UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

UN MODELO GENÉRICO PARA LA INTEGRACIÓN DE APLICACIONES HETEROGÉNEAS EN LA AUTOMATIZACIÓN DE PROCESOS CONTINUOS BASADO EN OBJETOS/REGLAS DE NEGOCIOS Y EN AGENTES DE SOFTWARE

Autor: Ing. Malinda Del V. Coa Ravelo

Tutor: Dra. Isabel Besembel

PROYECTO DE GRADO PRESENTADO ANTE LA ILUSTRE
UNIVERSIDAD DE LOS ANDES
COMO REQUISITO FINAL PARA OPTAR AL TÍTULO DE
MAGISTER SCIENTIAE EN COMPUTACIÓN

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
POSTGRADO EN COMPUTACIÓN
DICIEMBRE DEL 2002

A mi amado "chiquitico" , Carlos Junior BeBé.

Tabla de Contenidos

Indice de Tablas

Indice de Figuras

Agradecimientos

A Dios Todopoderoso, por darme la vida que es tan hermosa.

Al Divino Niño, por iluminar mi camino y protegerme junto con los míos.

A mi tutora, Dra. Isabel Besembel por su gran ayuda, asesoría e infinita paciencia. Muchísimas Gracias.

A mi hermoso hijo Carlos Junior, por llenar de felicidad mi existencia. Eres mi motivo de lucha e inspiración. Te amo, BeBé.

A mi esposo Carlos Junior, por brindarme su apoyo y llamarme la atención cuando en las noches deseaba dormir y debía trabajar. Refunfuñando me levantaba, pero no sabes cuánto me ayudaron tus regaños.

A Jakeline Peña, por cuidar de mi más preciado tesoro. Durante mis ausencias, mi hijo no pudo quedar en mejores manos.

A todos aquellos que, por un motivo u otro, merezcan estar en esta página y a quienes no nombro por temor a pecar de omisión. A todos ustedes, miles de gracias.

Malinda Coa

Resumen

En la evolución de la automatización de los sistemas de producción, las empresas han desarrollado o adquirido aplicaciones de software especializadas para el soporte del manejo de la información de los diferentes procesos productivos y de gestión. Puesto que muchas de las arquitecturas de aplicaciones de software desarrolladas han sido pobremente planificadas a los efectos de una futura integración, las empresas cuentan con una infraestructura de aplicaciones, casi todas heterogéneo, que aislan la información necesaria para el proceso de toma de decisiones.

Actualmente, los Agentes de Software constituyen una opción en las soluciones de integración (incluyendo las de aplicaciones de software), pues ellos exhiben características de sistemas robustos y flexibles, que pueden decidir sobre por sí mismos qué hacer para lograr los objetivos para los que fueron diseñados.

El objetivo de este trabajo de investigación es el de proponer una arquitectura que permite que las aplicaciones de un Complejo de Producción Continua (CPC), aún siendo heterogéneas, "conversen" entre sí y puedan compartir información. Este objetivo se ha logrado mediante una arquitectura basada en Agentes, quienes representan las Unidades de Producción que conforman un CPC. Dichos agentes, son los responsables de la integración apoyados en las Reglas de Negocios, que reflejan la lógica del negocio y en los Objetos de Negocios, quienes se utilizan para ejecutar dicha lógica.

La arquitectura de Agente Integrador corresponde a la de un agente híbrido, pues él refleja características reactivas -ante los cambios de su entorno- y deliberativas -razonando a través de su máquina de inferencia-; como una primera aproximación, el Agente Integrador es modelado como un Sistema Experto, dada la compatibilidad de características entre ambos sistemas.

Con el fin de probar las teorías y arquitecturas propuestas, se implantó un prototipo de naturaleza genérica, que demuestra la posibilidad de construcción de estos sistemas integradores como parte del proceso de Automatización.

Capítulo 1

El Problema

Las secciones del presente capítulo abarcan el planteamiento y formulación del problema, los objetivos y justificación de investigación, así como la organización de esta tesis.

1.1. Planteamiento y Formulación del Problema

Antes de describir el trabajo realizado, se hace necesario ubicar en un escenario el problema estudiado. Esta sección pretende lograr este cometido, emplazando el problema de interés en el contexto de la automatización e integración de los sistemas de producción continuos.

1.1.1. Los Sistemas de Producción

Los modelos de sistemas de producción en industrias son aquellos que describen el comportamiento de un proceso de producción, donde están involucrados parámetros y actividades, como son: la cantidad a producir de un producto determinado, las características del mismo, el tiempo en producirlo, etc. Esto forma parte de un sistema más complejo que integra las actividades financieras y de planificación de la industria. Entre los procesos de producción más comunes, actualmente se encuentran: los procesos de manufactura, procesamiento en lotes y producción continua.

Los Procesos de Manufactura

Los procesos de manufactura son aquellos procesos de producción discretos e independientes, donde la salida de un producto está asociada con la finalización de la tarea [46]. Las funciones principales de estos procesos se asocian a la maquinación y

ensamblaje de piezas. La maquinación forma productos intermedios que luego serán ensamblados. Por ello, no existe una mezcla de productos como en el caso de la industria química, petrolera o de alimentos.

A través de los años, el proceso de manufactura ha cambiado de los sistemas donde los operarios manejaban las máquinas y/o herramientas de manera individual a secuencias de operaciones, donde algunas máquinas trabajan de manera automática, hasta llegar a los modernos métodos de producción, donde mediante computadora se generan secuencias de operaciones, permitiendo la reconfiguración de las plantas de manera automática, para lograr reducir los tiempos de producción, con el reuso de los sistemas existentes [11].

Los Procesos de Producción en Lotes

En el procesamiento por lotes, un proceso recibe un conjunto de materia prima, la cual es transformada en un nuevo producto mediante mezclas y aplicación de energía. Una vez terminado el proceso, éste arranca nuevamente para repetir el mismo o realizar uno nuevo. Ejemplo de este tipo de procesos lo constituye la industria siderúrgica, la industria de alimentos y en gran parte de la industria química [11].

Los Procesos de Producción Continua

Como procesos continuos se conocen a los procesos de transformación o producción continua, es decir, son los procesos de transformación o producción que trabajan de manera permanente como son las plantas generadoras de energía eléctrica, la producción de petróleo, gas, la refinación de crudo, etc. En estos casos, los productos resultantes o que se están transformando continuamente alimentan otras plantas [11].

La mayoría de estos sistemas son considerados sistemas complejos, por el tipo de transformaciones que se dan y por la interrelación entre los procesos que están presentes en el sistema.

Esos procesos se han visto beneficiados por el desarrollo de la tecnología electrónica y las comunicaciones, que permiten la incorporación de técnicas de control que anteriormente no eran posibles. Un ejemplo de ello, es la medición de variables a través de sensores.

1.1.2. Automatización de los Sistemas de Producción

Se considera un sistema de producción automatizado aquel donde los procesos de producción son principalmente coordinados y controlados mediante sistemas automáticos [11].

En los procesos de manufactura, el modelo de automatización mayormente usado es el CIM (*Computer Integrated Manufacturing*) [15]. CIM envuelve los aspectos de control directo (manufactura y ensamblaje), planificación de la producción, diseño de piezas, así como también los aspectos administrativos y gerenciales de la planta.

En los procesos continuos, la Pirámide de Automatización, es la propuesta para la implantación de un sistema automatizado [47]. Este modelo es similar al modelo CIM para ambientes de manufactura.

El Modelo CIM

Este modelo, propuesto por la ISO [15], considera a la empresa en cinco niveles que se utilizan para la integración de los procesos de producción:

Fábrica. Constituye el nivel más alto e incluye los procesos de planificación, gerencia de la producción, incluyendo planificación a largo plazo.

Unidades de producción. Estas unidades manejan la coordinación de recursos y tareas necesarias en cada unidad de producción. Una unidad de producción se considera dedicada a un producto o línea de producto específico. Agrupan varias celdas de ensamblaje y distribuyen y coordinan las actividades entre las diferentes celdas.

Celdas de ensamblaje. Cada una de ellas contienen los sistemas de control para el secuenciamiento de las tareas asignadas a la misma.

Estaciones de trabajo. Están constituidas por uno o más equipos que realizan una actividad única. Ellas reciben órdenes de la coordinación de la celda de ensamblaje y ejecutan tareas o secuencias de operaciones asignadas a la celda de ensamblaje.

Equipos. Este es el nivel más bajo de la arquitectura CIM y consiste de los recursos para controladores individuales, tales como robots y los equipos sensores y actuadores asociados al mismo.

La Pirámide de Automatización

Por otro lado, ISO propone un modelo similar al CIM pero adaptado a la automatización de procesos continuos que se denomina Pirámide de Automatización [47]. Este modelo presenta una jerarquía de cinco niveles (ver figura 1.1), donde cada nivel se caracteriza por un tipo de información y procesamiento diferente. La integración de un proceso automatizado incluye la comunicación interna en cada nivel, así como la comunicación entre niveles, con el fin de lograr sistemas que permitan ejecutar las diferentes tareas de control existentes en una empresa [13].

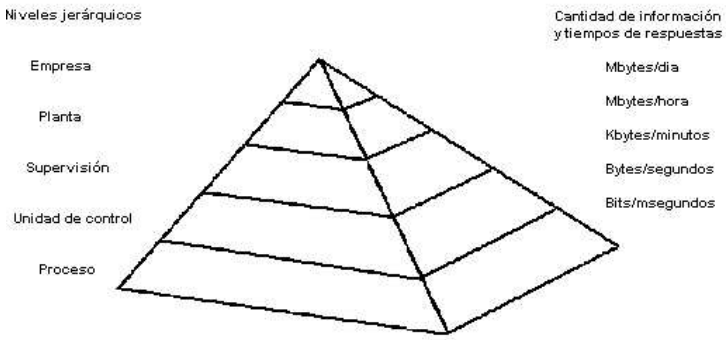


Figura 1.1: Pirámide de Automatización de la ISO

Los cinco niveles encontrados en la Pirámide de Automatización son:

Empresa. Al igual que en el nivel más alto del modelo CIM, a nivel de empresa, el sistema comprende los problemas de gerencia de producción, fijación de estrategias y niveles de producción, que están asociados a las políticas globales de la empresa, el factor financiero y el mercado. A este nivel se coordinan las actividades entre las diferentes plantas o unidades completas de producción. La cantidad de información que maneja este nivel es muy grande y proviene de las plantas y de los factores externos a la empresa.

Planta. Es la responsable del logro de las metas de producción fijadas, mediante el manejo y coordinación de los recursos (económicos, humanos y equipos) para la ejecución de las diferentes actividades. En este nivel se establecen los parámetros y criterios de producción, así como la optimización y planificación de las tareas de producción.

Supervisión. Coordina las diferentes unidades de procesamiento a través de la parametrización de los controladores que realizan los operadores de planta encargados de controlar los procesos.

Unidad de control. Los elementos que realizan el control de procesos, trabajan de acuerdo a una parametrización recibida desde el nivel supervisorio. Estos sistemas actúan de manera automática manteniendo el control regulatorio y/o secuencias de los procesos productivos, mediante una realimentación del proceso.

Nivel de proceso. Se refiere a los instrumentos y equipos que están en contacto directo con el proceso. Por lo general son dispositivos electromecánicos con posibilidades de comunicación.

El Enfoque Jerárquico: Unidades de Producción

Enmarcado en el proyecto [27], E. Chacón et al. en [12] presentan un modelo jerárquico para la automatización de procesos continuos. En este modelo, un Complejo de Producción Continua (CPC) puede verse como un conjunto de Unidades de Producción (UP) que comparten recursos. Un CPC está compuesto de varias UPs, que controlan internamente su comportamiento para lograr sus metas. El sistema completo logra sus objetivos, asignando metas simples a cada UP. Si una UP tiene varios subsistemas, el óptimo de esa UP se establece de la misma manera jerárquica. La UP es la unidad central de transformación de la arquitectura de un CPC y también puede ser interpretada como una composición de varias UPs. Una empresa y un CPC pueden verse como UPs.

Entonces, la arquitectura organizacional para el CPC puede estructurarse en capas, similar al modelo CIM. Esta arquitectura, que se muestra en la figura 1.2, comprende cuatro niveles:

Empresa. A este nivel se ejecutan las funciones de planificación estratégica y el manejo centralizado de la empresa, donde se toman decisiones acerca de las políticas de Finanzas, Recursos Humanos, Mercadeo, entre otras. Estas actividades implican el establecimiento de planes a largo plazo y políticas de la empresa. El nivel de empresa coordina las actividades de producción que ejecuta un Complejo de

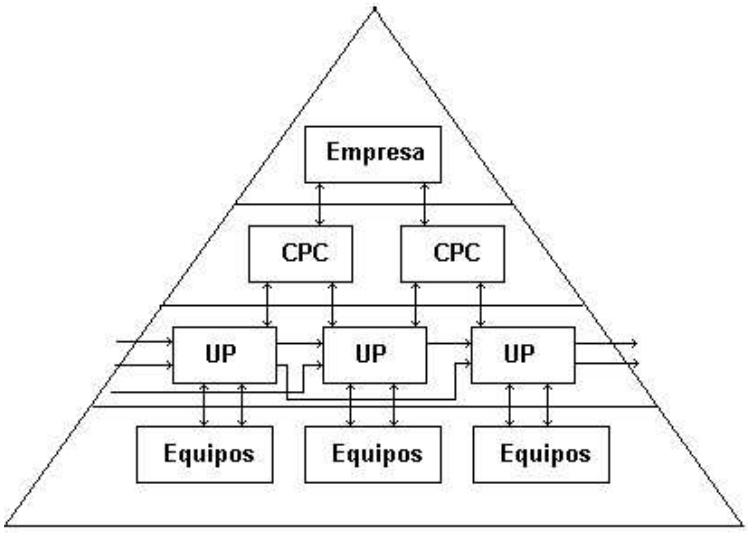


Figura 1.2: Arquitectura Organizacional de la Empresa

Producción (CP). En algunos casos una planta -en un CP- y la asociación de varias plantas conforman una topología que permite asegurar las metas de la empresa.

Complejo de Producción. Este nivel está orientado a coordinar las UPs que pertenecen a un CP, con el propósito de optimizar los recursos y programar las operaciones. La gerencia del CP establece las cantidades a producir, emite las órdenes de producción, ejecuta funciones de contabilidad y establece los requerimientos de material. Un CP puede tener varias UPs que produzcan el mismo tipo de producto o formar una línea de producción. Los sistemas de almacenamiento intermedios y finales son parte del CP y pueden considerarse, a su vez, como una UP.

Unidad de Producción. A este nivel, la UP ejecuta el proceso de producción de una clase de productos. Una UP puede considerarse semi-autónoma, cuando ejecuta las funciones de regulación y control y "conoce" cómo lograr sus objetivos y el estado de su producción. La UP puede estar compuesta de varios equipos, que ejecutan los procesos de transformación. Estas transformaciones pueden ser físicas y/o químicas. El equipo puede tener estaciones de control que realizan acciones de control en los procesos físicos. Las UPs deben conocer sus capacidades internas, estados y restricciones. El CP puede disponer de las UPs de modo tal que ejecuten diferentes procesos de producción o cambien el nivel de producción de acuerdo a

los requerimientos de la empresa o al estado interno de la UP.

Equipos. En este nivel se encuentra localizada la infraestructura, los dispositivos donde toman lugar los procesos y la instrumentación necesaria para realizar las mediciones y regulaciones de los mismos. También hay dispositivos de control y redes que están a cargo de controlar los procesos y obtener información del rendimiento de la infraestructura.

1.1.3. Integración de los Sistemas Automatizados de Producción

Los procesos de manufactura difieren de los de producción continua en los modos de producción, los métodos de control y los dispositivos requeridos. A pesar de estas diferencias, el proceso de automatización, en ambos casos, es jerárquico y con funciones específicas en cada nivel de la jerarquía. La integración de un proceso automatizado, incluye la comunicación interna en cada nivel -integración horizontal- y la comunicación entre niveles -integración vertical-, con el fin de lograr sistemas que permitan ejecutar las diferentes tareas de control existentes en una empresa.

Sin embargo, hasta ahora, las metodologías planteadas para la integración de sistemas de producción tales como CIMOSA de CIMOSA Association [35], GRAI-GREI de la Universidad de Burdeos [21], y el modelo PERA [45] de la Universidad de Pardue, no toman en cuentas características inherentes de los sistemas de producción continua, puesto que ellas fueron concebidas para los procesos de manufactura.

Como una posible solución a este problema, J. Montilva et. al. en [43] proponen una arquitectura denominada *Modelo de Referencia de Automatización Integral* (MRAI) que describe, de una manera genérica, como lograr la integración de los procesos de negocio, procesos de datos y conocimiento, aplicaciones de software y sistemas de información dentro de una empresa de producción continua, con el propósito de mejorar el rendimiento global de la organización.

MRAI presenta un método para la elaboración de planes estratégicos que permiten definir la estructura global integrada de información, gestión, comunicación y control que requiere una empresa de producción continua. El método se basa en la aplicación de las técnicas provenientes de los Sistemas de Información Empresarial y de la Ingeniería de Software Orientada a Objetos. En particular, se emplean conceptos de objetos y

procesos de negocios para construir un modelo integral del negocio, cuya elaboración se fundamenta en la aplicación del lenguaje de modelado UML (*Unified Modelling Language*) [50].

MRAI se basa en la Pirámide de Automatización y extiende la misma para considerar cinco caras denominadas *arquitecturas*, las cuales representan las estructuras que deben tener los elementos de datos, información, control y decisión de una empresa, con el fin de alcanzar un alto grado de automatización integral. Estas cinco arquitecturas descansan sobre el proceso productivo propiamente dicho, el cual se denomina *proceso físico* (ver figura 1.3).

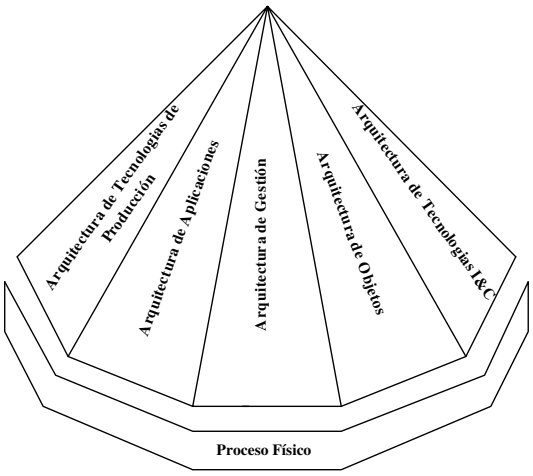


Figura 1.3: El Modelo MRAI

La *Arquitectura de Gestión* constituye la arquitectura central de la pirámide. Aquí se modelan los procesos de toma de decisiones de la empresa, requeridos para gerenciar el negocio a diferentes niveles jerárquicos.

La *Arquitectura de Tecnologías de Producción* representa las tecnologías que se emplean para transformar la materia prima. Esta arquitectura está estrechamente ligada con el proceso físico, pues las actividades que éste realiza se apoyan en estas tecnologías.

La *Arquitectura de Objetos* representa los tipos de entidades del negocio que de una u otra forma participan en sus diferentes procesos. Esta arquitectura, define las bases de datos y *data warehouses* requeridas por la empresa para soportar sus diferentes aplicaciones de software.

La *Arquitectura de Aplicaciones* describe todas y cada una de las aplicaciones de software, normalmente heterogéneas, que integran el negocio y que son necesarias para

apoyar la ejecución tanto de los procesos físicos, como los de toma de decisiones. Esta arquitectura está estructurada en varios niveles de complejidad. El nivel más alto de la arquitectura está constituido por los sistemas de información que posee el negocio, así como la relación que existe entre ellos. En el nivel intermedio se identifican las herramientas de planificación de recursos, tales como ERP (*Enterprise Resource Planning*), MES (*Manufacturing Execution Systems*) y MRP (*Manufacturing Resource Planning*). A un nivel más bajo se ubican las aplicaciones de propósito específico, tanto del proceso físico (p.ej., controladores, analizadores) como de los procesos de toma de decisiones (p.ej., procesadores de texto, hojas de cálculo).

Finalmente, *la Arquitectura de Tecnologías de Información y Comunicación* está constituida por todas las tecnologías de información y comunicación (redes de computadores, equipos de computación y software de operación y desarrollo) sobre las que se implementan las arquitecturas de aplicaciones y de objetos.

Dentro de este marco, uno de los aspectos más importantes que esta arquitectura toma en consideración es la integración de las aplicaciones, normalmente heterogéneas, que conforman a una empresa de producción continua. Estas aplicaciones son paquetes de software que ejecutan la evaluación y el control de actividades relacionadas con el Proceso de Toma de Decisiones (PTD), transformando la información en términos de indicadores. En general, este procedimiento consta de las siguientes actividades, siendo cualquiera la naturaleza del proceso sobre el que se ejecute:

- Sensar: El fin de esta tarea es medir las variables del proceso.
- Observar: Este proceso genera información acerca del estado del proceso.
- Evaluar: Involucra el análisis de la información del estado del proceso y su comparación con la misión asignada al proceso.
- Controlar: Define las acciones de control a realizar.
- Actuar: Las acciones de control se aplican sobre el proceso.

Estas aplicaciones manejan el conocimiento acerca de los recursos que informan sobre la disponibilidad y capacidad de las UPs, así como la información sobre los indicadores económicos, de ventas, de contabilidad de costos, etc. La arquitectura de aplicaciones,

el conocimiento acerca de los recursos, y la infraestructura deben ser manipulados de manera tal, que el PTD distribuido en toda la jerarquía ejecute sus funciones.

A resumidas cuentas se puede dilucidar que la integración de sistemas industriales no es una tarea sencilla. Esto se debe a que dichos sistemas son entidades complejas con un gran número de elementos. Sus procesos elaboran productos, consumen materiales, emplean información y recursos, están organizados en intrincadas redes de departamentos, divisiones, plantas, etc., que están distribuidos geográficamente y cooperan o negocian con múltiples compradores y vendedores. Además, estos sistemas son dinámicos y cambian continuamente sus elementos y relaciones, lo cual se debe a las constantes modificaciones que realizan en sus políticas de producción y gestión con el fin de satisfacer las demandas requeridas por el mercado.

Integración de Aplicaciones Heterogéneas en la Automatización de Procesos de Producción

En la evolución de la automatización de los procesos, la empresa ha desarrollado o adquirido aplicaciones especializadas para soportar el manejo de la información, debido a la gran diversidad de información necesaria, para llevar a cabo los diferentes procesos productivos y de gestión. Como se mencionó en el apartado anterior, muchas de estas aplicaciones son heterogéneas: provienen de diferentes proveedores, emplean modelos conceptuales y paradigmas muy diferentes y no tienen la capacidad para comunicarse entre sí o integrarse a otros productos. Estas aplicaciones heterogéneas aún siendo parte de la empresa, deben integrarse en el proceso de automatización.

La integración es más sencilla cuando todas las aplicaciones son construidas sobre la misma plataforma y bajo los mismos estándares. Cuando las aplicaciones no usan un sistemas manejador de base de datos común, ni los mismos modelos básicos de datos (relacional, jerárquico, de objetos, entre otros) se hace mucho más difícil integrar nuevos sistemas con esas aplicaciones, y entre ellas mismas. Muchos sistemas de aplicación existentes en las empresas (aplicaciones legadas) fueron desarrollados independientemente, usando una amplia variedad de formatos y diseños, porque no existían estándares empresariales. Se puede decir que, su desarrollo y evolución era conforme a la forma como las tecnologías iban emergiendo.

Los factores mencionados anteriormente, entre otros, han condicionado la evolución

de los sistemas automatizados que tratan de integrar un complejo industrial. Se ha hecho énfasis en el desarrollo o adquisición de sistemas que realicen y aseguren el trabajo de producción, pero no así en la optimización del mismo, tomando en cuenta sus relaciones con otros elementos que conforman el complejo productivo, ni en su integración con los sistemas de gestión. Hasta ahora la gestión de la empresa y el control de las plantas de producción funcionaban por separado, con lo cual la información enviada a la dirección de la empresa es anticuada e incompleta. A su vez, la que llega a las plantas, procedentes de los servicios centrales, es parcial y, a menudo, inadecuada.

Existen artificios que permiten extraer información de los procesos, pero muchos de ellos son sólo soluciones aisladas que particionan al ambiente industrial en "islas de información". Consecuentemente, las entidades que toman decisiones en cualquier nivel industrial, realizan sus funciones con cierta dificultad, al no disponer de mecanismos que les permitan obtener fácilmente información, confiable y segura, para realizar sus labores.

1.1.4. Definición del Problema

El problema que se presenta, es la falta de una herramienta que permita a los elementos, que llevan a cabo el control automatizado de la producción y la gestión, congregarse en un componente integrado, que haga posible el intercambio de la información entre las diversas áreas que conforman un complejo industrial, para así solucionar las dificultades derivadas del aislamiento de la información.

1.2. Objetivos de la Investigación

Esta investigación tendrá como objetivo principal proporcionar el modelo de una herramienta genérica que sea capaz de integrar, sincronizar, mantener y controlar las aplicaciones heterogéneas contenidas en las unidades de producción (dispuestas en la arquitectura de aplicaciones del modelo MRAI) que conforman un complejo de producción continuo.

Para cumplir con este objetivo se deberá:

- Definir y modelar la estructura y funcionalidad de la herramienta, de manera tal que pueda cumplir con las tareas preestablecidas para ella (integrar, sincronizar,

mantener y controlar todas las aplicaciones).

- Estudiar y evaluar diversos mecanismos que permitan implementar dicha herramienta.
- Diseñar una interfaz gráfica de usuario que medie entre la herramienta integradora y agentes externos humanos.
- Comprobar la factibilidad y viabilidad del modelo que se proponga a través de un ejemplo piloto. Para este fin se utilizó la empresa Aguas de Mérida.

1.3. Justificación de la Investigación

La integración de las aplicaciones que llevan a cabo el control y gestión de la producción se hace necesaria. A pesar de que las aplicaciones por separado son muy importantes, se lograría un beneficio mucho mayor si se integran totalmente en una solución completa y automatizada de negocios, para que así el proceso de toma de decisiones dentro de la empresa se realize de una manera ágil y efectiva. Se hace necesario entonces definir un estándar que pueda satisfacer los requerimientos de integración de los sistemas de aplicaciones de software y que, a su vez, sea aplicables a cualquier industria de procesos continuos dada la inexistencia de una herramienta para tal fin.

1.4. Conclusiones

En este capítulo se describió el escenario en el cual está enmarcado el proyecto de investigación. El mismo se refiere a la integración de las aplicaciones heterogéneas existentes en las unidades de producción de los sistemas de producción continua, y que conforman la arquitectura de aplicaciones del modelo de referencia MRAI.

Se estableció como objetivo principal, diseñar una herramienta integradora que permita congregar estas aplicaciones, con el fin de flexibilizar el intercambio de información entre los componentes que llevan a cabo, tanto la gestión como la producción de un complejo industrial.

Se justifica esta investigación, por los aportes implícitos que se derivan de tener información de primera mano en el momento de la toma de decisiones en la empresa.

Los capítulos sucesivos, de esta tesis, están organizados de la siguiente manera:

En el capítulo 2 se presenta una caracterización detallada de la integración de aplicaciones heterogéneas existentes en una empresa: tipos, arquitecturas y estrategias; de igual manera, se consideran algunas de las tecnologías o soluciones actuales aplicables a este dominio de integración.

En el capítulo 3 se sientan las bases de un conjunto de elementos teóricos sobre los cuales se basa la investigación. Los mismos, abarcan aspectos concernientes a los agentes de software, objetos y reglas de negocios. También se consideran los antecedentes o investigaciones previas realizadas, y relacionadas a la presente investigación.

En el capítulo 4 se propone una solución al problema planteado, se da una descripción detallada de la misma, así como sus limitaciones, ventajas y desventajas.

En el capítulo 5, con el fin de comprobar la validez de las ideas propuestas, se expone la implementación de un prototipo, y pruebas del mismo.

Por último, se plantean las conclusiones y recomendaciones finales de la investigación.

Capítulo 2

Integración de Aplicaciones Heterogéneas

En toda organización coexisten múltiples sistemas informáticos que operan con diferentes arquitecturas lógicas y físicas. Muchas veces, las demandas de información de las empresas pueden satisfacerse no con el desarrollo de nuevas aplicaciones, sino sencillamente con la apropiada integración de las que existen en la actualidad. Esta tarea puede resultar muy compleja desde el punto de vista técnico, dada la heterogeneidad de las plataformas de hardware y software que sustentan a las aplicaciones existentes. En el presente capítulo se exponen las diversas técnicas de integración de aplicaciones heterogéneas actuales.

2.1. Integración de Aplicaciones de Empresas (EAI, *Enterprise Application Integration*)

Anteriormente, las arquitecturas de sistemas que soportaban a una empresa eran pobremente planificadas [2]. Muchas organizaciones construyeron las mismas basadas en la tecnología del momento, sin prever cómo estos sistemas podrían, algún día, compartir información. Hay un gran número de organizaciones adaptadas a diferentes tipos de sistemas abiertos y propietarios, que aíslan la información. Todo esto dio como resultados ambientes heterogéneos.

Después de la creación de "islas de información" a través de generaciones tecnológicas, los usuarios y gerentes de negocios demandan la unificación de las aplicaciones que conforman a las mismas, con el fin de obtener una aplicación de empresa, simple y unificada, que represente al conjunto de procesos del negocio. La EAI surge como una

respuesta a dicha demanda, permitiendo que aplicaciones existentes compartan procesos y datos.

D. Linthicum en [37], define la EAI como "la ciencia, metodología, y tecnología subyacente en la integración de los sistemas de información de una empresa". EAI implica la comunicación irrestricta, de los datos y procesos de negocios, entre cualesquiera de las aplicaciones y fuentes de datos de una empresa. Este autor, divide la EAI en dos macro dominios: *Intra-empresa*, o integración de aplicaciones dentro de la empresa e *Inter-empresa*, o integración de aplicaciones entre dos o más entidades organizacionales distintas (p.ej. compañías, agencias, etc.). A su vez, cada macro dominio se divide en tipos de integración de aplicaciones, que incluyen el nivel de datos, nivel de interfaz de aplicaciones, nivel de métodos, nivel de interfaz de usuario y portales de información.

