servlet>

<!-- ************************ fin ********************** -->
```

Además, en la parte de *Standard Action Servlet Mapping* anexar el siguiente código:

```
<!-- Standard Action Servlet Mapping -->

<!-- ********* Agregar el servlet para el SMA ************** -->
<servlet-mapping>
    <servlet-name>DirectoryServlet</servlet-name>
    <url-pattern>/DirectoryServlet</url-pattern>
  </servlet-mapping>

<!-- ************************ fin ************************** -->
```

CÓDIGO PARA EL MENÚ DEL SISTEMA MULTI-AGENTE

Anexar el siguiente código de JavaScript.

```
<link href="sma/css/sma.css" rel="stylesheet" type="text/css">

<script language="javascript" type="text/javascript">

function infoControl()
{
 var parametros;
 var info = document.id.value;      //identificador del proyecto
 var rol= document.rol.value;
 var proceso = document.proceso.value;
 parametros = "caso=1&proceso="+proceso+"&rol="+rol+"&info="+info;
 url = 'DirectoryServlet?' + parametros;
 window.open(url,'','height=600,width=400,scrollbars=yes').moveTo
(850,120);
}

function infoProceso(){
 var parametros;
 var info = "NA";
 var rol= document.rol.value;
 var proceso = document.proceso.value;
 parametros = "caso=2&proceso="+proceso+"&rol="+rol+"&info="+info;
 url = 'sma/sma2.jsp?' + parametros;
 window.open(url,'','height=600,width=400,scrollbars=yes').moveTo
(850,120);
}

function infoSoporte(){
 var parametros;
 var info = "NA";
 var rol= document.rol.value;
 var proceso = document.proceso.value;
 parametros = "caso=2&proceso="+proceso+"&rol="+rol+"&info="+info;
 url = 'sma/sma3.jsp?' + parametros;
 window.open(url,'','height=600,width=400,scrollbars=yes').moveTo
(850,120);
}

function agregarSugerencia(){
 var parametros;
 var info = "NA";
 var rol= document.rol.value;
 var proceso = document.proceso.value;
 parametros = "caso=2&proceso="+proceso+"&rol="+rol+"&info="+info;
 url = 'sma/sma4.jsp?' + parametros;
 window.open(url,'','height=600,width=400,scrollbars=yes').moveTo
(850,120);
}
</script>
```

Anexar el siguiente código html.