2.1.1. Integración Intra-empresa, o integración de aplicaciones dentro de una empresa

Consiste en la integración de los sistemas que prestan servicios a una empresa. Un ejemplo de esta integración podría ser la conjunción de un sistema de inventario viejo, basado en *mainframe* y escrito en Cobol[1] y DB2[2], con un sistema SAP R/3[3] recién instalado, o quizás el enlace de ambos sistemas con los de automatización de ventas desarrollado por una determinada compañía.

Anteriormente, las empresas cumplieron los requerimientos de integración de aplicaciones intra-empresa a través de cualquier medio disponible. Esto incluyó mecanismos tales como actualizaciones en lotes nocturnas o de fines de semana, transferencias de archivos vía FTP (*File Transfer Protocol*), y datos reordenados. Después, se enlazaron los sistemas utilizando, conjuntamente, productos *middeware* tales como DCE (*Distributed Computing Environment*) de la Fundación Open Software o MQSeries (*Message Queuing Series*) de IBM.

La naturaleza punto-a-punto de estos productos, que proporcionaban el intercambio de información entre sistemas de origen-destino únicos, rápidamente se hizo difícil de

[1]Lenguaje de programación de alto nivel, de tipo procedural

[2]Sistema de gestión de bases de datos de IBM

[3]Software estándar para el manejo de negocios, perteneciente al proveedor de software SAP (Sistemas, Aplicaciones y Productos para el Procesamiento de datos)

manejar y constantemente requería cambios extensivos para todos los sistemas participantes.

Hoy día, las tendencias de integración se orientan hacia enfoques y tecnologías estratégicas y orientadas a soluciones, que incluyen la creación de una infraestructura común que permita el intercambio de información en configuraciones uno-a-uno, uno-a-muchos, y muchos-a-muchos (ver figura 2.1).

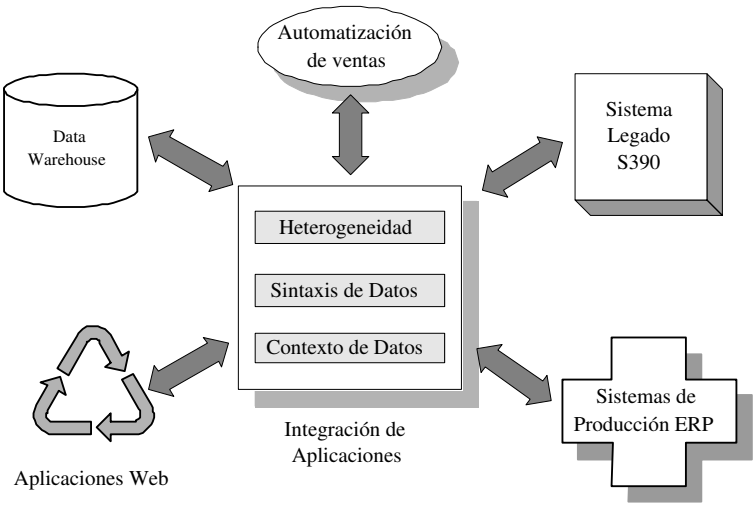


Figura 2.1: Infraestructura común para el intercambio de información en la empresa.

Tipos de EAI

Debido a que los sistemas existentes crecen y se desarrollan o se adquieren otros, las aplicaciones heterogéneas necesitan tener capacidad para comunicarse e interactuar entre ellas. Por medio de la integración de aplicaciones, las aplicaciones nuevas pueden tener acceso a la información y los procesos de negocios existentes.

Para llevar a cabo la EAI, las organizaciones deben entender tanto los procesos de negocios como los datos que las constituyen, con el objetivo de seleccionar qué elementos de dichos procesos y datos requieren de integración. Los tipos de soluciones EAI pueden tomar varias dimensiones. En [37], D. Linthicum identifica los cuatro niveles siguientes(ver figura 2.2):

EAI a nivel de datos. A nivel de los datos, la integración toma lugar entre repositorios de datos. Concretamente, los datos son extraídos desde una base de datos y utilizados para actualizar otras bases de datos, posiblemente después de las modificaciones apropiadas. Una de las principales ventajas de este enfoque es que no

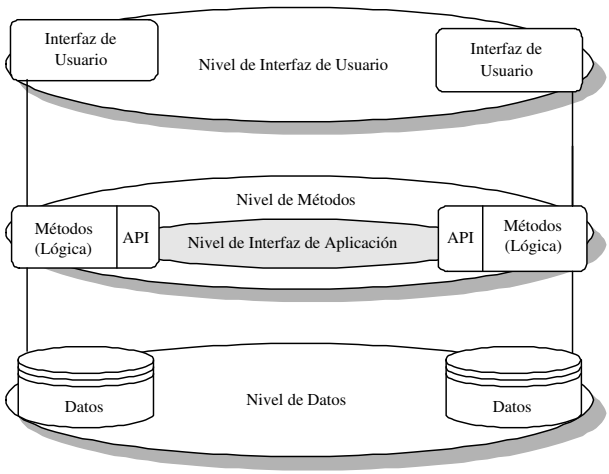


Figura 2.2: Tipos de IAE.

requiere de muchas modificaciones en las aplicaciones existentes. Además, cuenta con tecnologías probadas y económicas, p.ej. los *middleware* orientados a bases de datos tales como ODBC (*Open Database Connectivity*) y JDBC (*Java Database Connectivity*).

EAI a nivel de interfaz de aplicaciones. Una interfaz de aplicación es aquella que da acceso a los servicios proporcionados por una aplicación adquirida o un paquete estándar. Es posible distinguir tres tipos de servicios: de negocios, datos, y objetos. Un servicio de negocios proporciona acceso a alguna lógica del negocio, p.ej. calcular precios o actualizar información sobre compradores. Un servicio de datos provee una ruta hacia las bases de datos lógicas o físicas y es, en este sentido, similar a las EAI a nivel de datos. Un servicio de objetos es la combinación de servicios de negocios y datos, empaquetados como un objeto. La ventaja de los objetos es que las restricciones de integridad no pueden ser violadas, puesto que las actualizaciones de los datos siempre son logradas por el método apropiado de un objeto. Este tipo de EAI tiene mayor aplicabilidad en la integración de aplicaciones empaquetadas de proveedores tales como SAP[4], PeopleSoft[5], y Baan[6], las cuales exponen interfaces de sus procesos y datos, pero de diferentes maneras. Para integrar estos sistemas con otros existentes en la empresa, se deben

[4]Proveedor de software estándar para negocios
[5]Proveedor de aplicaciones y herramientas de productividad para hacer negocios en Internet
[6]Empresa de soluciones de software de gestión empresarial

usar estas interfaces para tener acceso a los procesos y datos, extraer la información, traducirla a un formato entendible por la aplicación destino, y transmitir la información.

EAI a nivel de métodos. Consiste en la comunicación de la lógica de negocios que existe en una o varias empresas. Por ejemplo, el método para actualizar los datos de un vendedor puede ser accedido por cualquier número de aplicaciones invocando un método común y compartido, generalmente residente en un servidor de aplicaciones compartidas o en una infraestructura de objetos distribuidos. Debido a que comparten métodos, las aplicaciones se hayan fuertemente acopladas y, por lo tanto, integradas. Los mecanismos para comunicar métodos entre aplicaciones son numerosos, entre ellos se encuentran: los objetos distribuidos, servidores de aplicaciones, monitores TP (*transaction processing*), *frameworks* y, simplemente la creación de una nueva aplicación que combine dos o más aplicaciones. A pesar de estas variadas soluciones, existen dos enfoques básicos:

1. Las organizaciones pueden crear un conjunto de servicios de aplicaciones (métodos) compartidos, que coexisten en un servidor físico común, tal como un servidor de aplicaciones.
2. Los métodos presentes en las aplicaciones, pueden comunicarse a través de una tecnología distribuida de métodos compartidos, tal como objetos distribuidos.

EAI a nivel de interfaces de usuarios. Es el enfoque de integración más primitivo, pero por ello no es el menos necesario (también es conocida como *screen scrapping*). En este escenario, las aplicaciones pueden acoplarse utilizando sus interfaces de usuario como punto común de integración. Por ejemplo, las aplicaciones *mainframe*, que no proporcionan acceso a los procesos de negocios o bases de datos que manejan, pueden ser accedidas a través de las interfaces de usuarios de la aplicación. Este enfoque puede ser visto como el menos atractivo, pero en muchos casos es el único disponible para la integración. Además, tiene la ventaja de no requerir cambios en las aplicaciones a ser integradas. A nivel de interfaz de usuario, se puede distinguir otro tipo de EAI muy popular, en estos tiempos, dado el auge de la *Web*. Este tipo de EAI lo constituyen los Portales de Información. Usando

este enfoque, las aplicaciones pueden integrarse, en una interfaz *web* de usuario, por medio de la información procedente de varias de ellas. En otras palabras, la información proveniente de varios lugares, tales como otros sitios (*sites*) o aplicaciones, se presenta en una misma interfaz de usuario. Esto ocurre, comúnmente, en un navegador (*browser*) *Web*.

2.1.2. Integración Inter-empresas, o integración de aplicaciones entre dos o más entidades organizacionales (*E-business*)

Los mecanismos, técnicas, y tecnologías de la integración intra-empresa, pueden ser aplicados a muchos escenarios de integración inter-empresas (comúnmente denominado *business-to-business*, B2B). EAI puede extender su alcance fuera de la empresa para incluir, por ejemplo, socios comerciales y vendedores en la arquitectura de integración. Esto permite, a cualquier aplicación o repositorio de datos, compartir con cualquier otra aplicación o repositorio de datos que esteé presente en el escenario comercial.

Los entornos de integración B2B están influenciados por tecnologías tales como: EDI (*Electronic Data Interchange*) y redes propietarias de valor añadido (*Value-Added Networks*, VANs). Sin embargo, en la actualidad, muchas arquitecturas B2B optan por tecnologías a tiempo real y compatibles con Internet como los agentes (*brokers*) de mensajes, servidores de aplicaciones y el nuevo estándar de intercambio de datos XML (*Extensible Markup Language*).

2.2. Arquitecturas para la Integración de Aplicaciones

La integración de aplicaciones puede ser soportada por varias arquitecturas diferentes [34]. Una de ellas es la solución punto-a-punto, donde una aplicación se conecta directamente a todas las otras aplicaciones (ver figura 2.3). Esta solución puede trabajar con un número pequeño de aplicaciones, pero cuando el número de éstas aumenta, las conexiones se hacen abrumadoras.

La tecnología *Brokers* de Mensajes reduce esta complejidad presente en la solución punto-a-punto (figura 2.4). La idea principal es reducir el número de interfaces introduciendo un *Broker* de Mensajes central, que soporte fácilmente las diversas interfaces.

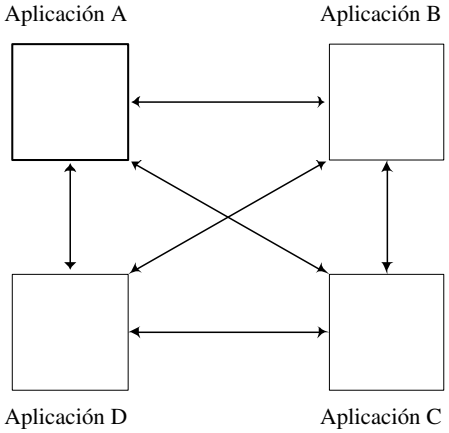


Figura 2.3: Integración Punto a Punto.

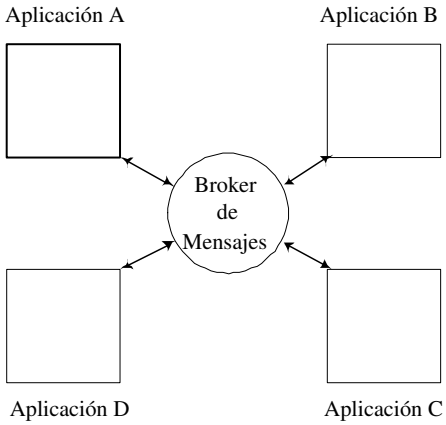


Figura 2.4: Integración a través de un *Broker* de Mensajes.

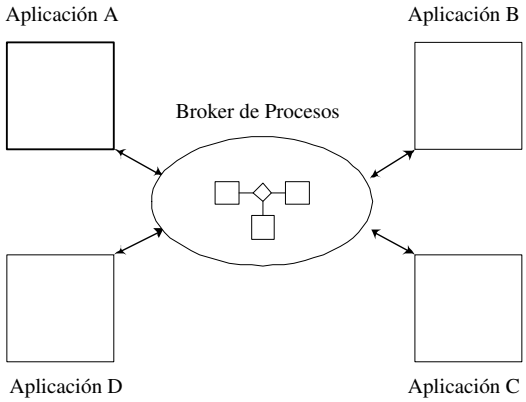


Figura 2.5: Integración a través de un *Broker* de Procesos.

Si una de las aplicaciones cambia su formato, sólo una conexión debe ser alterada: la del *Broker* de Mensajes.

El *Broker* de Procesos (figura 2.5), es una extensión del *Broker* de Mensajes. Este, además de manipular el formato de conversaciones, también encapsula la lógica del proceso para conectar aplicaciones. Cuando toda la lógica del proceso reside en un solo lugar, es posible estudiar, analizar y cambiar los procesos utilizando una interfaz gráfica de usuario. Esta tecnología puede ser vista como una continuación de los sistemas *workflow* [34].

2.3. Estrategias para la Integración de Aplicaciones

La integración de aplicaciones establece el curso de la conectividad de las nuevas aplicaciones con los sistemas de aplicaciones existentes, con tan pocas modificaciones

como sea posible a los sistemas existentes.

Según J. Montilva en [41], existen cuatro estrategias de integración de aplicaciones:

Transferencia. Implica la transferencia de un método, técnica, modelo o concepto de un dominio a otro, refiriéndose éste a un área particular de conocimiento o actividad de investigación en una disciplina.

Conexión. Conlleva la construcción de una interfaz entre los sistemas a integrar, sin requerir cambios en las características o propiedades esenciales de estos sistemas, puesto que el propósito de la interfaz es la de servir como puente entre ellos. Esta estrategia de integración es comúnmente usada en los llamados *middleware*.

Extensión. En esta estrategia de integración, un sistema de software existente es mejorado al incorporarle características de otro. Los elementos importados (conceptos, constructos, componentes de software y funciones) son ajustados o adaptados al sistema que está siendo extendido.

Combinación. Las características de dos o más sistemas de software son combinadas para crear un sistema completamente nuevo. El sistema resultante muestra las propiedades de los sistemas subyacentes, así como también nuevas propiedades que emergen de la integración.

2.4. Algunas Soluciones para la Integración de Aplicaciones

A continuación, se da una descripción de algunas de las técnicas utilizadas en la integración de aplicaciones, que han sido mencionadas en la sección 2.1 de este capítulo. Dichas técnicas se basan en las arquitecturas y estrategias, expuestas en las dos secciones anteriores.

2.4.1. Adaptación de Pantallas *Screen Scraping*

Uno de los primeros y más extendidos métodos de ampliar el uso de las aplicaciones es la técnica conocida como adaptación de pantallas (en inglés, *screen scraping*), que se puede utilizar tanto para renovar como para reciclar las aplicaciones de *mainframe* [16].

Esta tecnología, también conocida como captura de datos o navegación basada en pantallas, consiste en leer el flujo de información destinada a una terminal de *mainframe*, bien sea a través de un emulador de terminal basado en el cliente o en un programa basado en el servidor, y presentarlo como una interfaz gráfica de usuario, que se adapta mejor a lo que estos esperan de las modernas aplicaciones para equipos de escritorio (figura 2.6), evitando así modificar la aplicación de *mainframe*. También, permite introducir amplias modificaciones a la secuencia de información presentada al usuario, mediante la combinación de diversas pantallas en una única presentación gráfica, lo que da la impresión de haber modificado la aplicación sin variar la lógica del negocio.

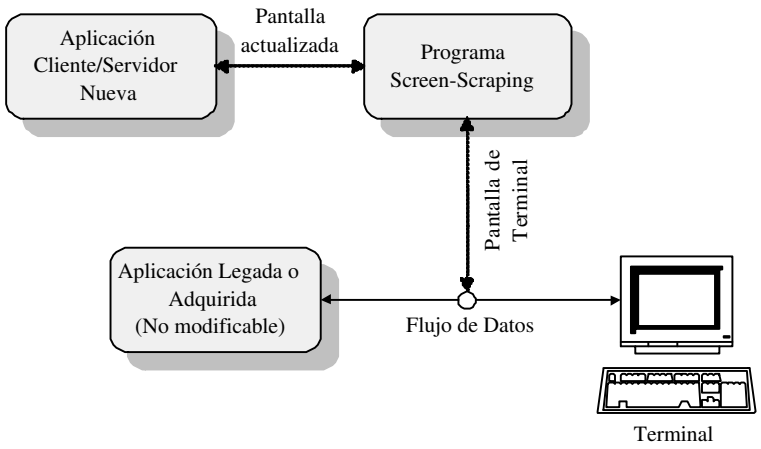


Figura 2.6: Aplicaciones *Screen Scrapping*

Como se indicó en el párrafo anterior, esta tecnología se utiliza de dos formas: mediante la renovación o por medio de reciclaje de las aplicaciones de *mainframe*. Mediante la renovación de una aplicación, simplemente se sustituyen una a una las pantallas de emulación de terminal por una interfaz gráfica de usuario. Por otra parte, el reciclaje aprovecha que la lógica del negocio de una aplicación heredada, se encuentra en gran medida expuesta en la secuencia de pantallas de emulación generada por las acciones de usuario. En este caso, la adaptación de pantallas permite capturar y recopilar dicha lógica, mediante la combinación de varias pantallas, para utilizarlas de una forma más sencilla.

Estos programas son usados para realizar interfaces con programas de aplicaciones remotas y son desarrollados en situaciones en las cuales, la única interfaz con una aplicación es a través de una interfaz de terminal existente. Esta solución de integración

se aplica cuando no hay recursos disponibles para modificar el sistema de aplicación remoto (por ejemplo, un sistema legado que carece de documentación) o es imposible modificar del todo el sistema remoto (por ejemplo, un paquete adquirido por un proveedor, sin código fuente). Cuando se utiliza esta técnica, la aplicación remota no tiene que ser modificada.

La técnica de adaptación de pantallas no está muy bien considerada por algunos sectores. A menudo se considera como un "mal menor" que tiende a ser lento, pesado y costoso de mantener. Sin embargo, gran parte de esta opinión se basa en las aplicaciones de adaptación de pantallas de hace algunos años que, como todo lo relacionado con la informática, se amplió y mejoró. En realidad, esta técnica de acceso basado en pantallas posee ciertas ventajas que la convierten en una potente herramienta para incorporar y ampliar la utilidad de las aplicaciones heredadas. Entre las más notables ventajas, destacan:

- un rápido desarrollo sin necesidad de modificar la aplicación remota
- un bajo costo
- reducción del impacto sobre el servicio de soporte técnico y los costos de entrenamiento
- muy efectiva debido a razones de rendimiento (sólo se debe usar para un bajo volumen de transacciones)

En esencia, el acceso actual basado en pantallas es muy diferente de la mera "adaptación de pantallas" del pasado, y dichas diferencias no se refieren simplemente a la estética. Actualmente, las nuevas tecnologías admiten una amplia gama de ambientes de desarrollo, permiten acceder a la aplicación heredada desde diferentes plataformas (incluidas Internet e Intranet), poseen una mayor capacidad de ampliación, resultan más fáciles de utilizar y requieren una formación mínima para su desarrollo y uso.

2.4.2. Motores de Interfaz

Proporciona a las aplicaciones la conectividad necesaria entre sistemas operativos heterogéneos, plataformas de *hardware*, bases de datos y redes (ver figura 2.7). Un motor de interfaz simplifica la programación de interfaces para múltiples sistemas de aplicación

heterogéneos, ya que ésta maneja las diferencias de protocolos de red, formatos de datos, y esquemas de registros. Los motores de interfaz son ventajosos cuando una o más aplicaciones a ser integradas no pueden ser modificadas (por ejemplo, las aplicaciones legadas o adquiridas).

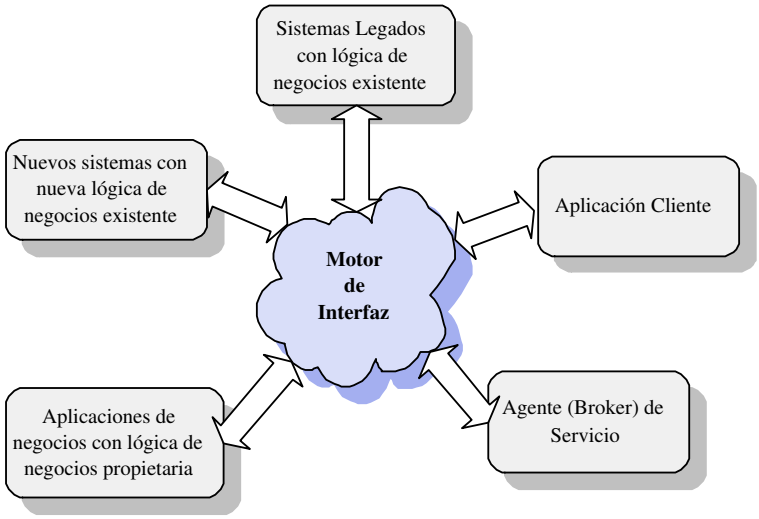


Figura 2.7: Motor de interfaz.

El motor de interfaz puede reducir los costos de mantenimiento y desarrollo porque proporciona [14]:

- *Escalabilidad y adaptabilidad*: Las aplicaciones pueden ser expandidas o cambiadas sin afectar a otros sistemas de aplicación que las acceden en forma remota.
- *Desarrollo simplificado de aplicaciones*: Todo lo que el desarrollador tiene que conocer para procesar una petición es cómo comunicarse con el motor de interfaz y qué entrada está esperando dicho motor.
- *Conectividad simplificada*: Cada aplicación objetivo tiene una sola conexión, en vez de muchas conexiones punto-punto que tengan que ser administradas separadamente para cada aplicación.

La conexión de un motor de interfaz algunas veces debe ser apoyada por scripts de screen-scraping, programación de interfaces o sistemas de mensajes para permitir la conectividad de la aplicación. Además, en algunos casos, es necesario realizar cambios a las aplicaciones existentes cuando estas no están soportadas por el motor de interfaz.

En estos casos puede ser necesario construir una interfaz o modificar una aplicación existente (si es posible) para ajustar la conexión.

2.4.3. Sistemas *Middleware*

El *middleware* es un módulo intermedio que actúa como conductor entre sistemas, permitiendo a cualquier usuario de sistemas de información comunicarse con varias fuentes de información que se encuentran conectadas por una red [20]. Un *midleware* es esencial para la migración de aplicaciones *mainframe* a aplicaciones cliente/servidor y para proporcionar comunicación a través de plataformas heterogéneas.

Desde un punto de vista amplio, una solución basada en productos *middleware* debe permitir conectar entre sí a una variedad de productos procedentes de diferentes proveedores. De esta forma se puede separar la estrategia de sistemas de información, de soluciones propietarias de un sólo proveedor.

El concepto de *middleware* no es un concepto nuevo. Los primeros monitores de teleproceso de los grandes sistemas basados en tecnología cliente-servidor ya se basaban en él, pero es con el nacimiento de la tecnología basada en sistemas abiertos que el concepto de *middleware* toma su máxima importancia.

Como se observa en la figura 2.8, los servicios *middleware* son un conjunto de *software* distribuido existente entre la aplicación, el sistema operativo y los servicios de red en un nodo del sistema de red.

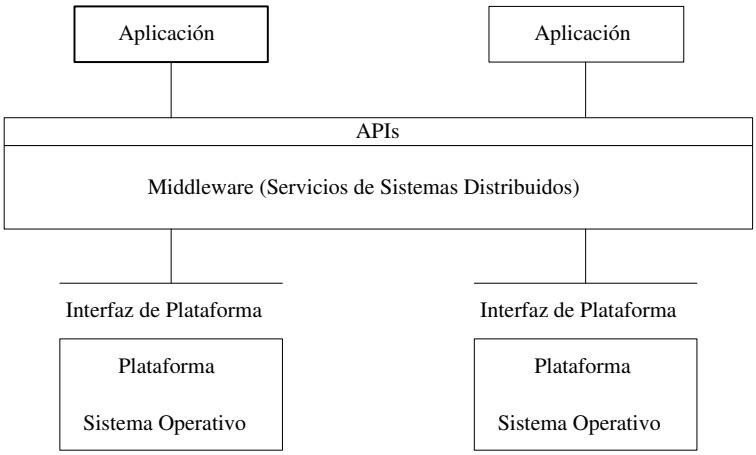


Figura 2.8: Uso de *Middleware*

Los servicios *middleware* proporcionan un conjunto funcional de APIs (*Application*

Programming Interfaces) adicionales al sistema operativo y servicios de red, que le conceden a la aplicación:

- Localización transparente[7] a través de la red, facilitando de esta manera la interacción con otras aplicaciones o servicios.
- Independencia de los servicios de red.
- Confiabilidad y disponibilidad.
- Escalar en capacidad, sin perder funcionalidad.

Las categorías del *middleware* se pueden definir de la siguiente forma [4]:

- **Monitores de proceso de transacciones** (TPMs, *Transaction Processing Monitors*). Proporcionan herramientas y un ambiente para desarrollar y desplegar aplicaciones distribuidas.
- **Llamadas a procedimientos remotos** (RPCs, *Remote Procedure Call*). Permite a la lógica de una aplicación estar distribuida en una red. La lógica del programa en los sistema remotos puede ser ejecutada como una simple llamada a una rutina local.
- ***Middleware* orientado a mensajes** (MOM, *Messaging Oriented Middleware*). Proporciona intercambio de datos programa a programa, permitiendo la creación de aplicaciones distribuidas. MOM es análogo a un *e-mail* en el sentido de que es asíncrono, y requiere de recipientes de mensajes para interpretar sus significados y tomar la acción apropiada.
- ***Middleware* para tecnologías orientadas a objetos** (ORB, *Objects Request Broker*). Permite a los objetos que conforman una aplicación estar distribuidos y compartidos, a través de redes heterogéneas.
- ***Middleware* orientados a bases de datos** (DOM, *Database Oriented Middleware*). Para acceso estándar a bases de datos. Permite desarrollar sistemas independizándolos de la base de datos que lo soporte.

[7] hacer que algo sea invisible al usuario

Entre las ventajas del uso de *middleware*, se tienen [20]:

- El proceso de desarrollo de aplicaciones se simplifica al independizar los entornos propietarios.
- Permite la interconectividad de los sistemas de información de la organización.
- Proporciona mayor control del negocio al poder contar con información procedente de distintas plataformas sobre el mismo soporte.
- Facilita el desarrollo de sistemas complejos con diferentes tecnologías y arquitecturas.

Dentro de los inconvenientes más importantes, señalados en [20], se destaca una mayor carga de máquina necesaria para que puedan funcionar. Entre sus campos de aplicación se mencionan:

- *Interconectividad*. Uno de los usos más importantes de las herramientas de *middleware*, es la de facilitar la interconectividad de los diferentes sistemas de una organización, integrando las diferentes islas de información departamentales que puedan existir.
- *Arquitectura orientada a objetos distribuidos*. El concepto de *middleware* permite también independizar los servicios proporcionados por diferentes objetos que se encuentran en una red, proporcionando una red de objetos independientes e interconectados entre sí.
- *Arquitectura cliente/servidor*. La utilización de *middleware* permite desarrollar aplicaciones en arquitectura cliente/servidor, independizando los servidores y clientes, facilitando la interrelación entre ellos y evitando dependencias de tecnologías propietarias.

2.4.4. EDI (*Electronic Data Interchange*)

EDI es el intercambio entre sistemas de información, por medios electrónicos, de datos estructurados de acuerdo con normas de mensajes acordadas [1]. A través del EDI, las partes involucradas cooperan sobre la base de un entendimiento claro y predefinido

acerca de un negocio común, que se lleva a cabo mediante la transmisión de datos electrónicos estructurados.

En el EDI, las interacciones entre las partes tienen lugar por medio de aplicaciones que actúan, a modo de interfaz, con los datos locales y pueden intercambiar información comercial estructurada. El EDI establece cómo se estructuran, para su posterior transmisión, los datos de los documentos electrónicos y define el significado comercial de cada elemento de datos. Para transmitir la información necesita un servicio de transporte adicional (por ejemplo, un sistema de tratamiento de mensajes o de transferencia de archivos).

Debe destacarse que el EDI respeta la autonomía de las partes involucradas, ya que no impone restricción alguna en el procesamiento interno de la información intercambiada o en los mecanismos de transmisión.

Los típicos campos de aplicación del EDI son el intercambio de información industrial, comercial, financiera, médica, administrativa, o cualquier otro tipo similar de información estructurada. Esta información, con independencia de su tipo concreto, se estructura en unos formatos que pueden ser procesados por las aplicaciones. Ejemplos de datos EDI son las facturas, órdenes de compra, declaraciones de aduanas, etc. La automatización de las interacciones por medio del EDI minimiza las transacciones sobre papel y la intervención humana, reduciéndose las tareas relativas a la reintroducción de datos, impresión, envío de documentos vía correo o vía fax.

2.4.5. Portales *Web*

Uno de los principales objetivos de las empresas que hacen negocios en la *web* es poner la información pertinente a disposición de los clientes, empleados y asociados de negocio, al mismo tiempo que generan ganancias en el proceso. Pero a medida que las organizaciones trasladan sus funciones de negocios críticas en línea, un obstáculo clave ha sido reunir la información proveniente de diversas bases de datos y aplicaciones, que en general se ejecutan sobre distintas plataformas.

Con ese desafío en mente, los portales *web* corporativos están señalando el rumbo en direcciones muy especializadas y ganando popularidad como la fuente de información clave dentro de una compañía [17]. Existen grandes mejoras en la tecnología de portales, que incluyen software de integración que vincula las funciones de atención al cliente con

las de administración (*front y back ends*), en un portal y que dan a los usuarios un único punto de acceso personalizado a múltiples tipos de información desde cualquier dispositivo, ya sea con o sin cables.

Hace un par de años, un portal *web* no era más que una ventana, un conjunto de casilleros que contenían imágenes llamativas, contenidos dispersos e hipervínculos para ir a otros sitios *web*. Por más revolucionario que fuera el portal desde el punto de vista conceptual, la información disponible solía decepcionar a la persona de negocios por su falta de relevancia. Pero a medida que las empresas empezaron a desarrollar el portal con fines corporativos, el interés creció considerablemente.

El valor del portal *web* va mucho más allá de la mera facilitación de información, ya que brinda servicios que conectan el contenido con aplicaciones integradas y comercio. Un clic del ratón ahora basta para disparar toda una serie de actividades, desde la autenticación hasta la verificación del crédito, el procesamiento de pedidos y el envío: transacciones masivas dirigidas a través del mundo del comercio electrónico de uno a otro extremo.

La capacidad de reunir el acceso a aplicaciones, información y servicios en forma personalizada, todo en la misma página *web*, proyecta una imagen de la compañía integrada y centrada en el usuario, y no en las aplicaciones o en los datos. Así, los usuarios pueden conectarse con otros empleados, clientes o asociados en función de sus áreas de interés. Además, los portales han evolucionado para incluir nuevos componentes cruciales que permiten a las empresas acceder y organizar información de negocios pertinente en todos los formatos, independientemente de dónde resida esa información, con lo cual se logra reducir el tiempo de desarrollo de los portales casi a la mitad.

2.5. Integración de Aplicaciones Heterogéneas a través de *Middlewares*

Como se mencionó en la sección anterior, una de las técnicas de integración de aplicaciones es a través de un *middleware*. El *middleware* de integración es un *software* intermedio que proporciona conectividad de aplicaciones, lleva a cabo la transformación y entrega de los datos entre múltiples sistemas de aplicaciones y bases de datos. La figura 2.9, ilustra el proceso en el cual una aplicación existente (o legada) comparte una base

da datos con una aplicación nueva, a través del *middleware*.

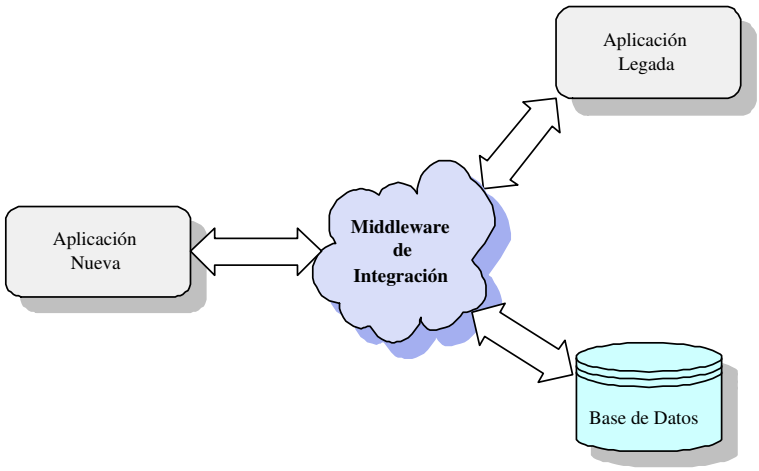


Figura 2.9: Integración a través de un *Middleware*

Cada programa de aplicación dispone de los medios necesarios o interfaces para comunicarse con otros usuarios o dispositivos. Esas entradas o puntos de salida existentes, pueden ser usados por el *middleware* de información para permitir la conexión e interoperatividad entre las aplicaciones.

A continuación se presentan algunos detalles técnicos de las categorías de *middlewares* señaladas en la sección 2.4.3

2.5.1. Monitores de procesamiento de transacciones (TPMs, *Transaction Processing Monitors*)

La tecnología de monitor TP (*Transaction Processing*) controla las transacciones de las aplicaciones y ejecuta el cálculo de la lógica/reglas de negocio, así como las actualizaciones a las bases de datos. Esta tecnología se usa ampliamente en el manejo de datos, acceso a redes, sistemas de seguridad, procesamiento de órdenes, reservaciones aéreas y servicios a clientes [7].

Puede proporcionar servicios de aplicación a cientos de clientes en un ambiente cliente/servidor distribuido. Esto lo logra multiplexando los requerimientos de transacciones (por tipo), en un número controlado de rutinas de procesamiento, que soportan servicios particulares. Estos eventos se muestran en la figura 2.10.

Los clientes son asignados, servidos y liberados utilizando servidores independientes que minimizan la sobrecarga. Las bases de datos ven, como clientes, sólo al conjunto

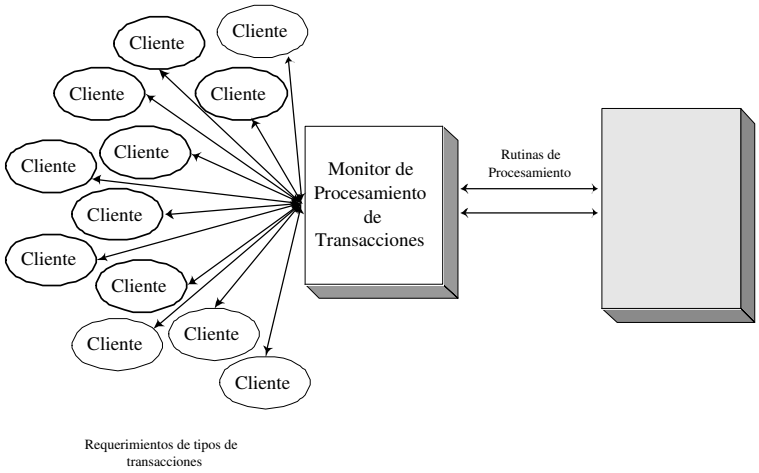


Figura 2.10: Monitores de procesamiento de transacciones.

controlado de rutinas de procesamiento.

La tecnología TP organiza numerosos requerimientos de clientes a través de rutinas de servicios de aplicaciones, que mejoran el desempeño del sistema. También esta tecnología (ubicada como un servidor), puede tomar la lógica de las transacciones de las aplicaciones desde el cliente. Esto reduce el número de actualizaciones requeridas por estas plataformas clientes. Además, incluye varias características de administración, tales como: reinicio de procesos que fallan, balanceo de carga dinámica y refuerzo de consistencia para datos distribuidos.

Los monitores TP son independientes de la arquitectura de bases de datos. Soportan cualquier modelo de negocios robusto y flexible, procedimientos modulares y reusables, así como APIs para componentes tales como: librerías de clientes heterogéneos, manejadores de bases de datos y recursos, entre otros.

2.5.2. Llamadas a procedimientos remotos (RPCs, *Remote Procedure Call*).

RPC es una infraestructura cliente/servidor que incrementa la interoperatividad, portabilidad y flexibilidad de una aplicación, permitiéndole estar distribuida sobre múltiples plataforma heterogéneas. Reduce la complejidad del desarrollo de aplicaciones que conectan múltiples sistemas operativos y protocolos de redes, aislando a los desarrolladores de aplicaciones de los detalles pertinentes a los diferentes sistemas operativos e interfaces de redes (las llamadas a funciones son las interfaces del programador cuando

usan RPC) [6].

Para tener acceso a la porción servidor remoto de una aplicación, las llamadas a funciones especiales, RPCs, son incrustadas en la porción cliente del programa de aplicación cliente/servidor. Cuando el programa cliente se compila, el compilador crea un *stub* local para la porción cliente y otro *stub* para la porción servidor de la aplicación. Estos stubs se invocan cuando la aplicación requiere una función remota y, por lo general, soportan llamadas sincrónicas entre clientes y servidores.

Utilizando RPC, la complejidad que involucra el desarrollo del procesamiento distribuido, se reduce a cumplir con la semántica de una llamada remota, sea que el cliente y el servidor estén o no localizados en el mismo sistema. Sin embargo, RPC incrementa la dificultad en el desarrollo de una aplicación, dada la complejidad de la naturaleza maestro-esclavo del mecanismo cliente/servidor.

RPC es apropiada para aplicaciones cliente/servidor en las que el cliente pueda hacer un requerimiento y esperar por la respuesta del servidor, antes de continuar con su propio procesamiento. Debido a que muchas RPCs no soportan interacciones cliente/servidor asincrónicas, no es aconsejable su implementación en aplicaciones que involucran objetos distribuidos o programación orientada a objetos.

2.5.3. *Middleware* orientado a mensajes (MOM, *Messaging Oriented Middleware*)

Al igual que RPC, MOM es una infraestructura cliente/servidor que incrementa la interoperatividad, portabilidad y flexibilidad de una aplicación, permitiéndole estar distribuida sobre múltiples plataforma heterogéneas. También reduce la complejidad del desarrollo de aplicaciones que conectan múltiples sistemas operativos y protocolos de redes, aislando a los desarrolladores de aplicaciones de los detalles pertinentes a los diferentes sistemas operativos e interfaces de redes (las APIs extendidas a través de las diversas plataformas y redes son, generalmente, proporcionadas por el MOM) [3].

MOM es un *software* que reside en ambas porciones de la arquitectura cliente/servidor y, típicamente, soporta llamadas asincrónicas entre las aplicaciones cliente y servidor. Las colas de mensajes suministran almacenamiento temporal, cuando el programa destino está ocupado o no conectado. MOM reduce la complejidad de la naturaleza

maestro-esclavo del mecanismo cliente/servidor. Este *software* es apropiado para aplicaciones que manejan eventos, así como en sistemas orientados a objetos.

2.5.4. *Middleware* para tecnologías orientadas a objetos (ORB, *Objects Request Broker*)

ORB es una tecnología *middleware* que maneja la comunicación y el intercambio de datos entre objetos. Promueve la interoperatividad de sistemas de objetos distribuidos, dado que permite la construcción de sistemas por medio de la unión de fragmentos de objetos, de distintos vendedores, que se comunican a través de él. Por lo general, los detalles de implementación del ORB no son importantes para los desarrolladores de los sistemas distribuidos, quienes se concentran sólo en los de las interfaces de los objetos [5].

Las funciones relevantes de una tecnología ORB son:

- Definición de interfaz
- Localización y posible activación de objetos remotos
- Comunicación entre clientes y objetos

ORB actúa como una clase de intercambio telefónico, proporcionando un directorio de servicios y ayudas para establecer conexiones entre los clientes y estos servicios. La figura 2.11 ilustra algunas de estas ideas.

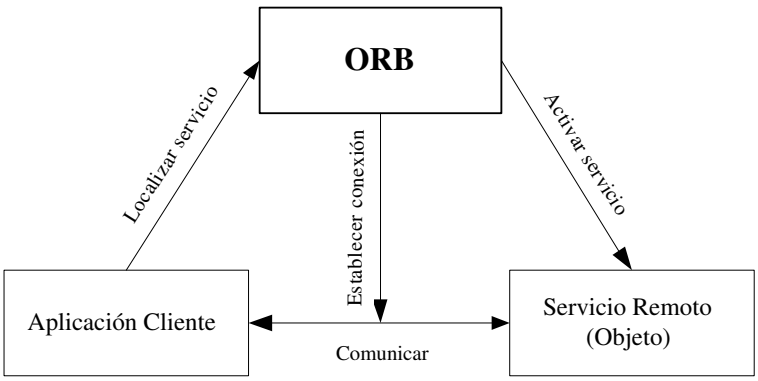


Figura 2.11: Objects Request Broker (ORB)

El ORB puede soportar muchas funciones con el fin de operar de una manera consistente y efectiva. Muchas de estas funciones son transparentes al usuario, siendo responsabilidad del ORB el hecho de crear la ilusión de localidad, es decir, hacer creer que

el objeto es local al cliente, cuando en realidad puede residir en una máquina o proceso diferente.

Un ORB le permite a los objetos esconder, a sus clientes, los detalles de su implementación. Esto puede incluir lenguaje de programación, sistema operativo, hardware, y localización del objeto.

Hay dos grandes tecnologías de ORB:

1. *Common Object Request Broker Architecture* (CORBA) de OMG (*Object Management Group*)

2. *Model Object Component* (COM) y *Distributed Model Object Component* (DCOM) de *Microsoft*.

CORBA

Se puede definir CORBA como un conjunto de especificaciones, definidas por el OMG (*Object Management Group*) cuya finalidad es facilitar la interoperabilidad entre componentes de software implementados en cualquier lenguaje, y que se ejecutan en cualquier sistema y plataforma hardware [20].

Para conseguir los objetivos que se persiguen, CORBA se apoya en tres pilares fundamentales: el lenguaje de definición de interfaces IDL (*Interface Definition Language*), los ORB (*Object Request Broker*), y el protocolo GIOP (*General Inter-ORB Protocol*). Estos se describen a continuación:

Lenguaje de definición de interfaces IDL. En CORBA la interfaz de un objeto es definida en IDL. La definición de la interfaz especifica los métodos que el objeto está preparado para realizar, sus parámetros de entrada, su resultado y cualquier excepción que pueda generarse durante la ejecución. En el momento de construir un objeto CORBA, el primer paso es definir cual va a ser la funcionalidad que va a proporcionar, para de esta manera, poder escribir la interfaz en IDL. Toda la información necesaria para construir un cliente del objeto es proporcionada por la interfaz. Se debe escoger un lenguaje de programación que facilite la implementación de la interfaz de cada objeto.

ORB. Es el sistema intermedio que establece relaciones cliente/servidor entre objetos (descrito al inicio de ésta sección). Mediante la utilización de un ORB, un cliente

puede invocar un método de un objeto servidor que se encuentre en la misma máquina o en otra distinta. El ORB intercepta la llamada realizada por el objeto que implementa la petición, pasa los parámetros, invoca el método y retorna los resultados. No es necesario que el cliente sepa dónde se localiza el objeto que ejecutará el método, el lenguaje en el que está programado, el sistema operativo, ni cualquier otro aspecto que no sea su interfaz. Hay que resaltar que los papeles de cliente y servidor que se dan a los objetos son simplemente para coordinar las interacciones entre ambos; estos papeles pueden cambiar ya que un objeto puede ser cliente o servidor dependiendo de la ocasión.

Protocolo GIOP. Dado que CORBA no dicta la implementación, se requiere un protocolo de red común, de tal manera que interactúen diversas implementaciones. El GIOP está especificado como la interfaz común, incluyendo el tipo y formato del mensaje, que se requiere para la interoperabilidad general. Se especifica la etapa semántica IIOP (*Internet Inter Orb Protocol*) para traducir el GIOP en TCP/IP. Esta combinación de GIOP e IIOP es necesaria para el soporte de la interoperabilidad.

COM y DCOM

COM hace referencia a una especificación e implementación, desarrollada por *Microsoft*, que provee una infraestructura para integrar componentes. Esta infraestructura soporta la interoperabilidad y reutilización de objetos distribuidos, permitiendo la construcción de sistemas a través de ensamblaje de componentes reusables provenientes, tal vez, de distintos vendedores.

COM define un API que permite la creación de componentes que se usan para la integración de aplicaciones, o para que diversos componentes interactúen. Siempre que se adhieran a una estructura binaria específica de *Microsoft*, los componentes, escritos en lenguajes diferentes, pueden interoperar.

DCOM es una extensión de COM que permite la interacción de componentes basada en una red. Mientras que los procesos COM pueden ejecutarse en una misma máquina (pero con espacios de direcciones distintas), los DCOM pueden extenderse a través de una red.

COM y DCOM son consideradas como tecnologías simples, que proporcionan una

gama de servicios para la interacción de componentes, desde aquellos que promueven la integración de componentes en una plataforma simple, hasta los que fomenta la interacción sobre redes heterogéneas.

RMI (*Remote Methods Invocation*)

Otro modelo ORB es el RMI; éste se ha especificado como parte del lenguaje/máquina virtual Java. RMI constituye una manera en la que un programador, utilizando el lenguaje de programación Java, puede escribir programas orientados a objetos, en el que los objetos en diferentes computadores pueden interactuar en una red distribuida. RMI es la versión Java de lo que se conoce como RPC, pero con la capacidad de pasar uno o más objetos en una solicitud. Los objetos pueden incluir información que cambie los servicios que se ejecutan en un computador remoto.

Una solicitud RMI es aquella que invoca el método de un objeto remoto. La solicitud tiene la misma sintaxis que se utilizaría para invocar un método de un objeto local (que se haya en el mismo computador).

2.5.5. *Middleware* orientados a bases de datos (DOM, *Database Oriented Middleware*)

DOM es una tecnología que permite la comunicación con una base de datos, sea desde una aplicación o entre bases de datos de diferentes tipos, en distintas plataformas y/o en múltiples localizaciones. Está definido por varios estándares de conectividad, entre los que destacan *Open Database Connectivity* (ODBC) y *Java Database Connectivity* (JDBC).

ODBC y JDBC

ODBC y JDBC son estándares creados por Microsoft. Ambos son interfaces a nivel de llamadas (*Call-level Interfaces*, CLIs), que proporcionan la conexión a distintas bases de datos.

ODBC es una especificación API para conectar programas a diferentes bases de datos, como Access, dBase, DB2, o Excel. JDBC es una especificación API para conectar los programas escritos en Java a los datos presentes en bases de datos populares.

En ambos, la API permite codificar órdenes de solicitud de acceso en SQL (*Structured Query Language*) que luego pasan al programa que administra la base de datos. Los resultados se devuelven a través de una interfaz similar.

2.6. Conclusiones

En las distintas organizaciones es frecuente encontrar sistemas no integrados o bien no todo lo integrado que sería deseable; este hecho se debe, en muchos casos, a una pobre planificación de los Sistemas de Información, en otros, a la historia de adquisiciones y fusiones entre organizaciones.

Sean cuales fueren los motivos de esta situación, el resultado es que se dispone de un conjunto de aplicaciones que manejan datos relacionados semánticamente entre sí, pero que no permiten su tratamiento y estudio de una forma global. Además de los datos producidos en la organización, frecuentemente existen datos externos (sobre el entorno o la competencia) que también son relevantes para ella; las fuentes de estos datos pueden ser muy variadas.

La misión del proceso de integración de información en una organización consiste en descubrir, almacenar y mantener las relaciones existentes entre sus datos, para que de esta forma puedan ser utilizados en mejorar los distintos procesos que se llevan a cabo en la organización (p.e., toma de decisiones, proceso de producción, etc.).

Existen diversos mecanismos para la interoperación o comunicación de la información existente en el negocio. Sin embargo, son soluciones que resuelven parcialmente el problema de integración. Aún más, entre muchas de ellas la incompatibilidad es un factor prominente.

Capítulo 3

Agentes y Procesos de Negocios

En este capítulo se presenta la exposición y análisis del grupo de teorías que sirvieron para fundamentar y sustentar la investigación realizada. Este conjunto de conocimientos constituyeron una plataforma sólida, a partir de la cual se derivaron los planteamientos y supuestos establecidos. También se presentan los antecedentes, más relevantes, relacionados con la investigación.

3.1. Agentes en Inteligencia Artificial

El estudio de los agentes es un área de investigación en pleno y rápido desarrollo. El abuso producido en el uso del término "agente" ha llevado a enmascarar lo que en realidad es un campo heterogéneo de investigación, aunque, eso sí con múltiples variantes. A pesar de que todavía no existe consenso para definir el concepto de agente, se han hecho ya importantes aproximaciones a la definición, diseño y construcción de diversos tipos de agentes.

Los esfuerzos en investigación sobre agentes inteligentes han sido dirigidos inicialmente por la comunidad de Inteligencia Artificial. La pregunta ¿Qué es un agente? ha suscitado una gran discusión en el seno de esta comunidad, como ocurrió en su tiempo con la pregunta ¿Qué es la inteligencia? [22].

Antes de realizar un acercamiento a los agentes inteligentes, se hace necesario ubicarlos en el contexto de la inteligencia artificial, que es donde ellos han nacido y crecido.

3.1.1. Inteligencia Artificial

La Inteligencia Artificial (IA) es un área científica de las ciencias de la computación. Se han dado muchas definiciones sobre ella, pudiéndose citar, como ejemplo, dos de ellas [19]:

- Campo de la informática, dedicado a hacer más inteligentes a los computadores.
- Estudio, investigación y desarrollo de técnicas que permiten a los computadores emular determinados dominios del comportamiento de las personas.

Actualmente, la IA abarca una enorme cantidad de áreas, desde aquellas de propósito general, como la percepción y el razonamiento lógico, a áreas más específicas, como la prueba matemática de teoremas, el procesamiento de lenguaje natural o el diagnóstico de enfermedades a través de sistemas expertos.

Cada vez más, los científicos de las diferentes áreas se abocan al estudio de la IA, pues, en general, encuentran en ella las herramientas y el vocabulario necesario para sistematizar y automatizar las tareas intelectuales en las que han estado inmersos. Simultáneamente, los científicos de la IA tienden cada vez más a aplicar los desarrollos alcanzados en el área, a los más diversos campos asociados con el comportamiento humano. Por ello, hoy puede considerarse la IA como un área universal.

Una clasificación de la IA

La tabla 3.1 muestra algunas definiciones de la IA. Las de la primera fila están asociadas a procesos mentales y de raciocinio, mientras que la de la segunda fila consideran el comportamiento humano. Las definiciones en la primera columna miden el éxito en función del desempeño humano y las de la segunda columna en función de la capacidad de raciocinio (un sistema es racional si realiza algo correcto).

Este conjunto de definiciones conllevan a cuatro posibles objetivos de la IA [22] (tabla 3.2).

Actuación humana: El paradigma del Test de Turing. En 1950, A. Turing propuso un test que fue diseñado para proporcionar una definición operacional de inteligencia. Turing definió el comportamiento inteligente, como la habilidad de alcanzar una actuación de tipo humano en todas las tareas cognitivas, suficiente

"El excitante nuevo esfuerzo en hacer pensar a las computadora . . . máquinas con mentes, en sentido pleno y literal" (Haugeland, 1985) "[La automatización de] actividades que asociamos con el pensamiento humano, actividades como pueden ser la toma de decisiones, la resolución de problemas, aprender . . ." (Bellman, 1978)	"El estudio de las facultades mentales a través de modelos computacionales" (Charniak and McDermott, 1985) "El estudio de métodos computacionales que hacen posible percibir, razonar y actuar´´ (Winston, 1992)
"El arte de construir máquinas que ejecutan funciones que cuando son realizadas por humanos requieren inteligencia." (Kurzweil, 1990) "El estudio de como hacer que las computadoras hagan cosas que actualmente las personas hacen mejor" (Rich and Knight, 1991)	"Un campo de estudio que trata de explicar y emular el comportamiento inteligente en términos de procesos computacionales" (Schalkoff, 1990) "La rama de la informática que está relacionada con la automatización del comportamiento inteligente" (Luger and Stubblefield, 1993)

Tabla 3.1: Definiciones de la IA. Extraído de [22]

Sistemas que piensan como humanos	Sistemas que piensan racionalmente
Sistemas que actúan como humanos	Sistemas que actúan racionalmente

Tabla 3.2: Objetivos de la IA

para engañar a un interlocutor. El test propuesto consiste en que un computador puede ser interrogado por un humano a través de un teclado, y se supera el test si el interlocutor no es capaz de adivinar si en el otro lado hay un humano o un computador.

Pensamiento humano: El paradigma de los modelos cognitivos. Si se quiere dotar a los programas de un pensamiento como el de los humanos, primero se debe tratar de definir cómo piensan los humanos. Esto se puede hacer de dos maneras: a través de la introspección o a través de experimentos psicológicos. Una vez que se tiene una teoría del pensamiento es posible llegar a expresarla como un programa de computadora. El campo interdisciplinario de la sicología cognitiva proporciona modelos de la IA y las técnicas experimentales de la sicología, para la construcción de teorías sobre el funcionamiento de la mente humana.

Pensar racionalmente: El paradigma de las leyes del pensamiento. El estudio del "pensamiento correcto", esto es, los procesos de razonamiento irrefutables, dio lugar a la codificación y formalización de las leyes del pensamiento, que se supone

que dirigen las operaciones de la mente, la lógica. La aplicación de la lógica a la IA ha dado lugar a programas que, con el suficiente tiempo y memoria, encuentran la solución a un problema descrito en notación lógica.

Actuar racionalmente: El paradigma de los agentes racionales. Es aquí donde la IA sitúa a los agentes. En este paradigma, la IA se basa en el estudio y construcción de agentes racionales. Un agente es simplemente una entidad que percibe su entorno y actúa sobre él. Por otra parte, actuar racionalmente significa actuar para alcanzar los propios objetivos según las propias creencias.

3.1.2. Definición de Agentes

Como se mencionó en la sección 3.1, a pesar no existir todavía un consenso para definir el concepto de agente, se han hecho ya importantes aproximaciones a la definición, diseño y construcción de diversos tipos de agentes.

Una de las primeras definiciones aparecidas describe a un agente como algo que percibe el entorno y actúa sobre él [48]. Esta definición depende fuertemente de lo que se considere como "entorno" y también de lo que se considere como "percibir" y "actuar".

Un agente autónomo es un sistema que habita en un entorno dinámico y complejo, en el que percibe y actúa de manera autónoma, alcanzando el conjunto de objetivos para el que fue diseñado [39]. Esta definición añade a los agentes un aspecto importante como es la autonomía y la capacidad de poseer un conjunto de objetivos. La autonomía referida a los agentes puede considerarse como que un agente ha de tener una actuación periódica, una ejecución espontánea e iniciativa, en la que él debe ser capaz de realizar acciones independientes.

También se define a un agente como un programa autocontenido que es capaz de controlar sus acciones y decisiones para alcanzar unos objetivos, basándose en su percepción del entorno [32].

A modo de resumen, se puede decir que un agente es un sistema que está situado y que forma parte de un determinado entorno, que percibe este entorno y que actúa en él de manera autónoma, persiguiendo el objetivo de cambiar su propia percepción [25]. De esta definición se deriva que los agentes están fuertemente ligados al entorno, y si se cambia el entorno puede ocurrir que lo que era un agente deje de serlo.

Esta definición también permite distinguir a un agente de un simple programa. Se puede decir que un programa "normal" es un agente si se consideran sus entradas como las percepciones que tiene del entorno, y sus salidas como la actuación sobre él, pero en realidad eso no es un agente, ya que sus salidas no afectan a sus percepciones futuras y además no tiene continuidad temporal. Se puede concluir que todos los agentes son programas, pero que no todos los programas son agentes.

3.1.3. Características de los Agentes

Cuando se intenta enumerar las características de los agentes aparece de nuevo el problema de los diferentes usos del término agente. A continuación se presenta una visión amplia de agente, que incluye características generales aceptadas para los agentes.

Visión amplia de los agentes

El uso más general del término agente se utiliza para definir sistemas hardware o software, con las siguientes propiedades [32]:

Autonomía. Los agentes han de actuar sin la intervención directa de los humanos o de otros agentes y, además han de tener algún tipo de control sobre sus acciones y su estado interno.

***Sociabilidad* (habilidades sociales).** Los agentes pueden interactuar con otros agentes (y posiblemente con humanos), mediante algún tipo de lenguaje de comunicación de agentes.

Reactividad. Los agentes perciben su entorno (que puede ser el mundo real, un usuario a través de una interfaz gráfica, un conjunto de agentes, Internet o quizás la combinación de algunos de los anteriores) y responde en un tiempo aceptable a los cambios que ocurren en él.

Capacidad de iniciativa. Los agentes no sólo han de reaccionar a los cambios de su entorno, también han de ser capaces de tomar la iniciativa, exteriorizando algún tipo de comportamiento orientado a alcanzar unos objetivos.

Visión estricta de los agentes

En IA, se considera que las propiedades de autonomía, sociabilidad, reacción e iniciativa no son suficientes para caracterizar a los agentes [30]. Es corriente en la IA caracterizar a los agentes usando nociones mentales como pueden ser conocimiento, creencias, intenciones y obligaciones.

En IA, un agente puede tener además de las características generales, las siguientes[32]:

Movilidad. Los agentes se trasladan a través de una red telemática para desempeñar tareas específicas.

Veracidad. Los agentes no comunican información falsa (se supone).

Benevolencia. Un agente ayuda a otros agentes y no entra en conflicto con sus propios objetivos.

Racionalidad. Un agente actúa en forma racional, con miras a cumplir sus objetivos.

Adaptación (Aprendizaje). Un agente cambia su comportamiento basado en las experiencias previas.

3.1.4. Teoría de Agentes

Se considera una teoría de agentes como la manera de especificar a los mismos [32]. Los teóricos de los agentes desarrollan formalismos que sirven para representar las propiedades de los agentes, de forma que, usando estos formalismos se puedan definir teorías que engloben las propiedades deseables de los agentes.

Actuación de los Agentes

El objetivo consiste en la construcción de agentes que puedan ser vistos como conocedores del mundo en que habitan y capaces de razonar sobre los posibles cambios que ocasionan sus acciones. Un agente no sólo ha de ser capaz de razonar sobre los cambios que hay en el entorno, sino también sobre los cambios que él mismo provoca en el entorno.

Como siempre, un buen punto de partida es estudiar algunas de las características deseables de los agentes. Un agente ha de ser capaz de aceptar nuevas tareas en forma

de metas descritas explícitamente. Ha de poder mejorar su actuación mediante entrenamiento o por aprendizaje de nuevos conocimientos sobre el dominio. Ha de adaptarse a cambios en el entorno y actualizar el conocimiento relevante. Para esto, los agentes necesitan saber cosas, tener conocimiento del estado actual del mundo, saber usar métodos de inferencia, inferir propiedades desconocidas mediante la propia percepción del mundo, han de saber sus objetivos, etc. De estas propiedades se deriva que un agente es cualquier cosa que percibe su entorno mediante sensores y que actúa sobre él mediante efectores o actuadores. En la figura 3.1, podemos ver un diagrama que representa a este agente genérico.

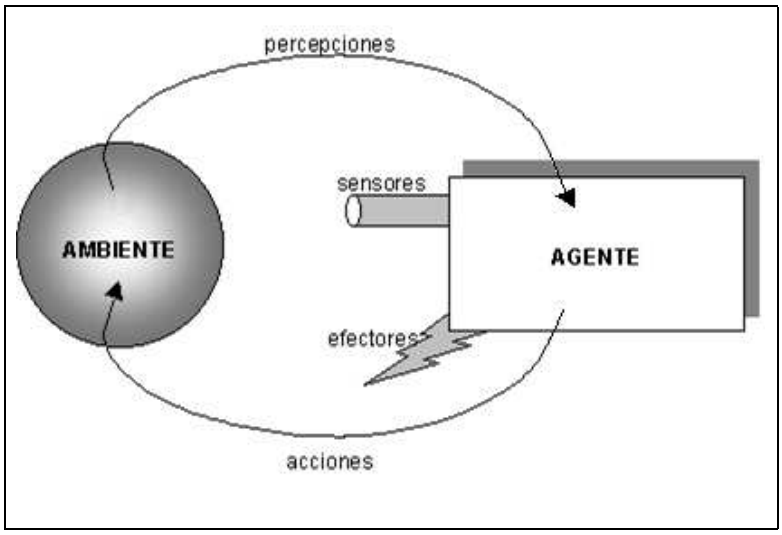


Figura 3.1: Agente genérico que interactúa con el entorno

Hace falta ahora encontrar una medida sobre la conducta de los agentes, es decir, sobre la inteligencia. Un agente racional es aquel que realiza la acción correcta [48]. La acción correcta es aquella que permite que el agente tenga éxito. La idea de éxito depende de cada agente, tener éxito puede significar supervivencia, alcanzar el máximo beneficio, entre otros. En este punto se puede definir un agente racional ideal como "algo" que, para cada secuencia de percepciones, trata de realizar una acción que maximiza su medida de actuación, en función de lo percibido y del contenido de su base de conocimiento. Se puede caracterizar a un agente según su entorno y sus percepciones, acciones y objetivos.