```
<table width="178" border="0">
 <%
   String proceso = session.getAttribute("proceso");
   String rol = session.getAttribute("rol");
   String proyecto = session.getAttribute("proyecto");
 %>
<input type="hidden" name="id" value="<%= proyecto %>" >
<input type="hidden" name="rol" value="<%= rol %>" >
<input type="hidden" name="proceso" value="<%= proceso %>">
 <tr>
    <td align="center">
     <table width="178" height="24" border="0" align="center"
      cellpadding="0" cellspacing="0"
      background="sma/imagenes/encabezado.GIF">
      <tr>
        <td width="178" height="25" valign="middle">
         <div align="left"><font size="1" face="Verdana">
         <strong> 
         <img src="sma/imagenes/generic.gif" width="18" height="18"
         align="absbottom">
         </strong>
         </font></div>
        </td>
      </tr>
     </table>
     <table width="178" height="25" border="1" cellspacing="2">
        <tr><td><font size="1" face="Verdana">
           <a href="javascript:infoProceso();">
            Informaci&oacute;n del proceso
           </a></font></td></tr>
        <tr><td><font size="1" face="Verdana">
           <a href="javascript:infoControl();">
            Informaci&oacute;n de control
           </a></font></td></tr>
        <tr><td><font size="1" face="Verdana">
           <a href="javascript:infoSoporte();">
            Informaci&oacute;n de soporte
           </a></font></td></tr>
        <tr><td><font size="1" face="Verdana">
           <a href="javascript:agregarSugerencia();">
            Agregar sugerencia
           </a></font></td></tr>
      </table>
    </td>
  </tr>
</table>
```

GLOSARIO

ACL	*Agent Communication Language.* Lenguaje de Comunicación de Agentes.
AMCIS	Asociación Mexicana para la Calidad de Ingeniería de Software.
AMS	*Agent Management System.* Sistema de Administración de Agentes.
AOSE	*Agent Oriented Software Engineering.* Ingeniería de Software Orientada a Agentes.
API	*Application Programming Interface.* Interfaz de Programación de Aplicaciones.
AUML	*Agents Unified Modelling Language*. Lenguaje Unificado de Modelado de Agentes.
BDI	*Belief-Desire-Intention*. Arquitectura de comportamiento de agentes basada en actitudes mentales como creencias, deseos e intenciones.
CMAS	*Cooperative Multiagent Systems.* Sistemas Cooperativos Multi-Agentes.
CMM	*Capability Maturity Model*. Modelo de Madurez de Capacidades.
CMMI	*Capability Maturity Model Integration*. Modelo de Madurez de Capacidades Integrado.
CUBL	Concurrent Unit Based Language.
DAO	*Data Access Object*. Objeto de Acceso a Datos (patrón de diseño).
DF	*Directory Facilitator.* Facilitador de Directorios. Agente de Servicio de Directorio en JADE.
EvalProSoft	Método de Evaluación de procesos para la industria de Software.
FIPA	*Foundation for Intelligent Physical Agents.* Fundación para Agentes Inteligentes Físicos.
GOPRR	Graph, Object, Property, Relationship, and Role.
GRASIA	Grupo de Agentes Software: Ingenieria y Aplicaciones.
GUI	*Graphical User Interface.* Interfaces Gráfica de Usuario.

HeAP MoProSoft	Herramienta para la Administración de Proyectos para MoProSoft.
IDE	Integrated Development Environment. Entorno Integrado de Desarrollo.
IDK	INGENIAS Development Kit
IIOP	*Internet Inter-ORB Protocol*. protocolo de Inter-ORBE de Internet
IMTP	*Internal Message Transport Protocol*. Protocolo de transporte de. Mensaje.
INGENIAS	Es el nombre de una metodología de desarrollo de Sistema Multi-Agente que proporciona un conjunto de métodos y herramientas para desarrollar dichos sistemas.
J2EE	*Java 2 Platform, Enterprise Edition*. Plataforma Java 2, Edición de Empresa.
J2SE	*Java 2 Platform, Standard Edition*. Plataforma Java 2, Edición Estándar.
JADE	*Java Agent Development Framework*. Marco de trabajo en Java para el Desarrollo de Agentes.
JDBC	*Java Database Connectivity*. Conectividad Java para Bases de Datos.
JSP	*Java Server Page*. Página Java de Servidor.
JVM	*Java Virtual Machine*. Máquina Virtual de Java.
KQML	*Knowledge Query and Manipulation Language.* Lenguaje ampliamente utilizado para comunicar agentes.
LEAP	*Lightweight Extensible Agent Platform*. Lenguaje que permite la ejecución de FIPA-compliant agentes en PDAs y teléfonos móviles.
LGPL	Lesser General Public License.
Lisp	Lenguaje de programación funcional. Su nombre deriva del término "procesamiento de listas" en inglés ("List Processing").
MESSAGE	*Methodology for Engineering Systems of Software AGent.* Metodología para la Ingeniería de Sistemas de Agentes de Software.
MoProSoft	Modelo de Procesos para la Industria de Software.
MTP	*Message Transport Protocols*. Protocolo de transporte de Mensaje.
Multipart/form-data	Es un tipo tipo de codificacion que tendrán los datos de las peticiones POST.

MVC	Modelo, Vista, Controlador (patrón de arquitectura).
ODP	*Open Distributed Processing*. Procesamiento distribuido abierto.
PDA	*Personal Digital Assistant*. Asistente personal digital.
Perl	*Practical Extraction and Report Language*. Lenguaje Práctico de Extracción e Informe.
PMBoK	*Project Management Body of Knowledge*. Cuerpo de Conocimiento para la Administración de Proyectos.
PostgreSQL	Servidor de base de datos relacional libre, liberado bajo la licencia BSD.
Prolog	Lenguaje de programación concebido para escribir programas de Inteligencia Artificial. Creado en 1972, las tareas se expresan describiendo los objetos que se necesitan y las relaciones lógicas entre ellos.
PROSOFT	Programa para el Desarrollo de la Industria de Software.
PSP	*Personal Software Process*. Proceso Personal de Software
PU	Proceso Unificado de Desarrollo de Software.
PyMES	Pequeñas y medianas empresas.
RAPE	Responsable de Administración del Proyecto Específico.
RETSINA	Reusable Environment for Task-Structured Intelligent Network Agents.
RGPY	Responsable de Gestión de Proyectos.
RMA	*Remote Monitoring Agent*. Agente de Monitoreo Remoto.
RMI	*Remote Method Invocation*. Invocación Remota de Métodos.
SDL	*Specification and Description Language*. Lenguaje de especificación y descripción.
SL	*Semantic Language*. Lenguaje semántico propuesto por la FIPA.
SL0	*Semantic Language Subset 0*. Subconjunto mínimo del lenguaje SL.
SMA	Sistema Multi-Agente.
SMAS	*Self-Interested Multiagent Systems*. Sistemas Multi-Agentes con intereses propios.

SWEBoK	*Software Engineering Body of Knowledge*. Cuerpo de Conocimiento para la Ingeniería de Software.
Tcl	*Tool Command Language*. Lenguaje de herramientas de comando.
Tomcat	También llamado Jakarta Tomcat o Apache Tomcat, funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Sun Microsystems.
TSP	*Team Software Process*. Proceso de Software en Equipo.
UML	*Unified Modeling Language.* Lenguaje Unificado de Modelado.
XML	*eXtensible Markup Language*. Lenguaje de Marcado Extensible.