Estructura de los Agentes

¿Cómo trabajan interiormente los agentes?. Es función de la IA el diseño del programa de un agente; este programa implementa la relación que existe entre las percepciones y las acciones de deben realizarse. El programa del agente se ejecuta en una determinada arquitectura de hardware, y esta arquitectura hace que las percepciones de los sensores estén disponibles para el programa. La relación entre agente, programa y arquitectura puede representarse como: agente = arquitectura + programa.

Para diseñar el programa de un agente hay que formarse una idea de las posibles percepciones y las acciones, los objetivos o qué medida de actuación puede alcanzar el agente, y además, en qué tipo de entorno se situará. Estos cuatro aspectos, se utilizan para caracterizar a un agente. En la tabla 3.3 se muestran ejemplos.

AGENTE	PERCEPCIONES	ACCIONES	OBJETIVOS	ENTORNO
Sistema de diagnóstico médico	Síntomas de los pacientes	Preguntas, tests, tratamientos	Salud del paciente, minimizar costes	Paciente, hospital
Analizador de imágenes vía satélite	Pixels de distinta intensidad, color, etc.	Dibujar una escena, obtener categoría	Corregir categoría	Imágenes enviadas por un satélite en órbita
Profesor interactivo de inglés	Palabras escritas mediante un teclado	Mostrar ejercicios, sugerencias, correcciones	Maximizar la puntuación de los tests del alumno	Conjunto de estudiantes
Controlador de una refinería	Temperatura, presión	Abrir, cerrar válvulas; ajustar temperatura	Maximizar seguridad y calidad del producto	Refinería

Tabla 3.3: Ejemplos de tipos de agentes con aspectos que los caracterizan. Extraído de [22]

Entorno

El entorno es donde habitan los agentes. Como ya se ha dicho, el entorno determina que un agente lo sea; esto es, el entorno limita y condiciona al tipo de agente que podemos encontrar en él. Esto significa que antes de diseñar un agente hay que tener en cuenta las propiedades del entorno. El entorno puede clasificarse como:

- *Real vs. Digital.* Un entorno real sería aquel en el que se encuentran los robots autónomos (p.ej. almacenes, fábricas). El clásico ejemplo de entorno digital es

Internet.

- *Accesible vs. Inaccesible.* Cuando los sensores del agente son capaces de obtener toda la información que hay en el entorno se dice que éste es accesible. Si parte de la información relevante del mundo queda oculta al agente significa que es inaccesible. En un entorno totalmente accesible no es necesario que el agente mantenga un estado interno, ya que a través de sus sensores conoce en todo momento el estado en el que se encuentra.

- *Determinista vs. No determinista.* El entorno es determinista si existe un conjunto de reglas que permiten pronosticar cómo va a cambiar. En otras palabras, en estos casos el futuro del entorno viene totalmente determinado por su estado actual y por la acción que ejecuta el agente. Para decir si un entorno es determinista o no determinista, hay que hacerlo desde el punto de vista del agente, ya que si el entorno es inaccesible, el agente considerará que es no determinista, aunque para el diseñador sea lo contrario.

- *Episódico vs. No episódico.* En un entorno episódico la interacción del agente con el entorno está dividida en fases de percibir y actuar. La percepción actual no está determinada por fases o episodios anteriores y no es necesario tener en cuenta las consecuencias futuras de las acciones actuales, esto significa que desaparece la necesidad que el agente planifique su actuación.

- *Estático vs. Dinámico.* Si el entorno puede cambiar mientras el agente está tomando decisiones respecto a alguna percepción, entonces es dinámico. Los agentes que están en entornos dinámicos son mucho más complejos que los que actúan en entornos estáticos.

- *Discreto vs. Continuo.* El entorno es discreto si existe una cantidad limitada de percepciones y acciones distintas y claramente discernibles.

3.1.5. Clasificación de los Agentes

De acuerdo a sus características [54], los agentes se pueden clasificar:

A. Según su capacidad de razonamiento

A.1. Agentes Reactivos. Reaccionan a cambios en su medio ambiente o a mensajes provenientes de otros agentes. No son capaces de razonar acerca de sus intenciones (manejo de metas). Sus acciones se realizan como resultado de reglas que se disparan o de la ejecución de planes. Después, se actualiza la base de hechos del agente y se envían mensajes a otros agentes o al medio que lo rodea. Los sistemas expertos compuestos de una base de conocimientos (que contienen un conjunto de reglas), una base de hechos y una máquina de inferencias, son ejemplos de este tipo de agentes. En los Sistemas Multiagentes (SMA, varios agentes que resuelven un problema de manera cooperativa), este tipo de agente también es capaz de comunicarse con otros agentes: escogiendo y mandando o recibiendo e interpretando mensajes, de acuerdo a la situación actual.

A.2. Agentes Intencionales. Son capaces de razonar acerca de sus intenciones y conocimientos, crear planes de acción y ejecutar dichos planes. Los agentes intencionales pueden ser considerados como sistemas de planificación:

- Pueden seleccionar sus metas (de acuerdo a sus motivaciones) y razonar sobre ellas (detectar y resolver conflictos y coincidencias de metas).
- Pueden seleccionar o crear planes (programación de acciones).
- Pueden detectar conflictos entre planes, ejecutar y revisar dichos planes.

En SMA, los agentes intencionales se coordinan entre sí al intercambiar información acerca de sus creencias, metas o acciones. Esta información se añade a sus planes.

A.3. Agentes Sociales. Tienen las capacidades de los agentes intencionales y además poseen modelos explícitos de otros agentes. De aquí que un agente social deba ser capaz de:

- Mantener los modelos de los otros agentes, mediante la actualización de conocimientos, metas y planes.
- Razonar sobre el conocimiento incorporado a estos modelos (intenciones, compromisos, reacciones anticipadas y comportamientos hipotéticos).
- Tomar sus decisiones y crear planes con respecto a los modelos de los otros agentes.

B. Según su autonomía, aprendizaje y cooperación

B.1. Agentes Colaborativos. Estos agentes enfatizan su autonomía y cooperación (con otros agentes) para realizar sus tareas. Pueden aprender, pero este aspecto no tiene tanta importancia, como los anteriores, para su operación. Para tener un conjunto coordinado de agentes colaborativos, éstos tienen que negociar para alcanzar compromisos mutuamente aceptados en alguna forma. Pueden usarse para:

- Resolver problemas que son demasiado grandes para sistemas centralizados (debido a limitaciones de recursos o en los que se necesita tolerancias a fallas).
- Permitir la interconexión y operación de sistemas existentes.
- Dar solución a problemas inherentemente distribuidos.
- Dar solución a los problemas donde la experiencia se encuentra distribuida.

B.2. Agentes de Interfaz. Estos agentes ponen énfasis en su autonomía y aprendizaje para realizar sus tareas. El caso más claro de este tipo de agentes corresponde al de un asistente personal, que colabora con sus usuarios en el mismo ambiente de trabajo. Esencialmente, los agentes de interfaz asisten y dan soporte al usuario para aprender el uso de una aplicación. El agente del usuario observa y monitorea sus acciones a través de la interfaz con el usuario, y le da sugerencias para mejorar su tarea. Así, el agente del usuario actúa como un asistente personal que coopera con el usuario para realizar una tarea con la aplicación .

Los agentes de interfaz aprenden para mejorar su ayuda al usuario en cuatro formas:

- Al observar e imitar al usuario.
- Al recibir retroalimentación del usuario.
- Al recibir instrucciones explícitas del usuario.
- Al pedir consejos a otros agentes.

La colaboración con otros agentes (si es que existe), se limita a solicitar consejos y no a conseguir compromisos como en el caso de agentes colaborativos.

C. Otros tipos de Agentes

C.1. Agentes Móviles. Los agentes móviles son programas de software capaces de viajar por redes de computadoras como Internet, interactuar con un servidor *host*, pedir información a nombre de su usuario y regresar a su lugar de origen, una vez que ha realizado las tareas específicas de su usuario.

C.2. Agentes de Información/Internet. Los agentes de información realizan la tarea de administrar, manipular o recolectar información proveniente de varias fuentes distribuidas. Los agentes de información pueden ser estáticos o móviles, no cooperativos o sociales y pueden o no aprender.

Por ejemplo, un agente de información estático puede interactuar con varias máquinas de búsquedas de Internet (p.ej. Yahoo, Lycos, WebCrawler, etc) y organizar las fuentes de información (p.ej. las URL de interés que cumplen con algún criterio de búsqueda), las cuales se entregan como respuesta al usuario.

C.3. Agentes Híbridos. Los agentes híbridos son aquellos que en su funcionamiento poseen la combinación de dos o más de las capacidades de los tipos anteriormente explicados.

C.4. Agentes Heterogéneos. Los sistemas de agentes heterogéneos integran diferentes clases de agentes y pueden además contener una o más clases de agentes híbridos. Actualmente, abunda una gran cantidad de productos de software que proporcionan una amplia gama de servicios, para un amplio rango de dominios. Aunque estos programas trabajan por separado, se ha incrementado la necesidad de lograr que interoperen. Ha surgido un nuevo dominio denominado Ingeniería del Software Basada en Agentes, cuya función es facilitar la interoperabilidad entre los diferentes agentes de software. Los beneficios de tener una tecnología de agentes heterogéneos son varios:

- Lograr que las diferentes aplicaciones interoperen y trabajen de forma cooperativa; lo cual permite aprovechar las capacidades de cada una.
- Los problemas referentes a la legalidad del software pueden mejorar, porque se puede obviar la necesidad de reescribir software costoso, dándole la oportunidad de continuar interoperando con otro sistema.

- La ingeniería basada en agentes de software proporciona una nueva aproximación para el diseño, implementación y mantenimiento de software, en general y de interoperabilidad del software, en particular.

3.1.6. Arquitecturas Concretas para Agentes

Se consideran cuatro clases de agentes, dependiendo de cómo se implemente la toma de decisiones [54]:

- *Agentes basados en lógica*, en los que la toma de decisiones se realiza a través de deducciones lógicas.
- *Agentes reactivos*, que transforman directamente situaciones en acciones.
- *Agentes creencia-deseo-intención*, que dependen de la manipulación de estructuras de datos que representan las creencias, los deseos y las intenciones del agente.
- *Arquitecturas en capas*, en los que la toma de decisiones se realiza a través de varias capas de software, cada una de las cuales representa un nivel de abstracción diferente sobre el entorno.

Arquitecturas basadas en lógica

Se basa en la idea de agentes como demostradores automáticos de sistemas. Si se dispone de una teoría que explique cómo un agente genera objetivos para satisfacer el objetivo final de diseño, esta teoría podría ser considerada una especificación de cómo debería comportarse el agente.

Programar un agente basado en esta arquitectura, implica el manejo de elementos como las reglas de deducción (el "programa" que determina la conducta del agente) y la base de datos actual (representa la información que el agente tiene acerca de su entorno).

El razonamiento que van a tener los agentes basados en esta arquitectura no es rápido, por lo que no resulta adecuado para entornos que cambian rápidamente. Una decisión tal vez puede ser la óptima en un momento dado, pero para cuando el agente se haya decidido, puede que ya no siga siendo la más adecuada.

Arquitecturas reactivas

La toma de decisiones de un agente se lleva a cabo teniendo en cuenta las conductas para el conjunto de tareas que han de ser realizadas. Cada conducta se puede ver como una función que continuamente toma como entrada lo percibido por el agente, transformándolo en una acción a desarrollar, y se puede definir por medio de una regla del tipo *situación-acción*.

Múltiples conductas se pueden disparar a la vez. Se pueden organizar las reglas de comportamiento en una jerarquía de capas, de manera que las inferiores son las más prioritarias, capaces de inhibir a las superiores. Las capas más altas contienen una representación más abstracta.

En función de lo que el agente perciba, se tomará la acción que le corresponda de mayor prioridad. El algoritmo resultante es de complejidad cuadrática, o incluso constante si se implementa a través de *hardware*.

Pese a las ventajas de las arquitecturas reactivas (simplicidad, economía, robustez...) existen ciertos problemas sin solución:

- Los agentes deben tener un modelo del entorno o trabajar con mucha información local.
- Es difícil hacer que este tipo de agentes aprendan con la experiencia.
- No hay una metodología específica para construir agentes reactivos.

Arquitecturas Creencia-Deseo-Intención

El razonamiento práctico tiene lugar a través de dos procesos: decidir ***qué*** objetivos se buscan alcanzar (deliberación), y ***cómo*** se van a realizar dichos objetivos (*means-ends*). Las intenciones juegan un papel crucial en el proceso de razonamiento práctico:

- La propiedad principal de las intenciones es que tienden a conducir a la acción
- Conducen el razonamiento *means-ends*
- Llevan a futuras deliberaciones
- Persisten

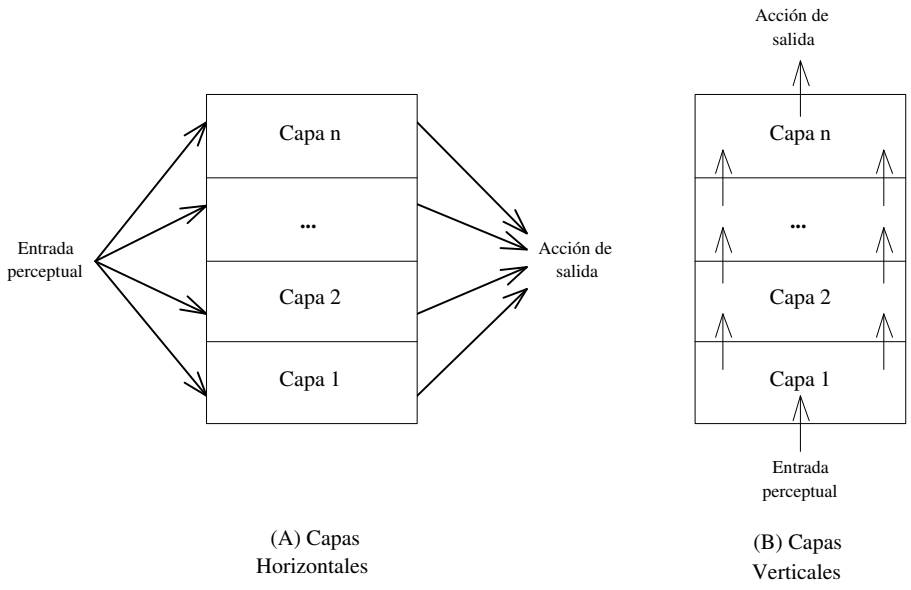


Figura 3.2: Flujos de control e información en las arquitecturas de agentes en capas

- Influyen en las creencias sobre las que se basarán futuros razonamientos prácticos

Un problema clave en el diseño de agentes con razonamiento práctico es lograr el equilibrio entre las diferentes posturas: se deben reconsiderar las decisiones lo suficientemente a menudo, como para que no se persigan intenciones absurdas (que anteriormente no lo eran), pero no se debe dedicar un tiempo excesivo a la reconsideración, puesto que esto reduciría demasiado el tiempo real de trabajo con el que llevarían a cabo los objetivos.

Arquitecturas en Capas

Un agente debe poder tener conductas reactivas y pro-activas. Una descomposición posible del sistema es, por lo tanto, tener diferentes subsistemas para cada una de estas conductas, creando así una jerarquía de capas.

Hay dos tipos de flujo de control en arquitecturas de capas (ver figura 3.2):

Horizontal. Cada capa está conectada directamente a los sensores de entrada y a los efectores de salida. Así cada capa, por sí misma, actúa como un agente proporcionando sugerencias de qué acción ejecutar.

Vertical. Existen capas intermedias no comunicadas directamente con el entorno.

El esquema horizontal es en concepto mucho más simple, puesto que se puede incluir en el sistema una capa por cada conducta deseada. Sin embargo, de este modo se dificulta

mantener la coherencia entre conductas que a su vez se ven relacionadas con otras.

3.1.7. Dominios de aplicación de los agentes

Actualmente, los agentes son aplicados en un amplio rango de dominios, entre los que destacan [33]:

- Aplicaciones industriales: control de procesos, manufactura, control de tráfico aéreo, etc.
- Aplicaciones comerciales: manejo de información en Internet, comercio electrónico, ***manejo de procesos de negocios***, etc.
- Aplicaciones médicas: monitoreo de pacientes, cuidados de la salud, etc.
- Entretenimiento: juegos, teatro y cine interactivo, etc.

En las siguientes secciones, se estudia a fondo la aplicación de los agentes en el manejo de procesos de negocios, por ser ésta el área que ocupa la presente investigación.

3.2. Agentes en el Manejo de Procesos de Negocios

Las empresas modernas están constituidas por unidades semi-autónomas, posiblemente distribuidas físicamente, donde cada una de ellas tiene cierto grado de control sobre recursos locales y, además, poseen diferentes requerimientos de información. Estas unidades semi-autónomas son coordinadas a través de un "proceso de negocio", el cual especifica las tareas que deben ejecutarse y las decisiones que deben tomarse en la generación de un producto o servicio.

Como se mencionó en el apartado anterior, uno de los campos de aplicación de los agentes lo constituye el manejo de procesos de negocios.

3.2.1. Procesos de Negocios

Una definición ampliamente aceptada de procesos de negocios (*business process*) es [18]:

> "Un proceso de negocio puede definirse como un conjunto de actividades que toma una o más clases de entradas, y crea una salida que es un servicio para un cliente".

Otras definiciones encontradas en la literatura definen un proceso de negocios de la siguiente manera:

> "Un proceso de negocio es simplemente un conjunto de actividades diseñadas para producir una salida específica dirigida a un cliente o mercado particular" [29].

> "Un proceso de negocio describe la secuencia de eventos, desde el inicio hasta su final, requeridas para originar un producto o servicio" [55].

Un proceso de negocios acepta entradas en términos de información y/o material, y produce salidas de la misma naturaleza (información y/o material). Por lo general, ellos involucran varias unidades funcionales de la organización y, frecuentemente, cruzan las fronteras de ésta.

J. Montilva en [42] define un proceso de negocios como un conjunto de actividades estructuradas y jerárquicas diseñadas para alcanzar fines organizacionales. Según este autor, los procesos pueden ser de diferentes tipos: de toma de decisiones, administrativo, de producción, de servicios, etc. En la figura 3.3 se muestran los procesos que usualmente se encuentran en una organización.

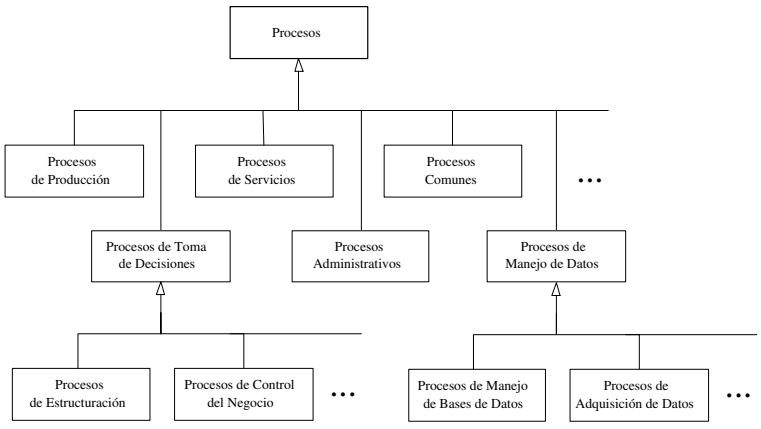


Figura 3.3: Generalización jerárquica de los tipos de procesos de negocios. Extraído de [42]

Existen dos aspectos importantes relacionados con los procesos de negocios: *las reglas de negocios y los objetos de negocios*. Estas relaciones puede ser vistas de la siguiente manera:

- Cualquier organización involucra, de muchas formas diferentes, una variada colección de entidades (p.ej. personal, clientes, materia prima, productos, etc.). Una entidad de negocio es un objeto relevante (concreto o abstracto) para la organización. Estas entidades relacionadas con la ejecución de los procesos se denominan *objetos de negocios.*

- Por otra parte, una organización debe adherirse, por ejemplo, a las leyes y regulaciones gubernamentales relacionadas con sus fines, y satisfacer las políticas, planes, y estándares establecidos por quienes la dirigen. Por lo tanto, los procesos son regulados o controlados por un conjunto de *reglas de negocios.*

A continuación se discute detalladamente cada uno de estos aspectos.

3.2.2. Objetos de Negocios

Una particularidad de la Orientación por Objetos (OO) [Anexo A] es la de poder modelar elementos de la vida real. Este es el caso de los objetos de negocios. Un objeto de negocio (ON) o *business object* es una representación mental de elementos comúnmente utilizados en un dominio particular del negocio y cubre elementos de diferentes tipos, tales como:

- Elementos físicos: Un producto, una planta, un empleado, ...

- Elementos de comunicación: Orden de pago, orden de producción,...

- Elementos de información: Capacidad de producción, inventario,...

Un ON [42] representa la estructura y el comportamiento de un concepto o algo del mundo real, que es significativo a un dominio particular de negocios (figura 3.4).

Los ONs son objetos de software interoperables, reusables y extensibles, que funcionan en un ambiente distribuido. Según J. Montilva [42], "Los objetos de negocios han sido diseñados para hacer posible el desarrollo de aplicaciones de software orientadas por objetos y distribuidas que reflejan la estructura y comportamiento de los negocios". Su importancia radica en que permite, a múltiples procesos de negocios, reutilizar objetos, conduciendo a desarrollo rápidos, bajos costos de mantenimiento, evolución incremental de las aplicaciones, entre otros.

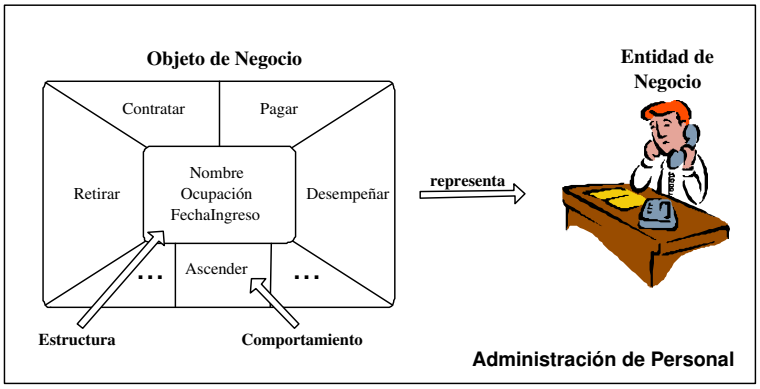


Figura 3.4: Ejemplo de un Objeto de Negocio

Un ON es una encapsulación de datos y métodos. En la figura 3.5, se puede observar una representación gráfica de la abstracción de los objetos del mundo real en objetos de software, donde el objeto del mundo real puede ser cualquier cosa y la idea básica es lograr la mejor representación posible en un objeto computacional, describiendo al objeto del mundo real por sus características en forma de atributos y su comportamiento a través de métodos.

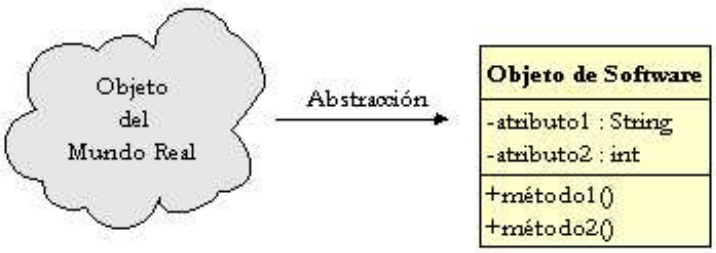


Figura 3.5: Abstracción de objetos

Desde el punto de vista de la programación, un ON es una colección de objetos de software, que representa las estructuras (atributos, relaciones, reglas, políticas) y el comportamiento de conceptos o cosas del mundo real que se manejan en un dominio particular de los negocios [24]. Ellos difieren de otros objetos, tales como: ventanas, menús, o barras de desplazamiento, en que éstos son comunes o familiares para usuarios en el dominio de los negocios.

Muchas veces algunos ONs poseen características que, a su vez, son propiedades de otros ONs, como puede ocurrir en el caso de las tareas que debe cumplir un gerente y las labores que deben realizarse en una unidad de producción.

Tipos de Objetos de Negocios

Se definen tres tipos de ONs [49]:

Objetos Entidades de Negocio. Son ONs que representan un concepto, rol, actor, recurso, lugar o algo. Ejemplos de estos objetos incluyen clientes, órdenes de ventas, producto, acciones, política de seguridad y vehículo. Un objeto entidad de negocio es lo primero que viene a la mente cuando se escucha el término "objeto de negocio".

Objetos Procesos de Negocio. Representan las actividades (interacciones, en términos de objetos) que comprende un proceso de negocio, el cual se puede concebir como un flujo de trabajo (*workflow*) que describe cómo se produce, paso a paso, un resultado final, y como los actores figuran en el proceso. Así los objetos entidades son los actores y recursos involucrados en las actividades descritas por un objeto proceso. Algunos ejemplos de este tipo de objetos serían: cumplimiento de orden, facturación, admisión a un hospital, activación de servicio, y entrega de producto. De hecho, algunos de estos procesos son esencialmente los mismos, puesto que sólo difieren en nombre, dependiendo del lenguaje industrial en el que estén inmersos.

Objetos Eventos de Negocio. Representa causas, sucesos, o el paso del tiempo. Estos objetos se inician o resultan de actividades entre objetos entidades. Ejemplos de ellos incluyen: fin del año fiscal, cantidad baja de inventario, factura esperada, arribo de envío a un puesto de control, y llamada concluida. Los objetos eventos de negocios son acontecimientos que son lo suficientemente importantes para ser llamados, rastreados, seguidos o evitados en un negocio.

3.2.3. Reglas de Negocios

Una regla de negocio (RN) o *business rule* es una declaración que define o restringe algún aspecto del negocio. Su propósito es hacer válida la estructura del negocio o controlar, o afectar el actuación del mismo [28]. De esto, se puede deducir que los procesos de negocios son regulados o controlados por un conjunto de RNs.

El término RN puede entenderse tanto a nivel de dominio del negocio, como al nivel operacional de un sistema de información (SI). Los conceptos fundamentales de RNs

están a nivel del dominio del negocio y, en ciertos casos, son automatizados a través de su implementación en un SI.

A nivel de dominio del negocio, una RN puede definirse como [52]:

- Una declaración de cómo se ejecuta el negocio, es decir, acerca de las pautas y restricciones con respecto a los estados y procesos en una organización.
- Una ley o costumbre que guía el desempeño o acciones de los actores conectados a la organización.
- La declaración de una política o condición que debe satisfacerse.

Una RN está basada en una política del negocio. Un ejemplo de ella, en una compañía de alquiler de vehículos, sería "sólo pueden rentarse, a los clientes, vehículos que estén en condiciones legales". Las RNs son expresiones declarativas: describen qué es lo que debe hacerse, más no cómo debe hacerse, es decir no describen/prescriben detalles de implementación.

Tipos de Reglas de Negocios

Las RNs pueden ser impuestas desde un medio externo al negocio, a través de regulaciones o leyes, o pueden ser definidas dentro del negocio con el fin de lograr los objetivos del mismo.

La siguiente clasificación se basa en la realizada por K. Taveter en [52]:

Restricciones de integridad. También llamadas *reglas de restricciones o reglas de integridad*. Es una afirmación que debe satisfacerse en todos los estados de evolución e historias de transición de estados de una empresa, que es vista como un sistema dinámico discreto. Existen restricciones de estado, que deben mantenerse en todo momento (p.ej. "un cliente de la compañía de alquiler de vehículos `EU-Rent` debe tener al menos 25 años de edad"), y restricciones de proceso, las cuales se refieren a la integridad dinámica de un sistema; ellas restringen las transiciones admisibles de un estado del sistema a otro (p.ej. una restricción de proceso puede declarar que los cambios de estado admisibles para un objeto denominado `OrdenDeAlquiler` están definidos por el siguiente camino de transición: reservado - asignado - operativo - devuelto y dejado).

Reglas de derivación. Es una regla de información o conocimiento que es derivada a partir de otra, a través de una inferencia o cálculo matemático. Las reglas de derivación capturan información o conocimiento que no necesitan ser almacenados explícitamente, pues éste puede ser derivado a partir de otros. Un ejemplo de una regla de derivación es el siguiente: "el precio de alquiler de un vehículo se deduce a partir del precio de alquiler del grupo de vehículos asignados para tal fin".

Reglas de reacción. También son llamadas reglas de estímulo-respuesta, reglas de acción, reglas de evento-acción, o reglas automáticas. Estas reglas tienen que ver con la invocación de acciones como respuestas a eventos. Ellas declaran las condiciones bajo las cuales se deben tomar acciones; esto incluye condiciones de disparo de eventos, pre-condiciones, y post-condiciones (efectos). Un ejemplo de una regla de reacción bajo el dominio de alquiler de vehículos sería: "cuando se reciba de un cliente una solicitud de reserva de un vehículo, perteneciente a un grupo específico, la sucursal debe chequear con la oficina central para asegurarse que el cliente no está en la 'lista negra' ".

Por otra parte P. Solivares en [51] clasifica las RNs en:

Reglas del modelo de datos. Engloba todas aquellas reglas que se encargan de controlar que la información básica almacenada para cada atributo o propiedad de una entidad u objeto es válida: no hay precios de artículos negativos, el sexo de una persona solo puede ser masculino o femenino, una fecha siempre debe ser una fecha válida ,etc.

Reglas de relación. Incluye todas aquellas reglas que controlan las relaciones entre los datos. Estas reglas especifican, por ejemplo, que todo pedido debe ser realizado por un cliente, y que el mismo debe estar dado de alta en nuestro sistema: además, una vez que un cliente haya hecho algún pedido, se deberá garantizar que no es posible eliminarlo, a menos que previamente se eliminen todos sus pedidos.

Reglas de derivación. Igual que en la clasificación anterior.

Reglas de restricción. Restringen los datos que el sistema puede contener. Este grupo de reglas se solapa en cierto modo con las reglas del modelo de datos, dado que

aquellas también impiden la introducción de datos erróneos. La diferencia estriba en que las reglas de restricción restringen el valor de los atributos o propiedades de una entidad más allá de las restricciones básicas que sobre las mismas existen: por ejemplo, para un saldo existe una regla básica (regla del modelo de datos) que indica que éste debe ser un número, pero además puede haber una regla que indique que el saldo nunca puede ser menor que cierta cantidad tope establecida para cierto tipo de clientes. La diferencia fundamental estriba en el hecho de que este tipo de reglas requiere para su verificación del acceso a otros fragmentos de información, algo que no sucede con las reglas del modelo de datos.

Reglas de flujo. Incluye aquellas reglas que determinan y limitan cómo fluye la información a través de un sistema y obligan a que se sigan solo los caminos válidos. Por ejemplo, un cliente puede hacer una petición de análisis a un laboratorio, que anota un encargado; hecho esto, se genera un parte para uno o más analistas, estos realizan las mediciones correspondientes y devuelven los partes con la información pertinente, a partir de la cual se genera un informe de análisis, que será un análisis válido sólo cuando sea firmado por los responsables de garantizar su corrección.

3.2.4. Objetos y Agentes

La definición tradicional de objeto y la idea antes expuesta de agente tiene al menos tres diferencias significativas:

1. Los agentes tienen mayor autonomía que los objetos. En particular, deciden por sí mismos si llevan a cabo o no una acción que ha sido solicitada por otro agente.

2. Los agentes pueden tener una conducta flexible (reactiva, pro-activa, social), mientras que esta noción no existe en el modelo orientado a objetos.

3. Un sistema multiagente es implementado mediante múltiples hebras, de forma que cada agente tenga el control de al menos una de ellas.

Una nueva tecnología comienza a explotarse, similar a la Programación Orientada a Objetos (POO) [Anexo A]. Esta tecnología la constituye la Programación Orientada a Agentes (POA). En ella, en vez de objetos, el principal constructo de programación

es un agente. La idea es tener un lote de agentes que desarrollan tareas simples y diferentes, coordinándose entre ellos mismos para resolver problemas mayores. Esto se puede combinar con los objetos existentes en un sistema.

¿Por qué utilizar agentes para desarrollar software?[AAR98]. Los agentes son consistentes con el paradigma OO. La eficiencia de la POA comienza con la eficiencia del paradigma de OO. La arquitectura de múltiples agentes es escalar y modular. La OO también. En la Tabla 3.4 se muestran algunas de las características resaltantes de los objetos y de los agentes.

Objetos	Agentes
Visibles	Informan de cosas
Pasivos	Realizan acciones
Tienen localización	Saben de cosas
Pueden contener cosas	Van a lugares

Tabla 3.4: Características resaltantes de objetos y agentes

Los objetos y agentes puede ser usados en la misma interfaz, pero se distinguen uno del otro. Los objetos pueden mantenerse como están: estáticos, predecibles, se mantienen justo en el sitio. Los agentes son repositorios funcionalmente adaptativos, ellos pueden notar eventos, usar reglas para interpretarlos y toman acciones basadas en esas interpretaciones. En la Tabla 3.5 se plantea la comparación entre la POO y la POA.

Descripción	POO	POA
Unidad Básica	Objeto/instancia	Agente
Parámetros que definen el estado de la Unidad Básica	Sin Restricciones	Creencia, ejecución, capacidades, escogencia, conocimiento, deseos, intenciones
Procesos de cálculo	Pase de mensajes y métodos de respuesta, operaciones	Pase de mensajes y métodos de respuesta
Tipos de mensajes	Sin restricciones, definidos dentro de la clase	Informes, requerimientos, ofertas, promesas, definidos dentro de la serie
Restricciones de métodos	Ninguno	Honestidad, consistencia
Secuencias de mensajes	Implícita	Definida en conversaciones
Convenciones sociales	Ninguno	Honestidad, consistencia

Tabla 3.5: Comparación entre la POO y la POA

3.3. Enfoques para el Manejo de Procesos de Negocios e Integración de Aplicaciones basados en Agentes

N. Jennings et. al. en [31], proponen una arquitectura multiagente para el manejo de procesos de negocios, denominada ADEPT (*Advanced Decision Envirotment for Process Task*), así como el diseño del agente apropiado para tal sistema, el cual involucra la transformación de la descripción de algún proceso de negocio en agencias, que se encuentran conectadas a un medio de comunicación común (ver figura 3.6). Los agentes pueden comunicarse por *E-mail*, o través de un ORB.

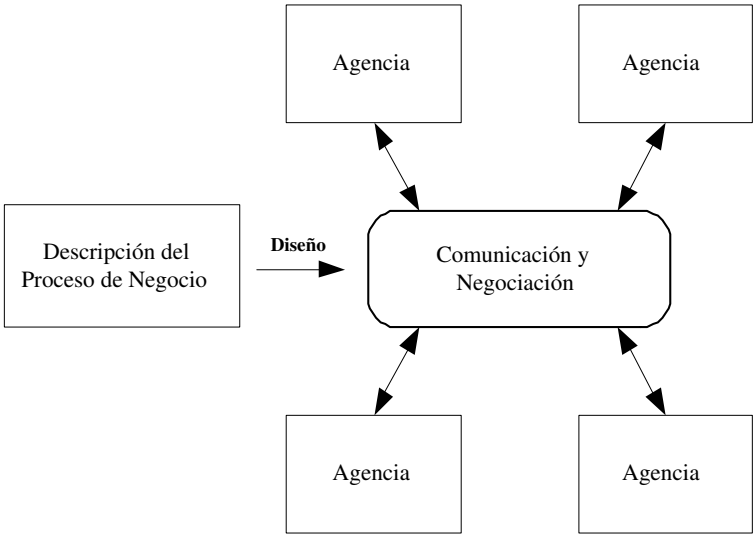


Figura 3.6: Diseño de un sistema para el manejo de procesos de negocios basado en agentes. Extraído de [31]

En ADEPT, una agencia se define recursivamente (figura 3.7): consiste de un agente responsable simple (o controlador), un conjunto (posiblemente vacío) de tareas que el agente responsable puede ejecutar, y un conjunto (posiblemente vacío) de sub-agencias. Cualquier comunicación con una agencia, debe hacerse por medio de su agente responsable, y las sub-agencias cooperan con éste agente en el logro de una tarea propia de la agencia.

De la manera descrita anteriormente, la arquitectura puede modelar una estructura jerárquica o categórica, o una mezcla de ambas. Cada agente, actuando de manera autónoma, valora constantemente la situación y decide: cómo comprometer los recursos de su agencia, requerir los servicios de otros agentes basado en acuerdos previos, o negociar un nuevo acuerdo de servicio. Un servicio corresponde a cada una de las tareas

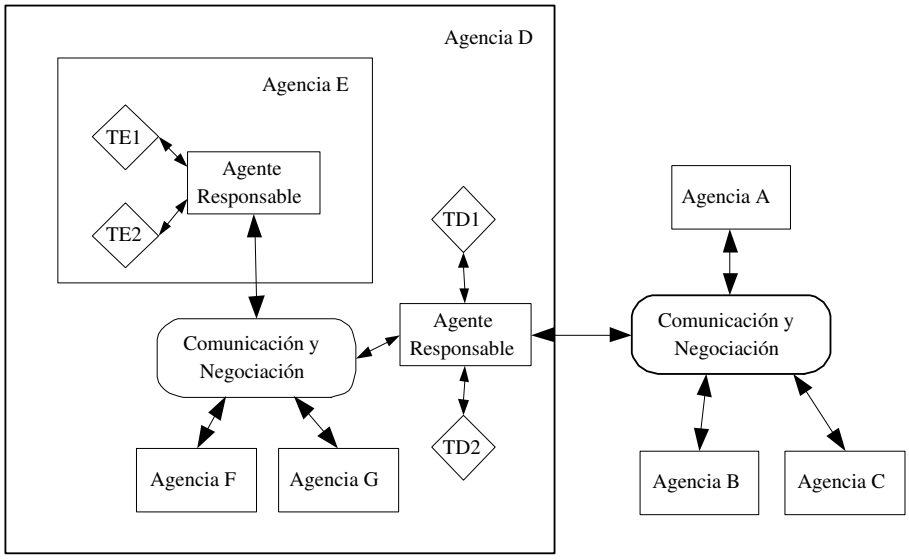


Figura 3.7: Jerarquía lógica de agencias. Extraído de [31]

más pequeñas (atómicas) pertenecientes a un proceso de negocio, o a una composición de varios servicios de otros agentes. Un servicio es el resultado de la negociación exitosa entre varios agentes

De este modo, el sistema ADEPT y la arquitectura de agentes están diseñados para asegurar la máxima flexibilidad en la adaptación a los cambios de los procesos de negocios. La autonomía de cada agente y sus acuerdos con otros agentes, es la clave de dicha flexibilidad.

En cuando a la integración de aplicaciones heterogéneas de una empresa, Z. Maamar et. al. en [38] presentan una aproximación para tal fin, basada en agentes, que sirve de soporte a la integración de aplicaciones considerando aspectos relevantes de la misma, tales como la distribución (donde toman lugar las operaciones de integración), y los recursos (aquellos que se necesitan para realizar dichas operaciones).

En este enfoque se especifica que una infraestructura, para la integración de aplicaciones, requiere trabajar con tres niveles diferentes (figura 3.8). Estos niveles son los siguientes:

- *Interconectividad a nivel físico*: permite la transferencia de datos simples, sin semántica. Primero los recursos son interconectados para intercambiar, después, los mensajes.

- *Integración a nivel de aplicación*: su propósito es el de realizar las operaciones

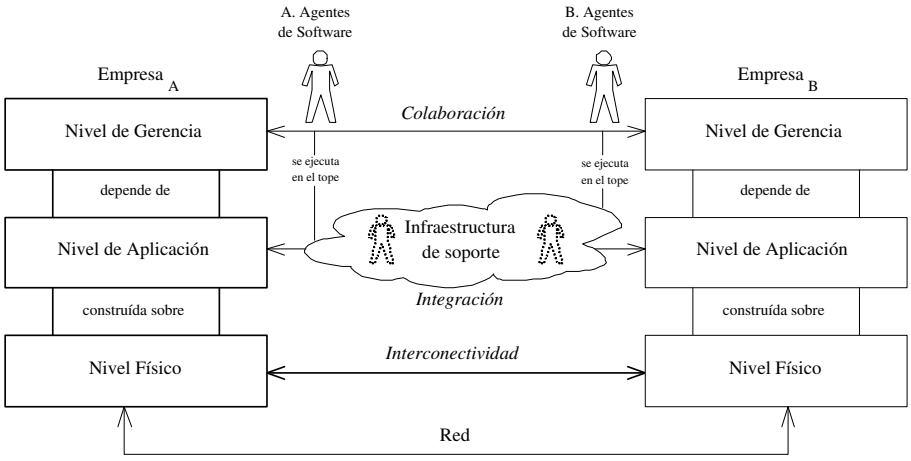


Figura 3.8: De la interconectividad a colaboración, a través de la integración. Extraído de [38]

entre aplicaciones diferentes (por lo general, distribuidas y heterogéneas).

- *Colaboración a nivel de gerencia*: va más allá de la integración de aplicaciones, a través de la expansión de los procesos a otras empresas. Para este propósito, debe usarse una colección de agentes de software (o solo agentes) que lleven a cabo estos procesos. Estos agentes, deberían ejecutarse en el tope de la infraestructura de soporte.

La infraestructura planteada en esta aproximación se ubica en el nivel de integración. Dicha infraestructura puede ser vista desde dos caras distintas: una de la empresa, y otra del usuario. Cada una de ellas, está representada por dos tipos de agentes, llamados agentes de interfaz y, agentes de usuario, respectivamente. En la tabla 3.6 se representan las distintas caras, los agentes que las representan y los servicios que éstos ofrecen.

En la arquitectura, cada empresa es encapsulada en una Sistema MultiAgente (SMA) los cuales mantienen su autonomía. Cada SMA integra varios agentes de interfaz, que están asociados a cada una de las aplicaciones de la empresa. Existen dos tipos de servicios: servicios de aplicación (ofrecidos por los agentes de interfaz), y servicios de integración (ofrecidos por los agentes de usuarios). Además, se asume que las aplicaciones dentro de una empresa se interconectan por medio de una red LAN (*Local Area Network*), y entre diferentes empresas la conexión es a través de una red WAN (*Wide Area Network*) (ver figura 3.9).

Cara	Agente	Servicios
Empresa	Agente de Interfaz: Mantiene la autonomía de una aplicación de la empresa y por lo tanto, conoce los protocolos a través de los cuales esta aplicación acepta requerimientos y devuelve resultados. De hecho, un agente de interfaz es el *proxy* de la aplicación	Servicio de Aplicación: Identifica una capacidad o aptitud de una aplicación. Es decir, informa al mundo exterior sobre la(s) capacidad(es) que posee una aplicación
Usuario	Agente de Usuario: Actúa en nombre del usuario. Es decir, asiste al usuario con el propósito de satisfacer los requerimientos de éste. Cada agente de usuario tiene la capacidad de crear un agente delegado, y enviarlo al (o los) SMA(s) apropiado(s). Esto significa que los agentes delegados son mobiles	Servicio de Integración: Es ofrecido a los usuarios, quienes intentan integrar varias aplicaciones con el fin de satisfacer sus necesidades. Por lo tanto, el cumplimiento de un servicio de integración involucra múltiples servicios de aplicación. En este enfoque, cada servicio de integración es asociado con un Workflow (WF)

Tabla 3.6: Caras, agentes y servicios del enfoque propuesto en [38]

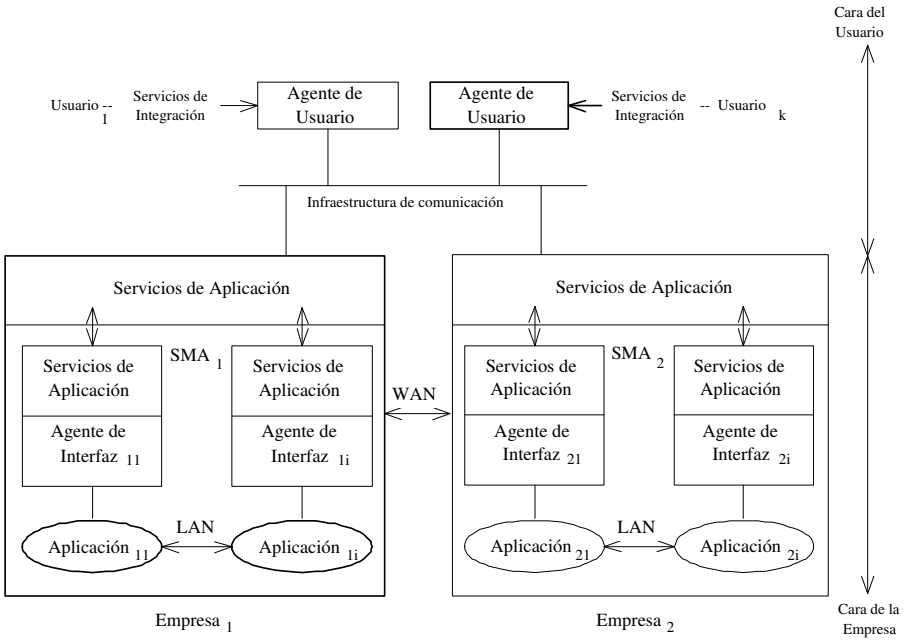


Figura 3.9: Arquitectura de la infraestructura para la IAE. Extraído de [38]

La infraestructura funciona de la manera siguiente: una vez que se inicia un servicio de integración, su agente de usuario utiliza el *workflow* (WF) apropiado para éste servicio. Después el agente de usuario ejecuta remotamente este proceso de WF en colaboración con diversos agentes de interfaz.

La figura 3.10 ilustra un ejemplo de un WF que involucra tres aplicaciones de empresa (Aplicación11, Aplicación12, y Aplicación21) y dos SMA (SMA1 y SMA2). Las aplicaciones Aplicación11 y Aplicación12 pertenecen al SMA1 de la Empresa1 y la Aplicación21 al SMA2 de la Empresa2. Los números en la figura 3.10 corresponden al orden cronológico de las operaciones.

Una vez que el usuario invoca un servicio de integración, el agente de usuario crea un agente delegado. Después, el agente de usuario asigna al agente delegado el WF de servicio de integración. Este WF menciona que el agente delegado tiene que moverse al SMA1. Cuando el agente delegado llega al SMA1 (1), interactúa con el agente de interfaz11 (2) para ejecutar el proceso del WF en la Aplicación11 (3). Esto significa que es invocado el servicio de aplicación del agente de interfaz11. El progreso del WF menciona que otros procesos necesitan ejecutarse en la Aplicación12. Por lo tanto, el agente delegado interactúa remotamente, a través de la LAN (4), con el agente de interfaz12 (5). Además, el WF dice que se requiere de la Aplicación21 antes que el WF

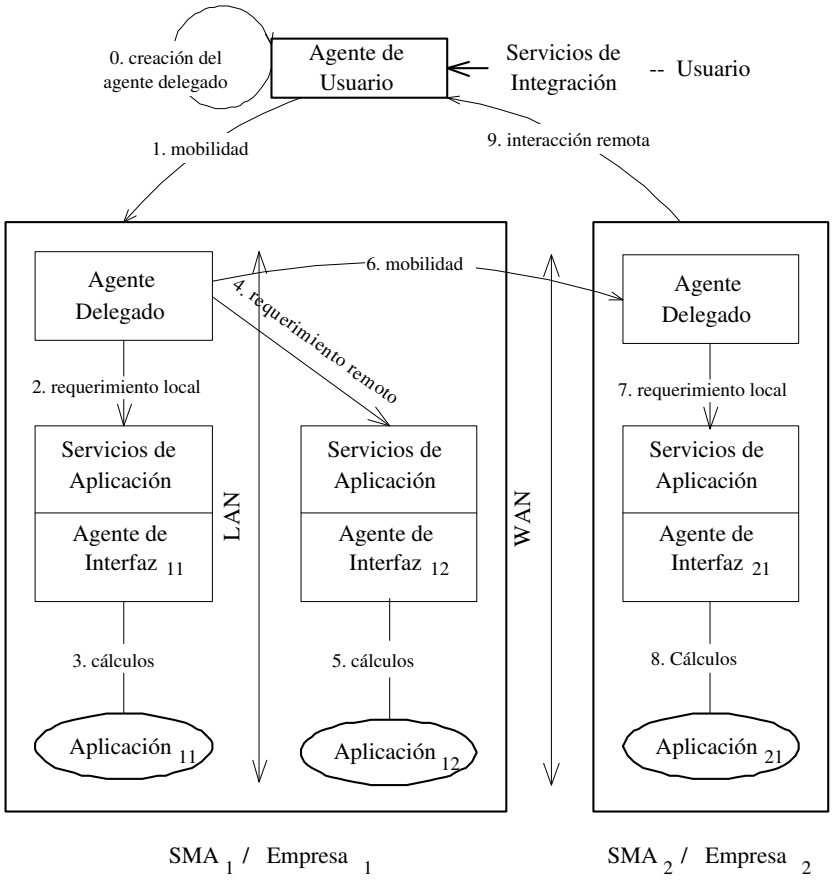


Figura 3.10: Funcionamiento de la infraestructura para la IAE. Extraído de [38]

pueda completarse totalmente. Por consiguiente, el agente delegado emigra al SMA2 (6) y se encuentra con el agente de interfaz21 (7,8). Entonces, se notifica al usuario sobre los resultados finales del servicio requerido (9). Al final, se elimina el agente delegado.

3.4. Conclusiones

Para una cantidad cada vez mayor de aplicaciones (incluso aquellas que tratan de lograr la integración), se necesitan sistemas flexibles y robustos que puedan decidir por sí mismos qué deben hacer para lograr sus objetivos de diseño. Una propuesta para ello son los agentes, que podemos definir como sistemas computacionales situados en algún entorno, y capaces de llevar a cabo acciones de forma autónoma en este entorno, destinadas a cumplir con los objetivos para los que fueron diseñados.

En este capítulo, se realizó una revisión exhaustiva de la bibliografía existente sobre agentes, así como también aquella que sirve como antecendente al estudio propuesto.

Capítulo 4

Propuesta de Solución

En este capítulo, se presenta una propuesta de solución al problema planteado en el capítulo 1. La misma, está basada en los argumentos teóricos expuestos en los capítulos 2 y 3. Con esta propuesta, se pretende dar una solución novedosa a la dificultad relacionada al proceso de integración de aplicaciones heterogéneas, pertenecientes a un ambiente de producción continua.

4.1. Modelado UML (*Unified Modelling Language*)

Un modelo es una descripción abstracta de un sistema o de un proceso, una representación simplificada que permite comprender y simular. El término modelado se emplea a menudo como sinónimo de análisis, es decir, de descomposición en elementos simples, más fáciles de comprender.

Con el fin de modelar el sistema propuesto se utiliza el Lenguaje Unificado de Modelado, UML [50]. Esta es una notación estándar para el modelado de sistemas de software o no, resultado de una propuesta de estandarización promovida por el consorcio OMG (*Object Management Group*).

UML es un lenguaje gráfico que permite visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. UML ofrece una forma de modelar cosas conceptuales como lo son procesos de negocios, y funciones de sistemas; además de cosas concretas, como lo son escribir clases en un lenguaje determinado, esquemas de base de datos, y componentes de software reusables.

El UML define una notación y un metamodelo. La notación es la parte fundamental que se ve en los modelos; es la sintaxis del lenguaje de modelado. Por ejemplo, la

notación del diagrama de clases define cómo se representan elementos y conceptos tales como clases, asociaciones y multiplicidad. Un metamodelo es el modelo de un modelo.En este caso representa el modelo orientado a objetos y lo define con rigor, en esencia en un diagrama (normalmente un diagrama de clases) que define la notación. En los anexos A y B se presentan los conceptos principales de la orientación a objetos y la notación básica UML para la creación de diagramas de clases, respectivamente.

Como se mencionó en el capítulo 1, el elemento principal en el Enfoque Jerárquico presentado por E. Chacón et. al. en [12], lo constituyen las Unidades de Producción (UPs) dado que un Complejo de Producción Continuo (CPC) mismo, puede verse como una UP. Es por ello, que el modelo de una UP sirve como base para el desarrollo del sistema integrador de aplicaciones que se pretende construir. A continuación, se describe dicho modelo.

4.2. Modelo UML para las Unidades de Producción

En la figura 4.1 se presenta el modelo de una UP genérica [10], representada por la clase **Unidad de Producción**, que está compuesta de otras UPs indicado por la relación **usa**, cuya composición se puede configurar de diferentes formas; la clase asociación **Configuración de la UP** representa esta capacidad. Cada UP tiene su propia configuración de los recursos que maneja, expresado a través de la relación **utiliza** que contiene la clase asociación **Configuración** con la clase **Recurso**. Las clases **Empresa** y **Complejo de Producción** son subclases de la UP, ya que ellas también se consideran UPs. De igual manera, las clase **Personal**, **Materia Prima**, **Servicio**, y **Equipo**, representan los tipos de recursos manejados por toda UP.

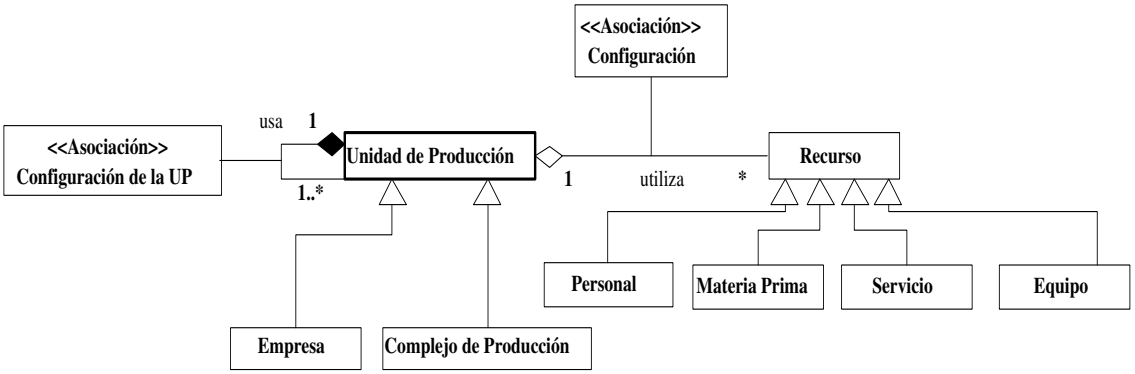


Figura 4.1: Diagrama de clases inicial de una UP. Extraído de [10]

El diagrama de clases inicial se extiende para indicar con más detalles los objetos y relaciones presentes en una UP. La figura 4.2 muestra este diagrama extendido, donde se considera una UP como un agregado de productos, capacidades, y misiones de la misma; todos ellos agrupados en las clase **Producto**, **Misión de la UP**, y **Capacidad de la UP**. Las relaciones entre estas clases representan el hecho que una UP produce

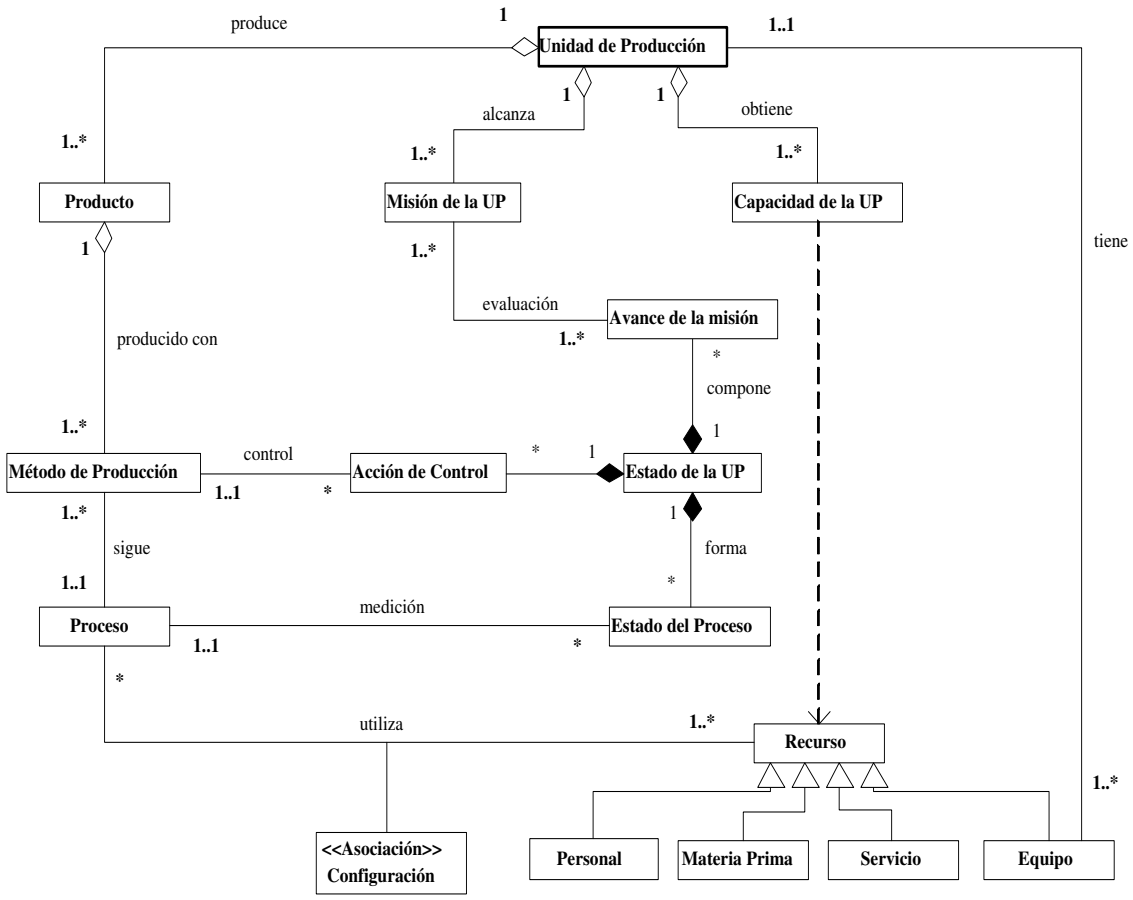


Figura 4.2: Diagrama de clases de una UP. Extraído de [10]

productos, para alcanzar una misión de la UP, basándose en un conjunto de capacidades de producción de la UP. Para la generación de productos se pueden utilizar diferentes métodos de producción agrupados en la clase **Métodos de Producción**, la cual está asociada con la clase **Producto** por la relación de agregación denominada **producido con**. La misión de una UP se evalúa para conocer qué tan cerca está la UP de la misma, la cual se mantiene en la clase **Avance de la Misión**. Por otro lado, se tiene una relación de dependencia entre la capacidad de la UP y sus recursos. Los métodos de producción pueden ser variados según una acción de control específica, seleccionada de acuerdo a

los estados posibles de la UP. Las clases **Acción de Control** y **Estado de la UP** soportan el manejo de ésta información. Finalmente, la clase **Proceso** contiene toda la información relevante sobre el mismo, el cual puede ser medido (sensar y observar) para obtener el estado de éste, que es colocado en la clase **Estado del Proceso**.

4.3. Modelo UML para el Sistema Integrado

En la figura 4.3 se presenta un diagrama de clases para la integración, el cual está basado en la Pirámide de Automatización propuesta en el capítulo 1. En este diagrama se muestra el hecho de que cada UP tiene asociada un conjunto de aplicaciones heterogéneas, que apoyan su funcionamiento y trabajan de manera independiente.

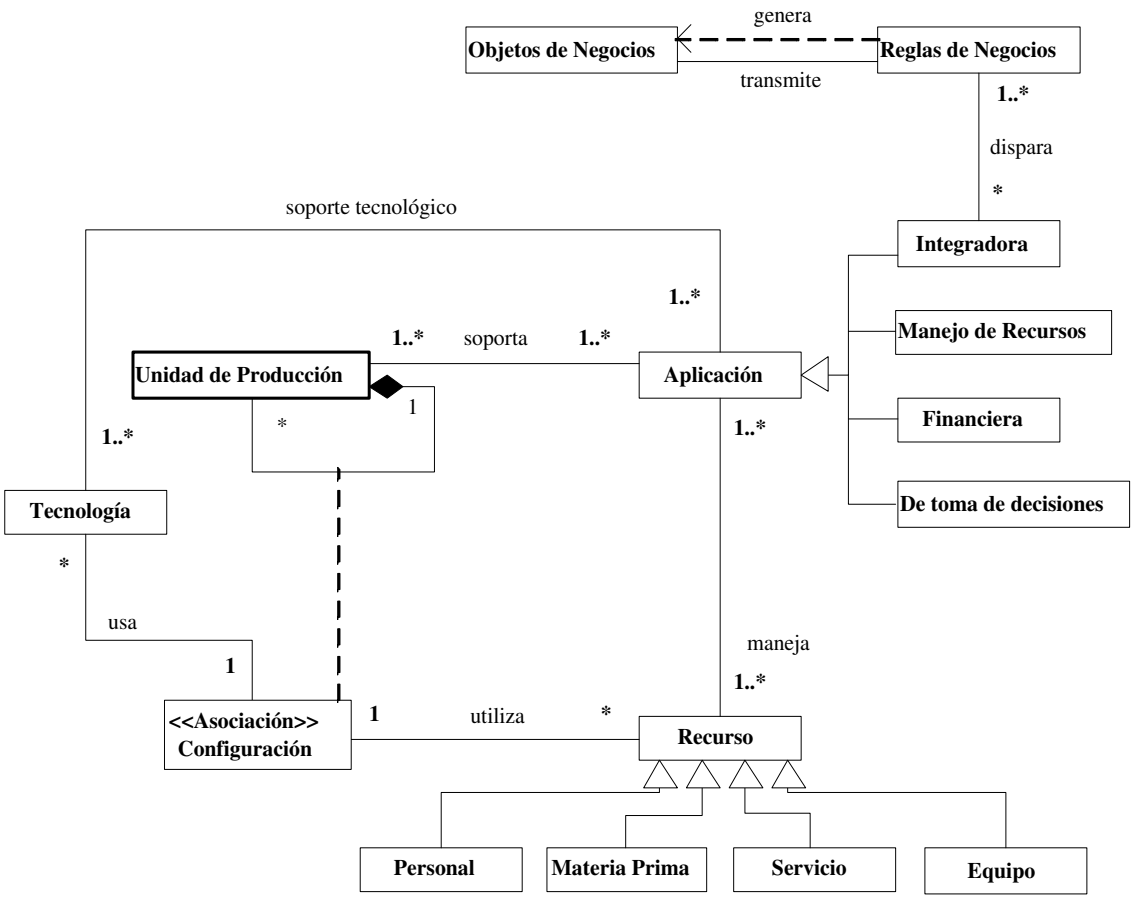


Figura 4.3: Diagrama de clase del sistema integrado. Extraído de [10]

El núcleo central de esta figura es la clase **Aplicación**, que se especializa en cuatro subclases, a saber: **De toma de decisiones**, **Financiera**, **Manejo de Recursos** e

Integradora. Esta última es la más importante, dado que ella soporta toda la información necesaria para la integración de las aplicaciones. Esta aplicación es capaz de soportar toda la lógica del negocio, que es representada a través de la clase **Reglas de Negocios**. En el diagrama, las reglas de negocios son accionadas y disparadas por la aplicación integradora. La clase **Objetos de Negocios** permite mantener la persistencia de aquellos objetos de negocios que se generan y utilizan en las reglas de negocios.

Por otra parte, la relación asociación **soporta** indica el hecho que de todas las aplicaciones que soporta una determinada UP, muchas de ellas pueden ser compartidas por diversas UPs. Similarmente, la relación **soporte tecnológico**, entre las clases **Aplicación** y **Tecnología**, significa que una aplicación puede ser soportada por muchas clases de tecnologías de información. Finalmente, la clase asociación **Configuración**, que relaciona las clases **Tecnología**, **Unidad de Producción**, y **Recursos**, es usada para registrar los diferentes tipos de configuraciones de la UP que utilizan las aplicaciones y los recursos.

4.4. Modelo UML para la Implementación del Sistema Integrado

El diagrama de integración, descrito en la sección anterior, constituye la base para el desarrollo de un modelo de implementación del sistema integrado. El mismo es extendido en la porción que involucra el proceso de toma de decisiones de la UP, es decir, aquel que implica las diversas aplicaciones heterogéneas de una UP. Todo lo anterior, con el fin de mostrar características y relaciones relevantes concernientes a la integración de las mismas. El diagrama de clases extendido para la implementación del sistema integrado se muestra en la figura 4.4.

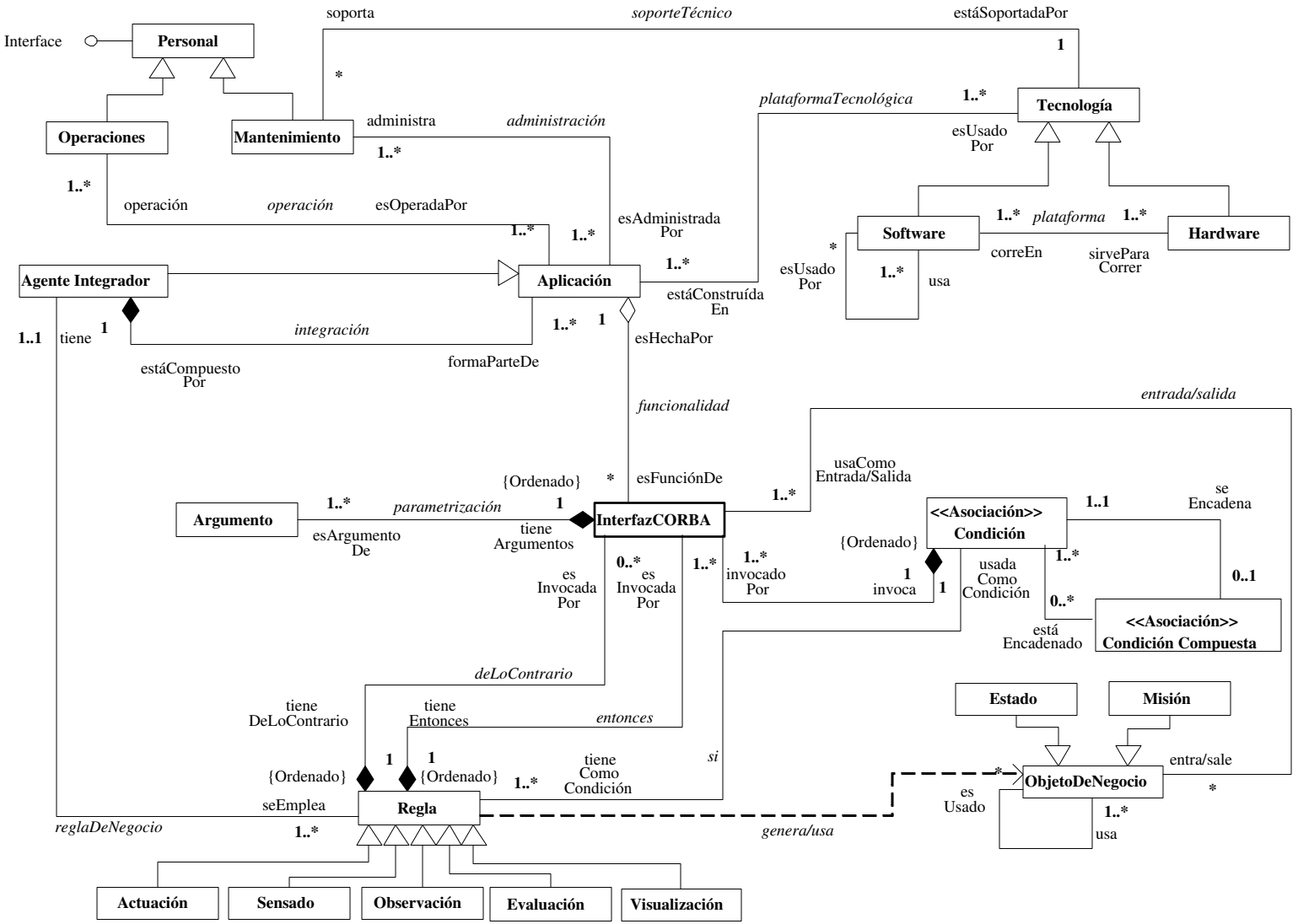


Figura 4.4: Diagrama de clase para la implementación del sistema integrado.

Como se mencionó en la sección anterior, la clase central del modelo de integración es la clase **Aplicación**. La idea de implementación consiste en registrar todas las aplicaciones existentes de la empresa[1], y construir una interfaz CORBA para cada una de ellas, para con esto lograr la integración de dichas aplicaciones. La aplicación especial llamada **integradora** es ahora denominada **Agente Integrador**, y constituye el mecanismo responsable de la sincronización, mantenimiento, y control de todas las aplicaciones integradas. Por último, cabe mencionar, nuevamente, que la lógica del negocio se representa por medio de las reglas y objetos de negocios.

Por ser este diagrama de implementación el elemento base para la propuesta de solución que se plantea, a continuación se describen cada una de sus clases y relaciones.

4.4.1. Clases

Las clases presentes en el diagrama de implementación son las siguientes:

Personal

Constituye una superclase que engloba a todo el personal de la UP. Posee dos subclases, a saber:

- **Mantenimiento**. Permite la administración del agente integrador, modificación de las reglas y objetos de negocios, y la asignación de la permisología necesaria para tener acceso al sistema.
- **Operaciones**. Encargada de definir las interfaces de los usuarios del agente integrador que poseen características de tipo gerenciales y operacionales. Dichas características son las de uso cotidiano de las aplicaciones de la empresa.

También esta superclase encapsula la **Interfaz** general para cualquier personal, asociado con alguna de las tareas descritas en las dos subclases reseñadas.

Tecnología

Superclase que soporta la manipulación de información referente a las diversas tecnologías utilizadas en la UP. Tiene dos subclase: **Software** y **Hardware**. Ellas permiten el manejo de los requerimientos de software y hardware, respectivamente, de las tecnologías presentes en la UP.

[1]Recuérdese que una empresa puede verse como una UP

Aplicación

Esta superclase contiene el registro todas las aplicaciones usadas en la UP, ya sean sistemas de información, sistemas de producción, o software especializado para el control, supervisión, y planificación.

Agente Integrador

Subclase de la clase **Aplicación**, que permite manipular la información del sistema integrado. A través de ella, se puede modificar las reglas y objetos de negocio.

InterfazCORBA

Es una clase que registra el conjunto de funciones disponibles para cada aplicación de la UP. Cada función es descrita a través de una interfaz CORBA (implantada en IDL). Estas funciones permiten aplicar las reglas de negocio desde el agente integrador.

Argumento

Clase que contiene los parámetros de cada función implantada en IDL (CORBA).

Regla

Permite la implantación de la lógica de negocio. Es una superclase que permite el manejo de los diferentes tipos de reglas de negocios del sistema integrado. Esta superclase se ha especializado de la siguiente manera:

- **Sensado**. Subclase para el manejo de las reglas de sensado correspondientes a los sensores del agente integrado. Los sensores son aplicaciones que pueden medir las variables necesarias para determinar el estado de un proceso.

- **Observación**. Subclase de las reglas de observación correspondientes a los observadores del agente integrador. Los observadores son aplicaciones que pueden calcular el estado de los procesos ejecutados por las aplicaciones.

- **Evaluación**. La evaluación exige tener el conocimiento del proceso y de los objetivos o misión que deben ser cumplidos por la UP. Se mide el progreso (separación de los objetivos), con el fin de determinar el curso de las nuevas acciones y así

mejorar el progreso o continuar con los mismos objetivos. Esta clase se incluye para dar soporte a lo anteriormente descrito.

- **Actuación**. Subclase que agrupa las reglas necesarias para modificar un proceso, con el fin de mejorar el rendimiento.
- **Visualización**. Subclase que permite formar las reglas para la visualización de los objetos y reglas de negocios, así como de cualquier otro objeto del sistema de negocios. Son aquellas reglas que permiten la extracción de la información que alguna aplicación solicite, para mostrar por pantalla o por cualquier salida estándar.

ObjetoDeNegocio

Superclase que permite crear y mantener los objetos de negocios, a través de uso de las reglas por parte de agente integrador.

4.4.2. Clases Asociación

En el modelo están presentes algunas clases del tipo asociación. Estas se hallan relacionando, principalmente, a aquellos elementos que representan las partes de las reglas de negocios. Ellas se describen a continuación.

Condición

Es la clase que agrupa la condición de disparo de una regla que puede o no tener funciones, de las aplicaciones, para ser disparadas.

Condición Compuesta

Clase utilizada para encadenar los átomos de la condición de las reglas de negocios. Para lograrlo, hace uso de operadores lógicos (y, o, not).

4.4.3. Relaciones de Asociación

soporteTécnico

Clases involucradas: **Mantenimiento** y **Tecnología**.

Indica quién es el responsable del soporte técnico de las diferentes tecnologías.

- Una persona de mantenimiento puede brindar soporte técnico a uno o varias tecnologías
- Una tecnología puede ser soportada por una o varias personas de mantenimiento

administración

Clases involucradas: **Mantenimiento** y **Aplicación**.

Asocia las aplicaciones con el personal responsable de su administración (mantenimiento, actualización, etc).

- Una persona de mantenimiento puede administrar una o varias aplicaciones
- Una aplicación puede ser administrada por una o varias personas de mantenimento

operación

Clases involucradas: **Operaciones** y **Aplicación**.

plataformaTecnológica

Clases involucradas: **Aplicación** y **Tecnología**.

Asocia las tecnologías de implementación con las aplicaciones

- Una aplicación está construida sobre a una o más tecnologías
- Una tecnología puede ser usada por una o varias aplicaciones.

plataforma

Clases involucradas: **Software** y **Hardware**.

Asocia los componentes de hardware sobre los que se ejecutan los paquetes de software.

- Un software se ejecuta sobre una o varias plataformas de hardware
- Un hardware sirve para ejecutar uno o más paquetes de software

reglaDeNegocio

Clases involucradas: **AgenteIntegrador** y **Regla**.

Relaciona el conjunto de reglas de negocios de las aplicaciones que se hayan integradas.

- El agente integrador de aplicaciones de una UP tiene una o más reglas de negocio
- Las reglas de negocio son empleadas por el agente integrador con el objeto de integrar las aplicaciones

entrada/salida

Clases involucradas: **InterfazCORBA** y **ObjetoDeNegocio**.

Controla los objetos utilizados o generados por las funcionalidades de las aplicaciones

- Una función de una aplicación puede utilizar como entrada/salida ninguno o varios objetos de negocio
- Un objeto de negocio puede ser utilizado o generado por una o varias funcionalidades de aplicaciones

4.4.4. Relaciones Reflexivas de Asociación

usa (Clase Software)

La clase **Software** se asocia con ella misma para indicar que un paquete de software puede utilizar o no, o correr en otro software

usa (Clase ObjetoDeNegocio)

La asociación reflexiva de la clase **ObjetoDeNegocio** muestra que un objeto de negocio puede o no utilizar otro objeto de negocio.

4.4.5. Relaciones de Agregación

En el diagrama se muestra un sola relación de agregación normal: la agregación ***funcionalidad***, en el que la clase **Aplicación** constituye la clase contenedora de ninguna o varias funcionalidades, representadas por medio de la clase **InterfazCORBA**.

4.4.6. Relaciones de Agregación Compuesta o Composición

En la tabla 4.1 se muestran las relaciones de composición que se visualizan en el diagrama de integración.

Composición	Clase Todo	Clase Parte	Descripción
integración	AgenteIntegrador	Aplicación	Un agente integrador está compuesto de una o más aplicaciones, las cuales debe integrar
parámetros	InterfazCORBA	Argumento	Una función de una aplicación, que es descrita a través de su interfaz CORBA tiene uno o más argumentos como parámetros. Un argumento es parte, o parámetro, de ninguna o varias funciones
condición	Condición	InterfazCORBA	Las reglas de negocios involucran la invocación de un conjunto de funciones de las aplicaciones integradas, como parte de la condición para el disparo de la regla
entonces	Regla	InterfazCORBA	Las reglas de negocios involucran la invocación de un conjunto de funciones de las aplicaciones integradas, como parte del consecuente del disparo de la regla
deLoContrario	Regla	InterfazCORBA	Las reglas de negocios involucran la invocación de un conjunto de funciones de las aplicaciones integradas, como parte de la acción contraría que se ha de realizar en caso de que no se cumpla las condiciones del disparo de la regla

Tabla 4.1: Relaciones de composición del diagrama de implementación de la figura 4.4

4.5. El Agente Integrador de Aplicaciones Heterogéneas

En la presente sección, y en las subsecuentes, se describen las características y el funcionamiento del agente integrador que se propone, como posible solución, al problema de la integración de aplicaciones presentes en una UP.

4.5.1. Estructura General del Sistema a Integrar

En la figura 4.5 se muestra una vista del sistema que se desea integrar. Un CPC puede ser considerado como un sistema complejo -una UP-, de naturaleza jerárquica, compuesto por otros subsistemas (UPs) relacionados entre sí, y con interacciones poco frecuentes. Por otra parte, cada subsistema está constituido por un número limitado de componentes con interacciones frecuentes. Estos componentes, que a su vez también son UPs, poseen aplicaciones que gestionan los procesos de negocios, y son las que se desean integrar.

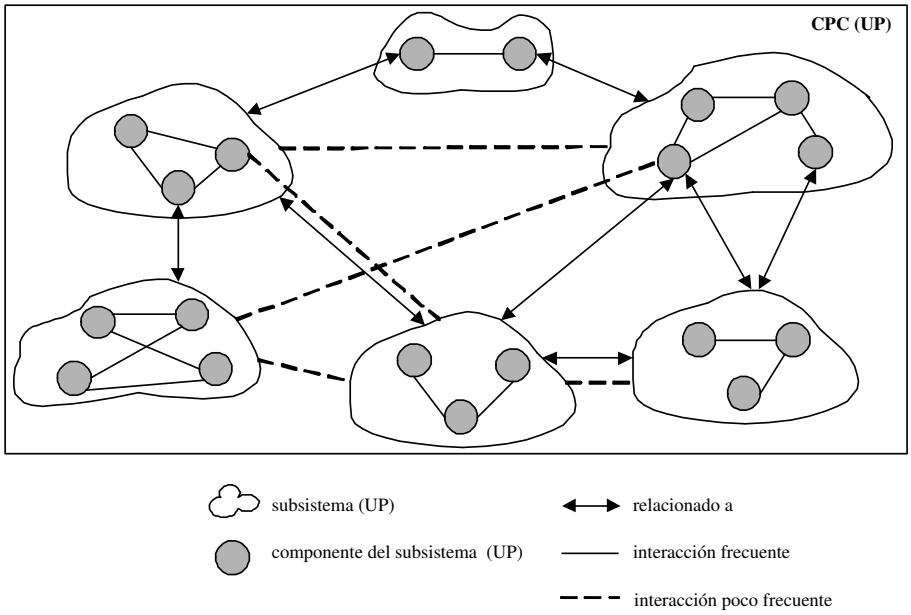


Figura 4.5: Vista del sistema a integrar

4.5.2. Estructura General de la Solución Basada en Agentes

En la figura 4.6 se presenta una vista de la solución orientada a agentes para la integración del sistema dado a conocer en 4.5.1. En ella se muestra a cada componente (UP) de un subsistema (UP) representado por un agente, que integra las aplicaciones

presentes en la UP correspondiente. Estas aplicaciones son heterogéneas, puesto que cada una de ellas pueden estar construidas sobre distintas plataformas de hardware y software.

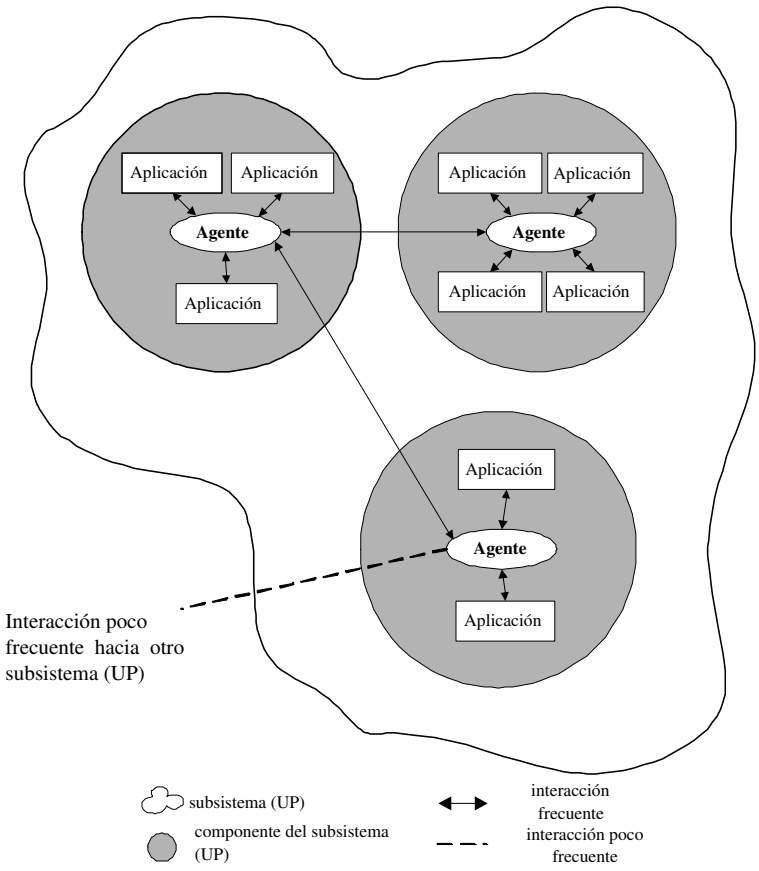


Figura 4.6: Vista de la solución basada en agentes

Se necesita poseer un mecanismo de manera que las aplicaciones, aún siendo heterogéneas, tengan la capacidad de compartir información, referente a los procesos de negocios que se ejecuten dentro de cada UP. El agente (en adelante llamado ***Agente Integrador*** - **AI**) que representa a cada UP, será el encargado de lograr éste objetivo. Los agentes de las UPs cumplen con dos funciones principales (ver figura 4.7):

- *Interna*, integrando las aplicaciones de la UP que representan, tomando en cuenta el modelo de negocios.
- *Externa*, permitiendo el intercambio de información con otras UPs o usuarios externos.

La estructura propuesta es recursiva, es decir, puede ser aplicada dentro de todos

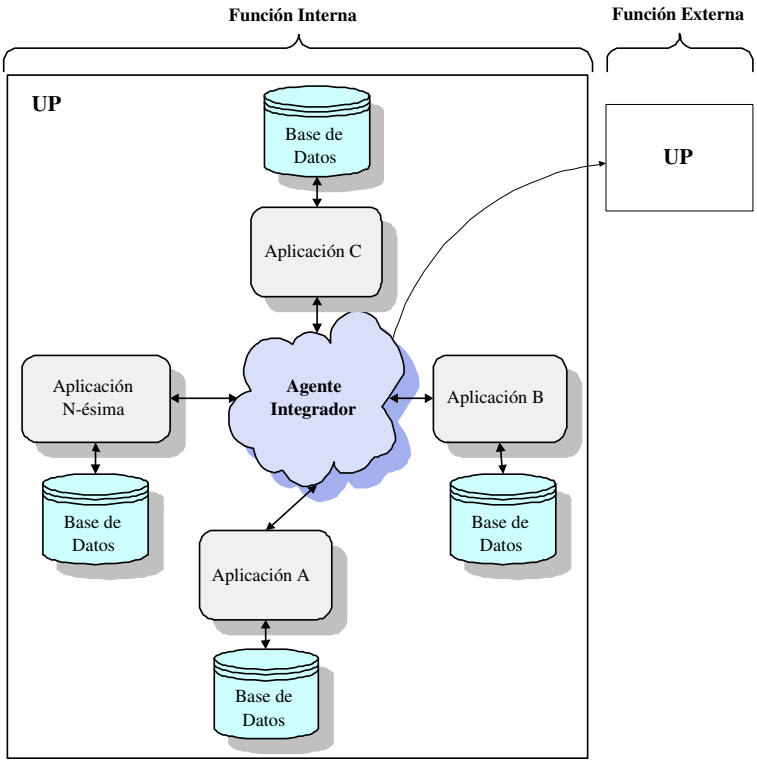


Figura 4.7: Funciones principales del Agente Integrador

los subsistemas y componentes que conforman un CPC puesto que, como se mencionó anteriormente, todos ellos pueden ser considerados como UPs. Un ejemplo de cómo es la integración basada en agentes, mostrada desde el punto de vista de la arquitectura organizacional de la empresa, se muestra en la figura 4.8. En ella se observa que subsistemas de los diferentes niveles de la organización, están representados por un AI que es el encargado de integrar la información que se maneja dentro de ellos. El intercambio de información se realiza dentro y entre los niveles a través de los AIs, logrando de esta manera tanto la integración horizontal como vertical, respectivamente.

¿Por qué utilizar agentes?

Son varias las bondades ofrecidas por las soluciones orientadas a agentes [23]. Sin embargo, en el presente contexto, las siguientes razones son las que motivan el uso de agentes:

- *Capacidad de actuar sin la intervención directa de una persona o de otro agente*: Un agente debe poder controlar sus propias acciones y estado interno. Una vez que el usuario activa el agente indicando algún objetivo de alto nivel, éste debe

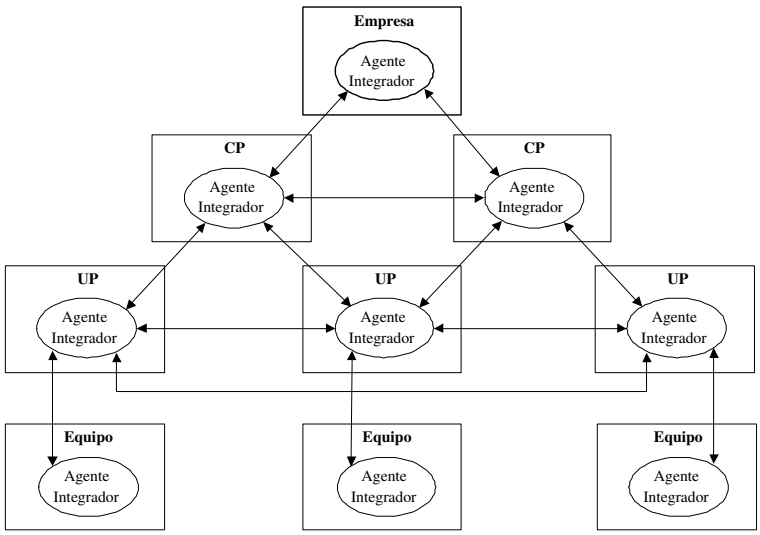


Figura 4.8: Vista de la solución basada en agentes para la arquitectura organizacional de la empresa

actuar independientemente, seleccionando estrategias y monitoreando el progreso en busca de sus objetivos. Si falla alguna estrategia, buscará otra, pero sin intervención humana o con la mínima indispensable.

- *Capacidad de comunicación*: Un agente debe tener habilidad para interactuar con otros agentes o incluso con alguna persona, para solicitar información o bien para exponer los resultados obtenidos en la ejecución de sus tareas, a través de algún tipo de interface.

- *Capacidad de reactividad*: Un agente debe poder sensar el estado del ambiente en el que se encuentra inmerso y- en función de esto- actuar, respondiendo de manera adecuada a cambios producidos en el mismo. Los efectos producidos pueden modificar el estado de su entorno.

- *Orientado a objetivos*: Un agente no sólo debe actuar por cambios detectados en el medio ambiente, sino que -además- debe "trabajar" en función de los objetivos para los cuales fue diseñado.

- *Continuidad temporal*: Un agente es un proceso temporalmente continuo. A diferencia de un programa convencional del cual se conoce su inicio y fin, un agente debe ejecutarse hasta que se haya alcanzado con el conjunto de objetivos solicitados, o bien mientras el ciclo perdure y su usuario desee detenerlo. La continuidad

temporal es la propiedad que da "vida´´ al agente, haciendo posible que se mantenga alerta a una solicitud o a algún cambio en el medio ambiente. El ciclo de vida de un agente depende de sus características, de las tareas que realice y de los deseos de su usuario en cuanto al tiempo durante el cual el agente debe ejecutarse.

- *Consistencia con el paradigma orientado a objetos*: La eficiencia de la Programación Orientada a Agentes (POA) comienza con la eficiencia del paradigma de orientación a objetos. La POA es considerada por Y. Shoam [32] como una especialización de la Programación Orientada a Objetos (POO). Los agentes y los objetos pueden ser usados en la misma interfaz [40], pero se distinguen uno del otro; los objetos pueden mantenerse como están: estáticos, predecibles, se mantienen justo en el sitio. Los agentes son repositorios funcionalmente adaptativos: pueden notar eventos, usar reglas para interpretarlos, y tomar acciones basadas en esas interpretaciones.

4.5.3. Características del Agente Integrador

El AI encargado de integrar las aplicaciones de una UP, posee las siguientes características:

- Está ubicado en un ambiente particular sobre el que tiene control parcial y observabilidad: recibe entradas relacionadas con el estado de su ambiente a través de sensores y actúan sobre el ambiente a través de efectores. Es decir, es un *agente reactivo*, ya que debe ser capaz de responder de manera rápida a los cambios que ocurren en su ambiente.

- Está diseñado para cumplir un objetivo específico: integrar aplicaciones.

- Es semi-autónomo pues su actuación, dentro de la UP, está basada en un conjunto de reglas predeterminadas. Esto implica que ellos no deciden autónomamente si ejecutan o no los requerimientos del sistema. Sin embargo, hay autonomía externa debido a que la actuación del AI de una UP específica, es distinta a la de cualquier otro AI.

- Es deliberativo, puesto que un método o acción del agente se ejecuta, a través de una maquina de inferencia, tomando en cuenta una base de conocimiento referente

al estado del sistema en que se encuentra inmerso.

- Coopera con otros agentes para lograr metas. El agente nunca rechaza deliberadamente la cooperación con otros agentes. Sólo cuando las acciones requeridas sean imposibles o desventajosas, él rechazará su ejecución.

4.5.4. Arquitecturas del Agente Integrador

En el entorno que se presenta, el AI es una entidad autónoma -en el sentido de que funciona independientemente de otros agentes- y cognitiva, con capacidades de conocimiento y comunicación, y cuyo comportamiento está dirigido por unas reglas que restringen su autonomía interna.

En la figura 4.9, se da a conocer la estructura general del Agente Integrador de Aplicaciones Heterogéneas que se propone. En esta figura se observa que el AI está compuesto de niveles. En su primer nivel, el AI cuenta con sensores que perciben los eventos generados por las aplicaciones. Estas medidas actualizan los modelos que el agente posee sobre sí mismo, y sobre el entorno. Es decir, el AI tiene un modelo de sí mismo con el fin de resolver problemas locales (*modelo propio*), y otro de sólo con los que interactúa (*modelo de conocidos*). Dichos modelos constituyen las creencias (segundo nivel) del AI, que es la manera como él cree que están sus entornos, tanto interno como externo.

En un tercer nivel (razonamiento), el AI también posee un interpretador que, basado en los modelos y las capacidades del agente, instancia un plan que se ejecuta en un hilo independiente, puesto que el AI necesita despachar sus acciones y seguir sensando el entorno. Un plan instanciado puede implicar acciones sobre objetos (nivel de acciones), o la colaboración con otros agentes que están interesados en la información que se maneja dentro de la UP (nivel de colaboración).

A continuación se presentan diferentes arquitecturas, con el fin de explicar tanto la estructura funcional como estructura de software del AI.

Arquitectura Funcional del Agente Integrador

El agente necesita tomar decisiones que logren integrar la información manejada por las aplicaciones presentes en cada UP. Esto lo hace en base al presente, sin ninguna

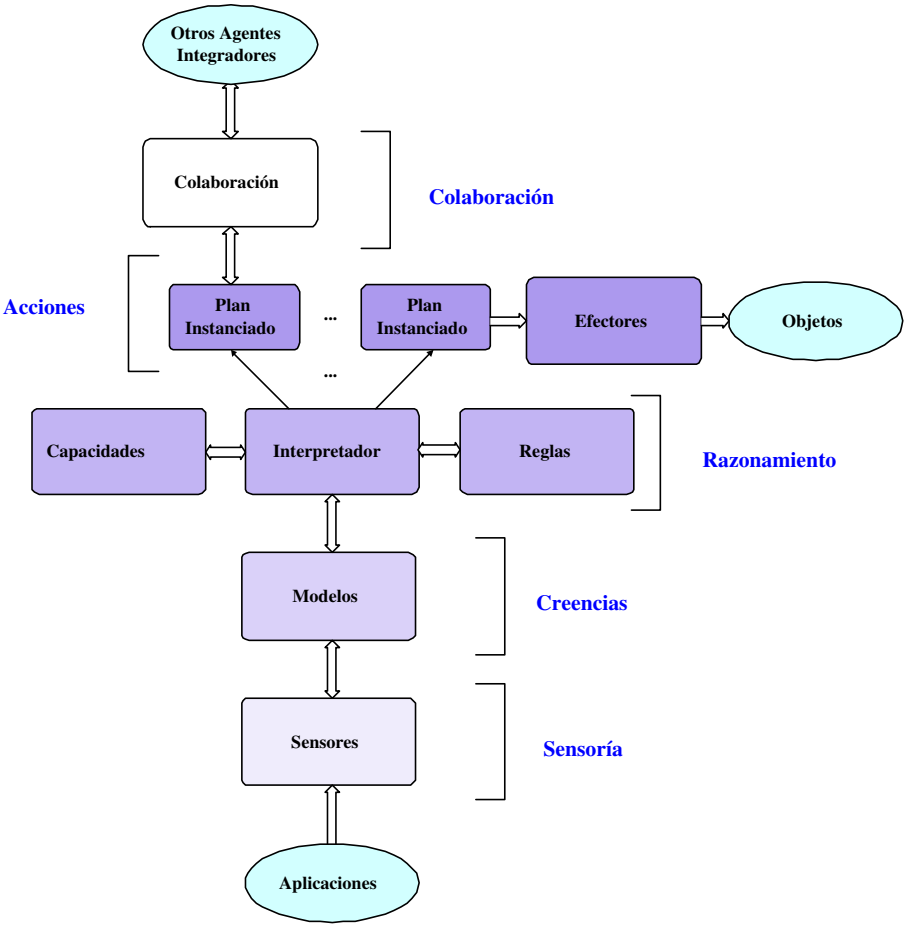


Figura 4.9: Arquitectura General del Agente Integrador de Aplicaciones

referencia a su pasado. Es decir, responde directamente a su ambiente y, por lo tanto, es un agente reactivo, un agente estándar.

El agente también tiene carácter deliberativo, pues funciona en base a un conjunto de reglas preestablecidas. Con el fin de mostrar este rasgo, se incorpora un motor de inferencia que aporta a las características deliberativas del sistema. El motor de inferencia permite definir el agente de forma declarativa, el conocimiento del mismo se define mediante hechos y el comportamiento mediante reglas. La plataforma plantea un modelo de agente inteligente al cual hay que ajustarse.

Por presentar características reactivas y deliberativas, el AI puede ser visto como un *agente híbrido*. En la figura 4.10 se muestra el comportamiento del agente (los elementos denotados por una elipse representan las funcionalidades propias del AI).

En ésta figura se refleja el hecho que el AI es invocado por eventos provenientes de las aplicaciones, o él puede ***sensar*** los mismos cada cierto período de tiempo a fin de

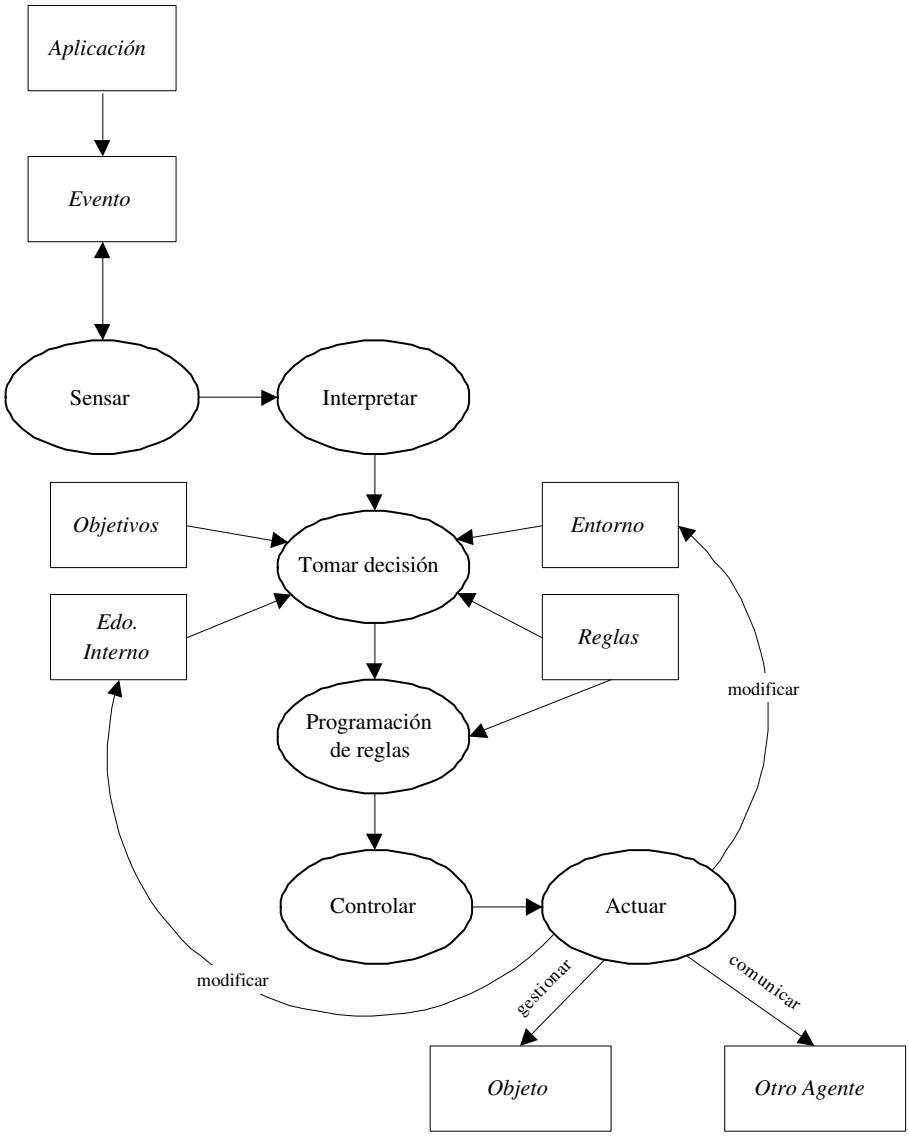


Figura 4.10: Funcionamiento del Agente Integrador

determinar si alguna aplicación requiere de un servicio. El AI maneja los eventos de acuerdo a prioridades preestablecidas, de manera que pueda conocer cuál o cuáles debe procesar primero. Dependiendo del tipo de evento, el AI prioriza las tareas a ejecutar:

- *Eventos críticos*: deben manipularse en cierta fracción de tiempo. Esto garantiza la reactividad del AI.
- *Eventos no-críticos*: son las tareas normales que ejecuta el AI
- *Eventos opcionales*: funcionalidades adicionales como monitoreo o diagnósticos.

.

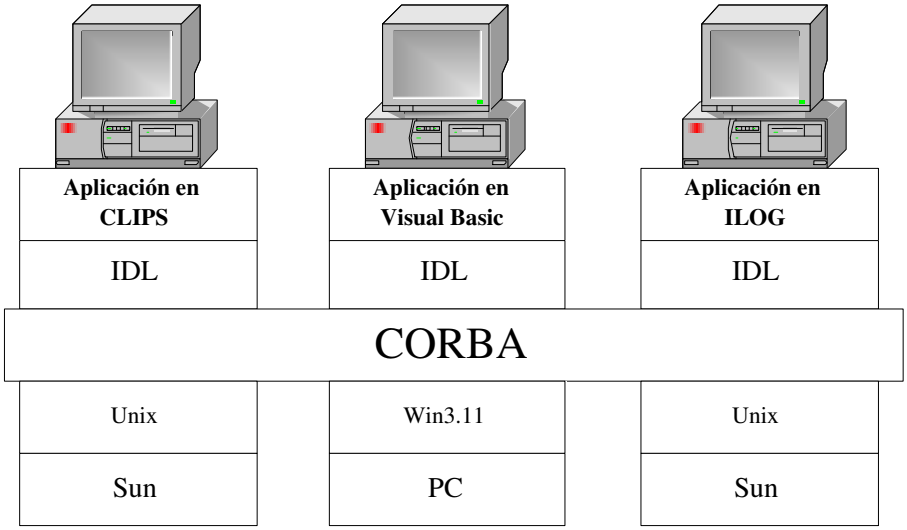


Figura 4.11: Ejemplo de la plataforma de integración

Después que el AI ***interpreta*** los eventos, procede a tomar decisiones basado en un conjunto de reglas, su(s) objetivo(s), estado interno (valores de sus atributos), y entorno (conocimiento representado a través de hechos). De acuerdo con los elementos anteriores, el AI ***programa las reglas*** a disparar. Este subconjunto de reglas constituyen un plan instanciado, que es ***controlado*** por el AI. El disparo de una regla o varias reglas encadenadas implica la actuación del agente sobre:

- *Un Objeto*: Gestión (crear, modificar o eliminar) sobre un Objeto de Negocio
- *Otro Agente*: Comunicación con un agente distinto, pues el AI tiene conocimiento (una lista) de los agentes que están interesados en la información que se está generando en su UP.
- *El Entorno*: Modificación de la base de hechos que representa el conocimiento que el AI tiene sobre el entorno en el que se haya inmerso.
- *Su Estado Interno*: Modificación de los atributos propios del AI

En la figura 4.11 se muestra un ejemplo de la plataforma de integración. La misma se logra a través de CORBA, dado que ésta soporta la distribución de objetos a través de diferentes arquitecturas de máquinas y sistemas operativos. A cada una de las aplicaciones, pertenecientes a la UP, se le registra su interfaz CORBA (IDL) referente a las funcionalidades que tiene la misma y son las que en realidad generan los eventos de las

aplicaciones para notificar al AI que algo importante está ocurriendo dentro de ellas. De esta manera los datos que gestiona una aplicación no son tratados directamente por el AI, sino que son manejados por las mismas aplicaciones.

Arquitectura en Capas del Agente Integrador

En la figura 4.12 se observa la arquitectura en capas del AI. Ésta arquitectura permite enfocar la relación existente entre los módulos del software que forman parte de un AI, quien debe ser descompuesto en capas puesto que:

1. El comportamiento del AI en los niveles superiores depende de las capacidades de los niveles inferiores.

2. Un nivel sólo depende de sus niveles adyacentes.

Figura 4.12: Arquitectura en Capas del Agente Integrador

A continuación se describen cada uno de las capas presentes en la mencionada arquitectura (tabla 4.2):

Capa	Nombre	Descripción
5	Colaboración	Verifica y dirige los mensajes a otros agentes
4	Acción	Ejecuta el plan instanciado en un hilo
3	Razonamiento	Procesa las creencias para determinar qué se debería hacer; instancia un plan
2	Creencias	Almacena las creencias del agente; actualiza las creencias de acuerdo a las entradas de los sensores
1	Sensoria	Determina cambios en el ambiente; actualiza mensajes

Tabla 4.2: Descripción de las capas de software del Agente Integrador

4.5.5. Modelado de las Reglas y Objetos de Negocios

Como se mencionó en los apartados anteriores, el AI funciona en base a un conjunto de reglas preestablecidas. Dichas reglas conforman la representación de fragmentos de la lógica del negocio; es decir, constituyen las directrices del negocio y, de acuerdo a ellas, se genera o transfiere la información de un punto a otro dentro del sistema integrado.

Estas reglas, que son las Reglas de Negocios (RN), pueden ser estructuradas bajo el paradigma Evento-Condición-Acción (ECA), como en las bases de datos activas. En el modelo que se presenta, estas son organizadas a través de reglas Evento-Condición-Acción-Acción (ECAA), las cuales se pueden transformadas en una o más reglas ECA negando la condición.

La semántica de las reglas ECA es como sigue:

- *Evento*: Cuándo una regla de negocio debe ser procesada
- *Condición*: Qué debe ser verificado
- *Entonces-Acción*: Qué hacer si la condición es cierta
- *En caso contrario-Acción*: Qué hacer si la condición es falsa

Generalmente, dentro de las aplicaciones se haya reflejada la lógica del negocio. Esto conlleva a la modificación de la lógica de programación de las aplicaciones cuando sea necesario cambiar la lógica del negocio, lo que requiere de un experto en caso de que las aplicaciones permitan las modificaciones requeridas. Construir reglas directamente en las aplicaciones (si éstas lo permiten) añade las complejidades de programación y pruebas, reduciendo la flexibilidad del sistema.

En el contexto de esta propuesta, las RNs se modelan fuera de las aplicaciones, con el fin de automatizarlas más que implementarlas a través de métodos de programación tradicional. En este punto, es importante aclarar que el conjunto de reglas que se automatizan son las llamadas reglas de reacción [52] o de flujo [51] (ver sección 3.2.3), quienes determinan cómo fluye la información a través del sistema, en presencia de determinados eventos.

El hecho de tener las RNs aparte tiene sus ventajas, entre ellas se mencionan:

- Permite modificar la lógica del negocio sin temor a la "ruptura" del resto de la aplicación puesto que ambas lógicas, la del negocio y la de programación, se encuentran separada.
- Mejor adaptabilidad del negocio, dada su constante evolución.
- Aumento de la flexibilidad, debido a que no se requiere de cambios en las aplicaciones; es decir, la infraestructura de aplicaciones permanece intacta.
- Las reglas pueden ser compartidas por muchas aplicaciones.

En lo que respecta a los ONs, ellos son utilizados por las aplicaciones para la ejecutar la lógica del negocio. Sólo aquellos ONs útiles para las aplicaciones están registrados en el sistema. Ellos son creados y mantenidos en el sistema por la RNs.

Los ONs deben tener interfaces bien definidas, de manera que puedan ser implementados independientemente. Además, deben ser capaces de cambiar sus atributos, e interactuar con otros ONs.

4.5.6. Arquitectura Funcional del Sistema Integrado

En la presente sección se describe el funcionamiento general del sistema integrado. Éste puede ser visto como una población de agentes que interactúan entre sí con el fin

de compartir información. Cada agente representa a una UP del sistema, y él mismo también conforma una UP al interrelacionarse con entidades externas a la organización.

Dado que todo elemento del sistema integrado puede ser visto como una UP, en lo que sigue se presenta una descripción sencilla de su funcionamiento, pues ella misma constituye un elemento escalable, jerárquico y modular, que se puede extender al sistema completo.

Cada una de las aplicaciones pertenecientes a una UP se hayan registradas, y sus interfaces CORBA especificadas en IDL. Estas interfaces constituyen funcionalidades que son las que indican qué cosas hace o puede hacer la aplicación. De éste modo, los datos son manejados directamente por la aplicación misma.

Cuando ocurren cambios importantes en los procesos de negocios gestionados por las aplicaciones, las funcionalidades disparan eventos que informan al AI que algún acontecimiento importante ha sucedido, o está sucediendo, dentro de la aplicación. En el momento en que se produce un evento de interés, por parte de la aplicación, el AI es notificado y éste procede a chequear las RNs en base al evento y al conocimiento sobre sí mismo y el entorno.

Luego de proporcionarle la información inicial, el AI selecciona las reglas apropiadas a disparar, y las ejecuta encadenándolas, si es necesario, y controlando las acciones de los consecuentes de las reglas disparadas. Los consecuentes de las RNs también están construidos a partir de las funcionalidades de las aplicaciones, con el fin de poder compartir información entre ellas. También dichos consecuentes se pueden referir a la gestión de un ON - que es donde se refleja la información compartida entre las aplicaciones y relacionada con la ejecución de los procesos de negocios-, a la modificación del conocimiento que el agente tiene almacenado en sus modelos, o la comunicación con los AIs de otras UPs interesados en la información que se está produciendo.

Esta actuación se mantiene en todas las estructuras que representan UPs dentro del sistema, logrando esta manera la interoperatividad de las aplicaciones heterogéneas presentes.

4.6. Conclusiones

En la arquitectura propuesta se ha utilizado la arquitectura de agente híbrido para construir el AI. Los diferentes niveles de inteligencia y comportamiento son asociados

a dos tipos de agentes: reactivo y deliberativo.

La arquitectura general del sistema automatizado está organizada como una población de agentes (AIs) que integran las aplicaciones de las UPs. A través del AI las aplicaciones pueden localizar y "conversar" con otras aplicaciones.

Las aplicaciones interactúan a través de sus funcionalidades. Lo que se hace es interceptar las aplicaciones por medio de dichas funcionalidades, de modo que se pueda conocer cuando ocurren los hechos importantes dentro de ellas y así poderlos notificar al AI.

El sistema integrado cuenta con un conjunto de RNs asociadas, y esto permite la manipulación de los ONs que apoyan la ejecución de los procesos de negocios. El AI, a través de una máquina de inferencia, determina y ejecuta todo el flujo de control de las reglas necesitadas por las aplicaciones.

La estructura propuesta es recursiva y jerárquica, y puede ser aplicada a cada uno de los elementos constituyentes de los distintos niveles de una empresa.

Capítulo 5

Implementación de un Prototipo

En este capítulo se presenta un prototipo con el fin de comprobar la factibilidad y viabilidad del modelo del Agente Integrador propuesto en el capítulo 4. Una porción de la lógica del negocio de la empresa Aguas de Mérida, sirve como ejemplo de prueba para dicho prototipo.

5.1. Enfoque de prototipos

Un prototipo es un modelo a escala de un sistema de información que se desea desarrollar. Muestra las potencialidades del sistema antes de que éste sea construido, especialmente lo relacionado con la interfaz humano-computador; logrando así que el usuario aporte sugerencias que mejoren su interacción con el sistema y corroboren la especificación de requerimientos [9].

Un prototipo es una versión inicial de un producto de software, en la cual se describe la interacción humano-computador, sin implementar completamente la funcionalidad del sistema. Por lo tanto, es una representación de un sistema, que incorpora componentes del producto real; aunque no es un sistema completo, posee las características del sistema final o parte de ellas. Por lo regular, un prototipo tiene un funcionamiento limitado en cuanto a capacidades, confiabilidad o eficiencia [53].

5.1.1. Clases de Prototipos

J. Barrios en [9] define dos tipos de prototipos:

- *Desechable*, que sólo muestra la posible interfaz del sistema que se va a construir y se usa sólo para la captación de requerimientos y funcionalidad.

- *Utilizable* en el sistema real, el cual una vez cumplido su objetivo como prototipo, se refina, se ajusta y pasa a formar parte del sistema que se está construyendo.

Esta autora también define varias categorías de prototipos. Entre ellas se encuentran:

- *Automatizado de pantallas:* Permite moverse de una pantalla a otra simulando un sistema funcional, pero sin ejecutar verdaderamente ninguna actividad sobre los datos.

- *Automatizado funcional*: Es cuando al prototipo de pantallas se le agregan facilidades funcionales, dotándolo de procesos sobre los datos.

Complementario a estas divisiones en [53], se clasifican los prototipos en:

- *Vertical*: Desarrolla completamente algunas de las facetas del producto.

- *Horizontal*: Desarrolla parcialmente todas las facetas del producto.

5.1.2. Fases para el desarrollo de prototipos

La construcción de un prototipo incluye parte de la fase de definición de requerimientos y parte de la fase de diseño de un sistema de software (figura 5.1), ya que éste se relaciona directamente con la definición y el modelado de la interfaz humano-computador, no sólo en lo correspondiente a modos de interacción, sino en lo referente a las funciones que el sistema debe ejecutar.

La construcción de prototipos sigue las fases de desarrollo de sistemas de software, pero sin profundizar demasiado en los detalles que constituirían el sistema completo. Dichas fases son las siguientes:

Análisis - Requerimientos. En ésta fase se hace un estudio del sistema. Se hará la recopilación de requisitos, tanto del sistema como del software, se documenta todo lo que se ha estudiado y se establece lo que se va a hacer.

Diseño. Una vez que se sabe qué hay que hacer, en esta fase se determina cómo se va a hacer. Se diseña la arquitectura de los datos, la del software e interfaz.

Construcción. Es la traducción del diseño en un lenguaje de programación utilizando, además, herramientas de apoyo a la corrida para corregir errores en los programas construidos.

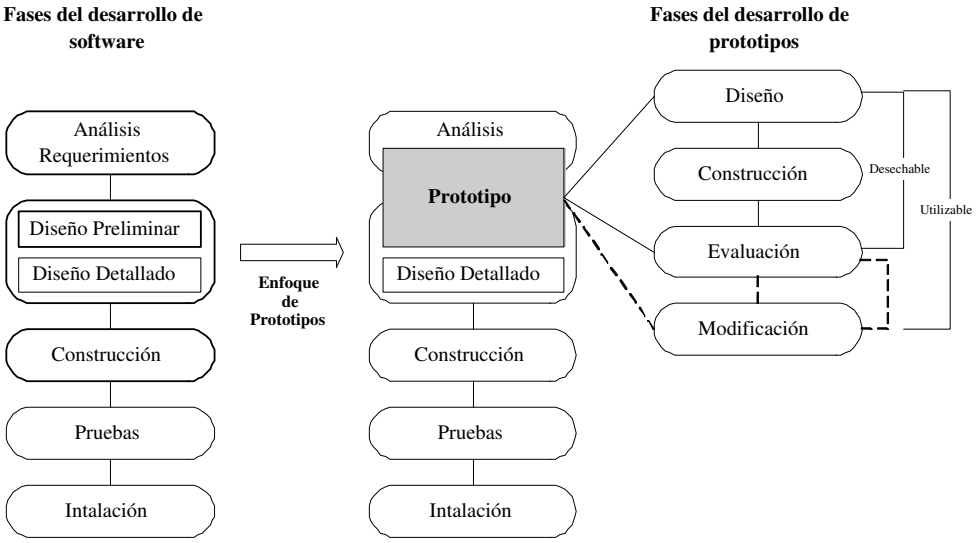


Figura 5.1: Fases de desarrollo de prototipos. Adaptado de [9]

Prueba. Se trata de probar si el software obtenido se ajusta a lo que se quería obtener. Si no es así se vuelve a las fases anteriores.

Instalación. Al culminar esta fase el sistema de software ya es totalmente funcional, por lo tanto debe ser instalado para que el usuario pueda utilizarlo cotidianamente como apoyo a sus actividades.

Mantenimiento. Se refiere a cualquier modificación, ajuste o adaptación del sistema ya instalado en el ambiente real, bien sea por errores que no se hayan detectado antes, por cambios en el entorno, o que se requiera modificar el modo en que el sistema ejecuta ciertas operaciones.

En el contexto actual, el modelo del prototipo, con el que se pretende probar las teorías propuestas, arranca con el establecimiento de los requerimientos del sistema, se definen los objetivos y los requisitos conocidos según las áreas de mayor prioridad e importancia para el sistema. Luego se hace un diseño preliminar, sobre el cual se construye un prototipo o modelo del sistema, compuesto de ventanas, tablas de la base de datos, formatos de entrada y de salida básicos. Posteriormente, se realizan las pruebas en base a un ejemplo.

Este prototipo se ajusta lo mejor posible a la solución requerida. Es un prototipo utilizable, automatizado funcional y horizontal (ver sección 5.1.1). No obstante, hay que

tener en cuenta que el prototipo elaborado no es el sistema final, puesto que normalmente apenas es utilizable. Puede ser demasiado lento, grande, general (sin considerar casos particulares), contendrá errores (debido al diseño rápido) o estará codificado en un lenguaje o para una máquina inadecuada. En la siguientes secciones se describen las fases del prototipo desarrollado.

5.2. Análisis - Requerimientos

El análisis del sistema de software que se desea desarrollar, ya ha sido descrito ampliamente en el capítulo anterior. Como se mencionó en el mismo, el Agente Integrador (AI) de aplicaciones heterogéneas propuesto es deliberativo, pues es un sistema basado en el conocimiento. La aproximación tradicional para diseñar agentes con características reactivas y deliberativas, consiste en verlos como sistemas basados en conocimiento o sistemas expertos [54].

5.2.1. El Agente Integrador modelado como un Sistema Experto

El enfoque de agentes vistos como sistemas expertos es el que se utiliza en esta investigación con el objetivo de obtener una primera aproximación del AI. Antes de presentar una correspondencia entre ambos, se hace necesario conocer lo que son sistemas expertos o sistemas basados en conocimiento.

Definición de Sistema Experto

Conocidos bajo las denominaciones de Sistemas Basado en Conocimiento (*Knowledge Based Systems*), o Sistemas Expertos (*Expert Systems*), vienen a ser programas computacionales que resuelven problemas que normalmente requieren del conocimiento de un especialista o experto humano [44]. Es un sistema capaz de tomar decisiones inteligentes interpretando grandes cantidades de datos sobre un dominio específico de problemas.

Arquitectura de un Sistema Experto

En la figura 5.2 se muestran los componentes de un Sistema Experto (SE), el cual está conformado por [44]:

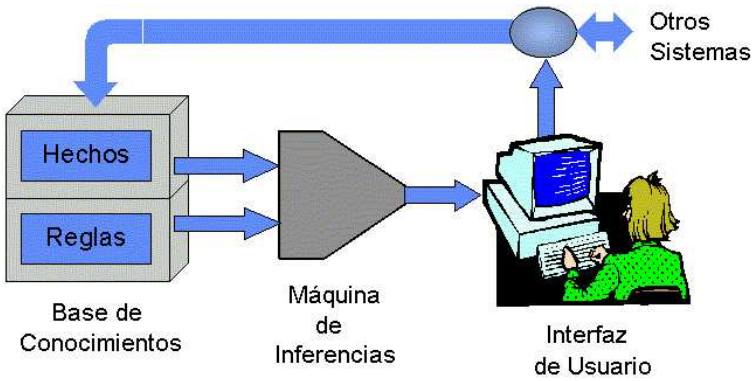


Figura 5.2: Arquitectura de un Sistema Experto o Sistema Basado en Conocimiento

Base de Conocimiento. Comprende el conocimiento del experto humano convenientemente formalizado y estructurado. Está constituido por la descripción de los objetos y las relaciones entre ellos, así como de casos particulares y excepciones. Algunos SEs incluyen metaconocimiento o conocimiento sobre el conocimiento, es decir, la capacidad para buscar en la base de conocimiento y abordar la resolución del problema de una manera inteligente usando diferentes estrategias para la resolución con sus condiciones particulares de aplicación. Es decir, se trata de definir criterios mediante los cuales el sistema decide la estrategia de búsqueda a utilizar en función de unos datos iniciales. El conocimiento se puede representar mediante cálculo de predicados, listas, objetos, redes semánticas y/o *reglas de producción*.

Máquina de Inferencia. También llamado *intérprete de reglas*, es un módulo que se encarga de las operaciones de búsqueda y selección de las reglas a utilizar en el proceso de razonamiento. Por ejemplo, al tratar de probar una hipótesis dada, el motor de inferencia irá disparando reglas que irán deduciendo nuevos hechos hasta la aprobación o rechazo de la hipótesis objetivo.

Base de Hechos. Se trata de una memoria temporal auxiliar que almacena los datos del usuario, datos iniciales del problema, y los resultados intermedios obtenidos a lo largo del proceso de resolución. A través de ella se puede saber no sólo el estado actual del sistema, sino también cómo se llegó a él.

Interfaz de Usuario. Todo sistema dispone de una interfaz de usuario, que gobierna el diálogo entre el sistema y el usuario. Para el desarrollo de estas interfaces, algunas herramientas de desarrollo incorporan generadores de interfaz de usuario

o bien se utilizan herramientas de desarrollo de interfaces gráficas existentes en el mercado.

Agentes y Sistemas Expertos

Los SEs pueden, desde cierta perspectiva, considerarse como agentes basados en conocimiento. Un agente basado en conocimiento, al igual que un SE, está compuesto por una base de conocimiento y un mecanismo de inferencia. El agente opera almacenando sentencias sobre el mundo en su base de conocimientos, usando su mecanismo de inferencia para inferir nuevas sentencias y utilizando éstas para decidir que acción realizar. Dichas sentencias están expresadas en algún lenguaje de representación del conocimiento el cuál tiene asociado algunas reglas de inferencia. Estas reglas de inferencia son las que gobiernan el comportamiento del mecanismo de inferencia.

En un agente, a partir de un acto de percepción, se genera una cierta acción. La repetición de este proceso en el tiempo define el comportamiento del agente. En un agente basado en conocimiento, y también en un SE, las acciones son determinadas mediante el pareamiento de la situación del mundo (como es percibida por el agente) con el antecedente de una regla situación - acción.

5.2.2. Requerimientos de Software

Dadas las características del sistema que se desea implementar, a continuación se describen los paquetes de software necesarios para tal fin.

Lenguaje de Programación Orientado a Objetos: Java

Java es un Lenguaje de Programación Orientado a Objetos ideado por Sun Microsystem [8]. Es un lenguaje de propósito general por lo que se podría crear cualquier tipo de aplicación con él, pero su mayor éxito se produce en Internet, con los famosos Applets o las aplicaciones Cliente/Servidor.

Las características principales que nos ofrece Java respecto de cualquier otro lenguaje de programación, son:

- *Es simple*: Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos.

- *Es orientado a objetos*: Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo.

- *Es distribuido*: Java se ha construido con extensas capacidades de interconexión TCP/IP. Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando.

- *Es robusto*: Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución.

- *Es de arquitectura neutral*: el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará.

- *Es multihilo (multithreaded)*: Java permite muchas actividades simultáneas en un programa. El beneficio de ser multihilos (*multithreaded*) consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real.

Sistema Manejador de Bases de Datos Relacionales: MySQL

MySQL es un servidor de bases de datos SQL (*Structured Query Language*). En la actualidad, SQL es el lenguaje más usado mundialmente para la definición e implementación de bases de datos.

Las principales características de MySQL son su velocidad y robustez. A pesar de seguir en desarrollo MySQL ofrece una alta funcionalidad.

Sistema Experto: *Java Expert System Shell* (JESS)

La capacidad de razonamiento de los agentes de la plataforma se ha conseguido gracias a la integración en la misma de un paquete denominado JESS (*Java Expert System Shell*) [26].

JESS es una máquina de inferencia y un entorno de desarrollo encriptado, escrito completamente en el lenguaje Java. JESS se puede definir como un Entorno de Desarrollo de Sistemas Expertos en Java. A través de él, se pueden construir *applets* y

aplicaciones Java que tengan la capacidad de "razonar", utilizando conocimiento suministrado por medio de reglas declarativas.

Las principales funcionalidades que ofrece dicho entorno son:

- *Un motor de inferencia*: encargado de decidir qué regla de comportamiento del agente se dispara en cada momento. Puede considerarse que simula la capacidad humana de alcanzar una conclusión razonando.

- *Lenguaje declarativo basado en reglas*: permite la definición de las reglas de comportamiento y hechos que constituyen los estados mentales de los agentes inteligentes.

El hecho de que JESS esté escrito íntegramente en Java posibilita que el comportamiento de los agentes sea programado usando Jess, Java o una combinación de ambos. La integración de JESS en la plataforma consiste básicamente en la declaración de un objeto motor de inferencia, que dispone de todo el control sobre el agente.

El motor de inferencia se ejecuta en una hebra o hilo, a la cual se la traspasa toda la información recibida del entorno. Dependiendo de la información que reciba, el estado mental del agente variará, cambiando a su vez las reglas que el motor de inferencias disparará.

El usuario puede además interactuar en tiempo real con el agente a través desde su interfaz modificando su estado mental.

Utilizando JESS se crea una base de conocimiento -donde se definen las reglas- y una base de hechos -con información sobre el entorno- que, posteriormente, son manipulados por el programa del agente, basado en JESS y Java, utilizando las mismas para tomar decisiones apoyado en su máquina de inferencia.

5.2.3. Requerimientos de Hardware

El prototipo está montado sobre una sola máquina. Sin embargo, dada las características distribuidas de Java, éste puede ser extendido a otras máquinas a través de la implementación de clases que soporten tal distribución.

Para la ejecución del prototipo se requiere una computadora con las siguientes especificaciones mínimas:

- Microprocesador Pentium 500 MHz
- 128 Mb de memoria RAM
- Disco Duro de 2.0 Gb
- Soporte de video SVGA
- Teclado y ratón

5.3. Diseño

El modelo base de implementación del prototipo es el mostrado en la figura 4.4. Para modelar dicho prototipo se extrajeron las clases principales como son ***Agente Integrador***, ***Aplicación***, ***InterfazCORBA***, ***ObjetoDeNegocio***, y ***Regla*** (junto con las clases necesarias para formar una RN: ***Condición***, y ***Condición Compuesta***).

Para poder implantar el modelo de clases, es necesario contar con una estructura de almacenamiento que permite recuperar el estado de los objetos en cualquier instante del tiempo. Así, la persistencia de los objetos del modelo se logra a través de una BD. En el cuadro 5.1, se muestra el esquema conceptual de la BD, utilizado en la implementación del prototipo, en términos del modelo Entidad-Relación (referencias sobre los atributos de las clases en el Anexo C). Esta estructura hace posible mantener en el tiempo y recuperar la información referente a cada uno de los componentes que conforman el sistema integrado.

Separar el procesamiento de las RNs requiere de una base de reglas (análoga a una base de datos). Una base de reglas separa la lógica del negocio de una aplicación y ellas (las reglas) pueden ser compartidas por muchas aplicaciones. Esta separación permite que las reglas existan independientemente de las aplicaciones, lo que hace posible cambiar los parámetros de la organización sin tener que "bajar" los sistemas de información.

Por otra parte, el AI se implanta como una aplicación "inteligente" utilizando un paquete de sistemas expertos (JESS). La correspondencia entre el modelo del AI y un SE se muestra en la figura 5.3 (los elementos punteados no forman parte del SE). Está figura fue presentada en el capitulo anterior, y se replica parte de ella para mostrar cómo los componentes "inteligentes" del AI poseen sus equivalentes en un SE (cuadro 5.2).

Entidades	Relaciones
Personal(codUsu,usuario,passwd,nombre,telefono,tipoPer)	**Soporte_Tecnico**(codUsu,codTec)
Tecnologia(codTec,nombre,descripcion,fabricante,fechaCompra, fechaInstalacion)	**Administracion_Operacion**(codUsu, codApl,tipo)
Software(codTec,sistOper,version)	**Plataforma**(codTecS,codTecH)
Hardware(codTec,ram,dd,video)	**Plataforma_Tecnologica**(codTec,codApl)
Aplicacion(codApl,nombre,descripcion,tipo,categoria,status, version)	**Funcionalidad**(codFun,codApl)
InterfazCORBA(codFun,transaccion,numArg,tipoRes,status)	**Parametros**(codArg,codApl)
Argumento(codArg,nombArg,tipoArg,direcc)	**Si**(codCon,codFun)
Regla(codReg,nomRegla,fechaCreación,fechaUltMod)	**Condiciones**(codCon,codReg)
Actuacion(codReg)	**Entonces**(codFun,codReg)
Sensado(codReg)	**DeLoContrario**(codFun,codReg)
Observacion(codReg)	**SubObjeto**(codObj1,codObj2)
Evaluacion(codReg)	**Entrada_Salida**(codObj,codFun,tipoES)
Condicion(codCon,opComparacion,valor)	**Genera_Utiliza**(codObj,codFun,tipoGU)
CondicionCompuesta(codCon)	**Integracion**(codAplAI,codApl)
ObjetoDeNegocio(codObj,nombre,tipoObj,descripcion,longitud, info)	
Estado(codObj)	
Mision(codObj)	
AgenteIntegrador(codAplAI)	

Tabla 5.1: Esquema conceptual de la persistencia del modelo de implementación

Componente del AI	Equivalente en un SE
Modelo	Base de Hechos
Reglas	Base de Reglas
Interpretador	Máquina de Inferencia
Capacidades	Forman parte de las precondiciones de las reglas
Plan Instanciado	Consecuentes de las reglas

Tabla 5.2: Componentes "inteligentes" del AI y su equivalente en un SE

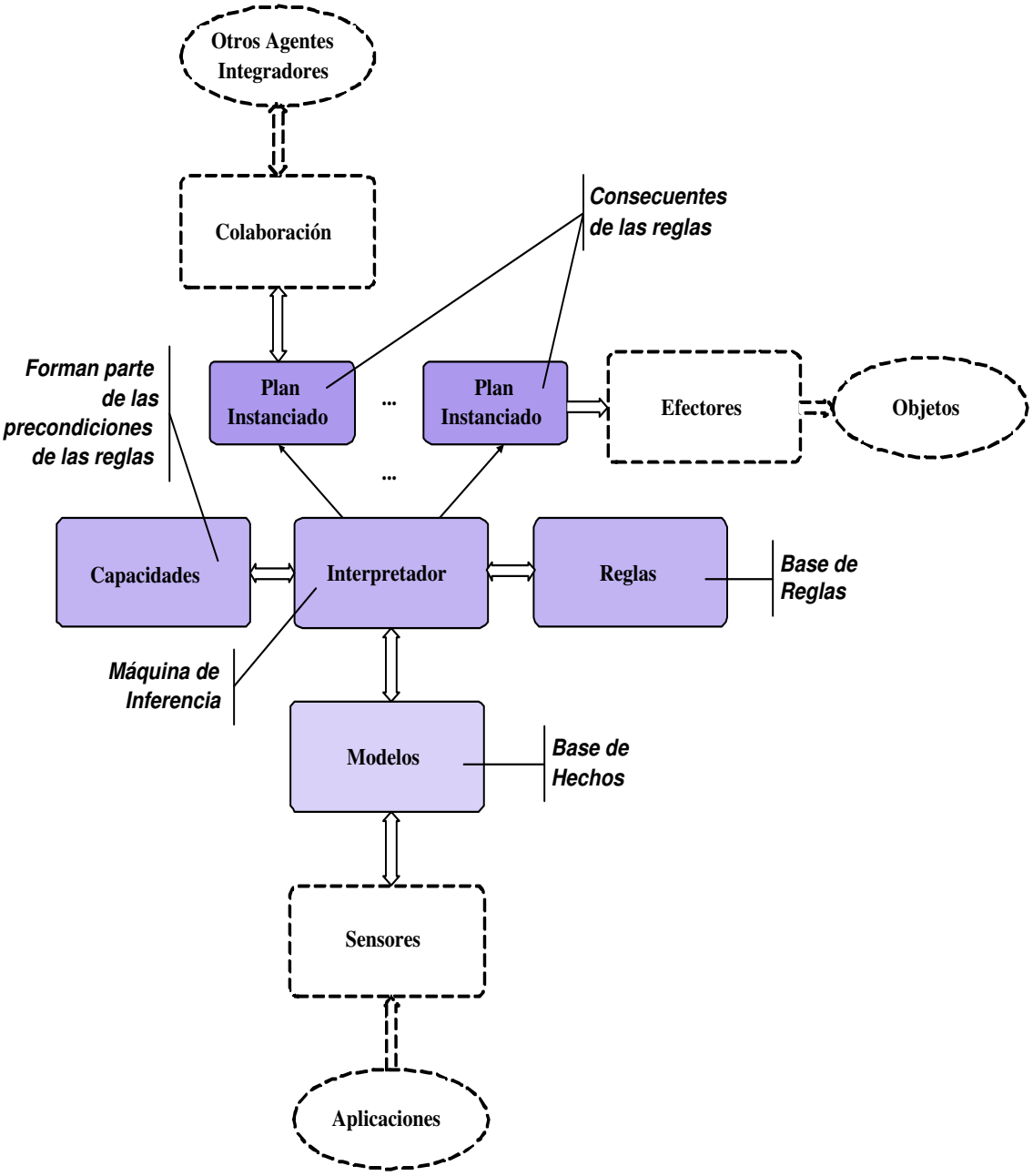


Figura 5.3: Correspondencia ente el AI y un SE

5.4. Implementación y Pruebas

El prototipo desarrollado consta de una interfaz compuesta de un conjunto de ventanas, a través de las cuales el usuario puede interactuar con el sistema integrado. La ventana principal del prototipo cuenta con tres opciones en su barra de menú (figura 5.4). Estas son:

- **Clases**: Permite la gestión de las diferentes clases del prototipo.
- **Agente**: Hace posible la interacción y visualización del comportamiento del AI, así como la manipulación de las RNs.
- **Salir**: Para cerrar la ventana principal del prototipo.

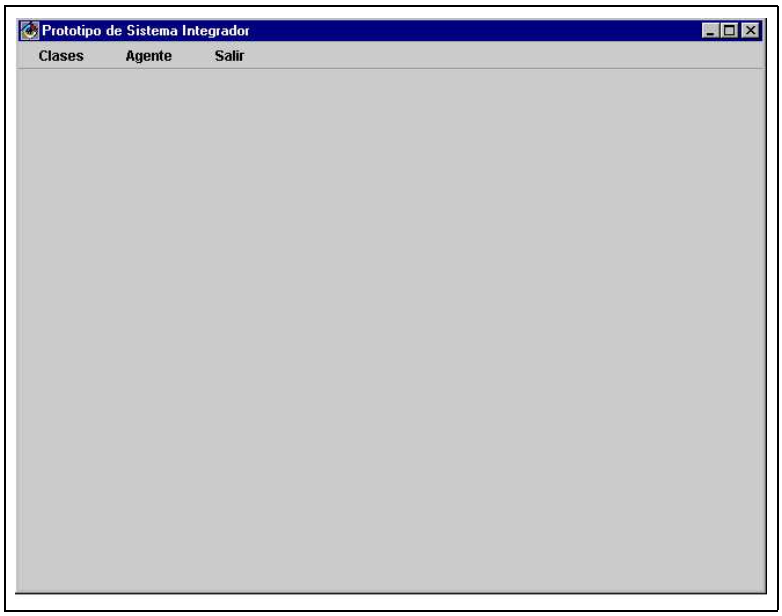


Figura 5.4: Ventana Principal del Prototipo del Sistema Integrador

A continuación se explican detalladamente cada una de estas opciones de menú.

Menú Clases

El menú **Clases** (figura 5.5) contiene un grupo de opciones que permiten al usuario manejar el conjunto de clases que forman parte del prototipo. A través de la opción **Atributos y Métodos** él podrá conocer los atributos y métodos de las clase presentes en el sistema (figura 5.6). Así mismo, éste podrá instanciar un objeto de una clase o conocer sobre algún objeto que es extensión de alguna clase.

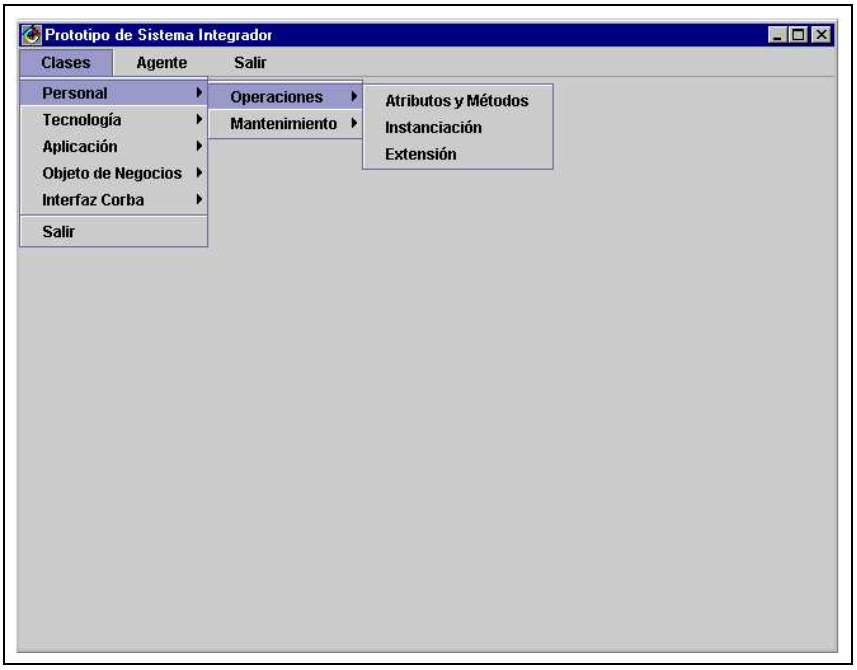


Figura 5.5: Menú **Clases** del Prototipo

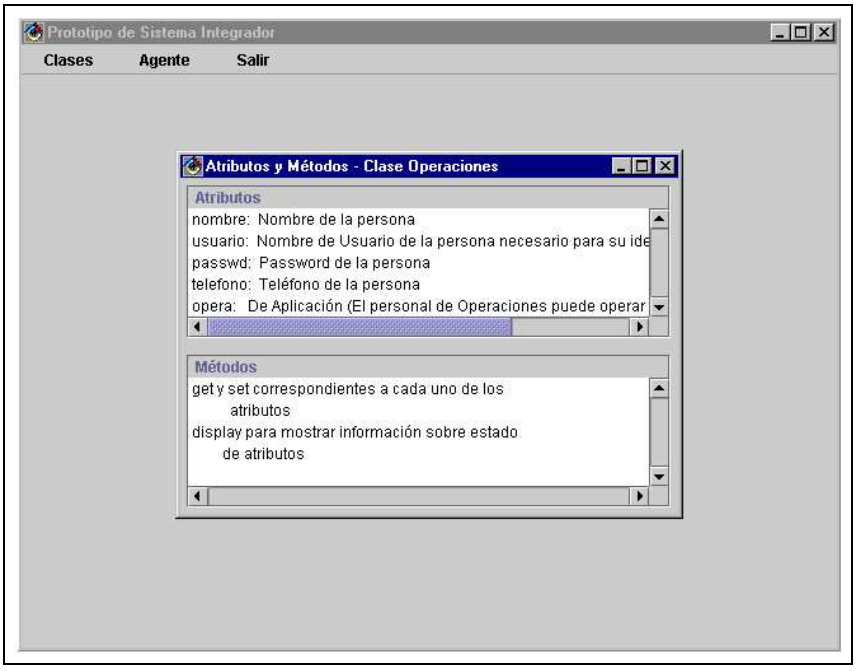


Figura 5.6: Menú **Clases** - Opción **Atributos y Métodos**

Específicamente, a través de la opción **Instanciación** del menú **Clases**, el usuario puede *crear nuevos objetos*, como se muestra en las figuras 5.7 y 5.8 que corresponden a la interfaces de instanciación de las clases **Operaciones** y **Aplicación**, respectivamente. En las figuras 5.9 y 5.10 se presentan algunos ejemplos de dichas interfaces.

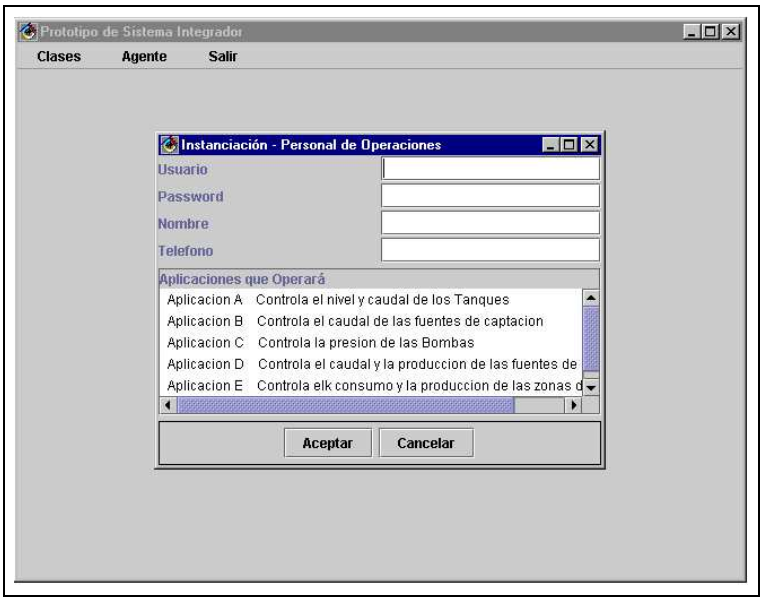


Figura 5.7: Menú **Clases** - Opción **Instanciación** (clase **Operaciones**)

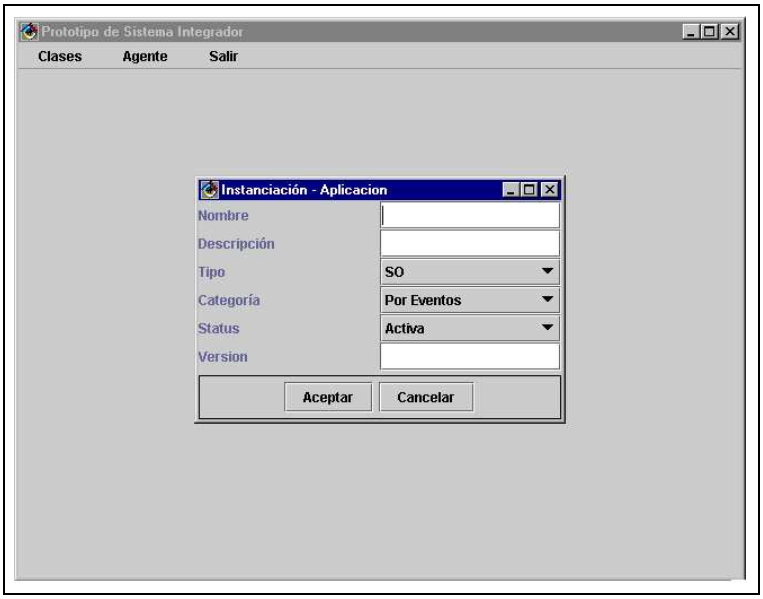


Figura 5.8: Menú **Clases** - Opción **Instanciación** (clase **Aplicacion**)

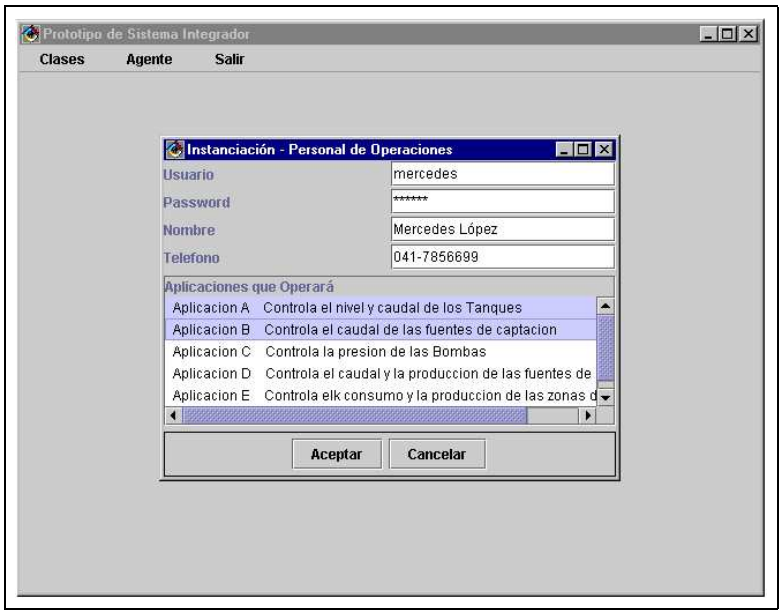


Figura 5.9: Menú **Clases** - Opción **Instanciación**. Ejemplo de la clase **Operaciones**

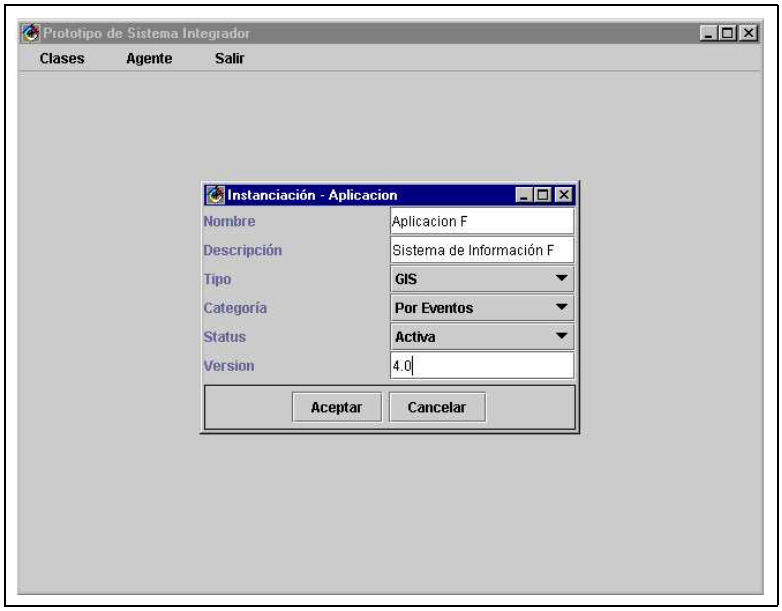


Figura 5.10: Menú **Clases** - Opción **Instanciación**. Ejemplo de la clase **Aplicacion**

Por otra parte, la opción **Extensión** permite obtener información sobre los objetos persistentes en el prototipo. Las figuras 5.11 y 5.12 muestran las interfaces de extensión para las clases **Operaciones** y **Aplicación**, respectivamente. En ellas se puede observar que existen botones que permiten al usuario *consultar, modificar o eliminar objetos persistentes del sistema*. En las figuras 5.13 y 5.14 se presentan algunos ejemplos de las mencionadas interfaces.

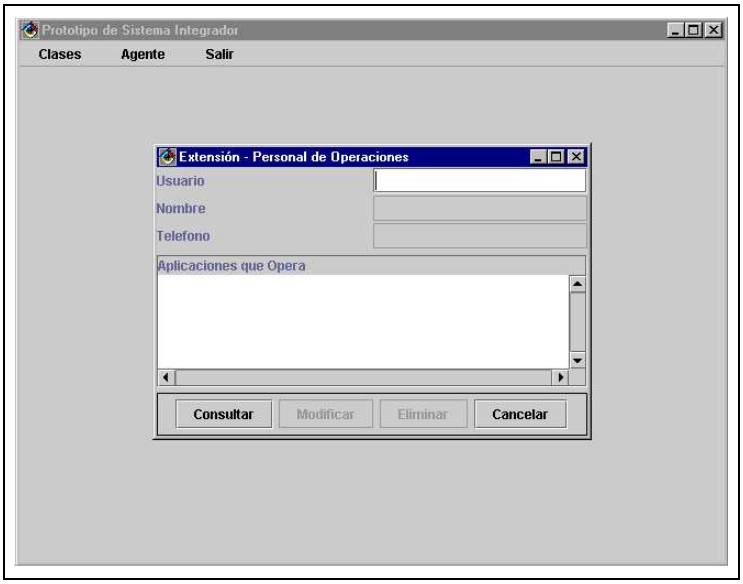


Figura 5.11: Menú **Clases** - Opción **Extensión** (clase **Operaciones**)

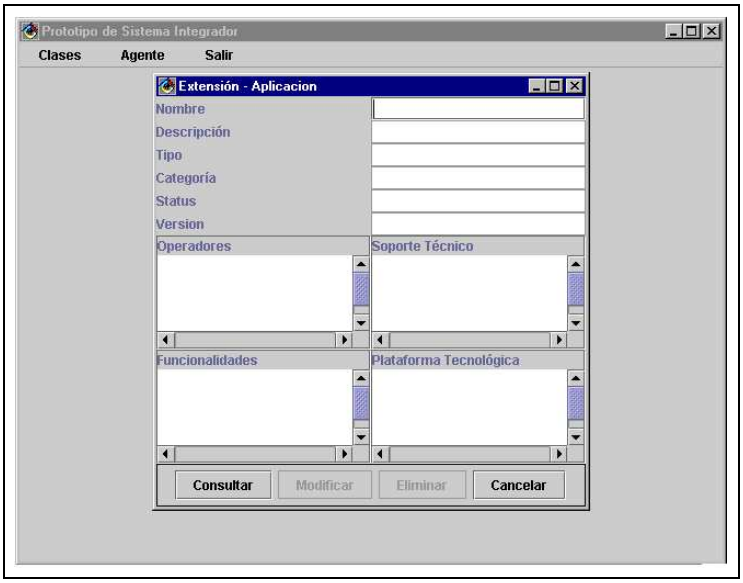


Figura 5.12: Menú **Clases** - Opción **Extensión** (clase **Aplicacion**)

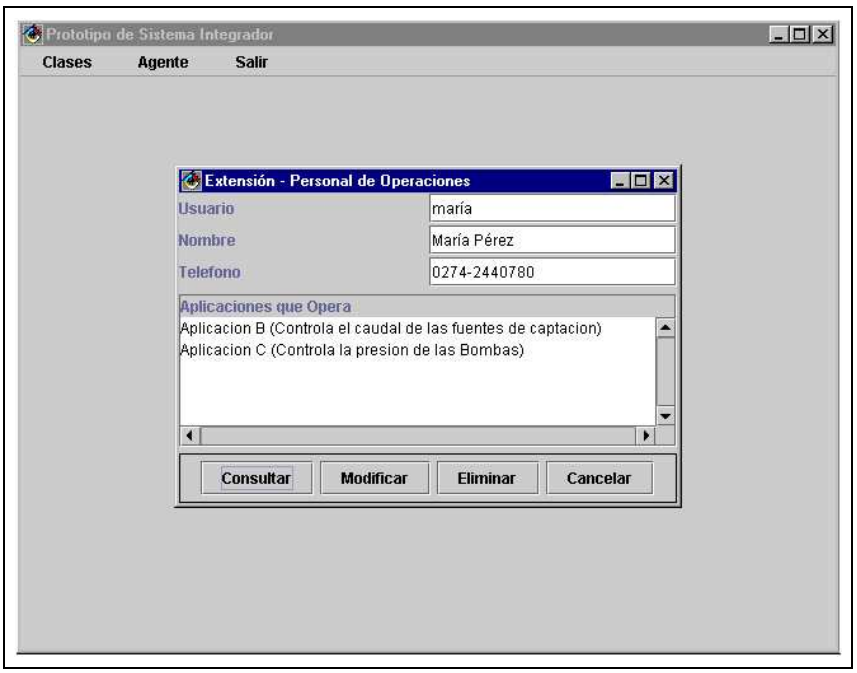


Figura 5.13: Menú **Clases** - Opción **Extensión**. Ejemplo de la clase **Operaciones**

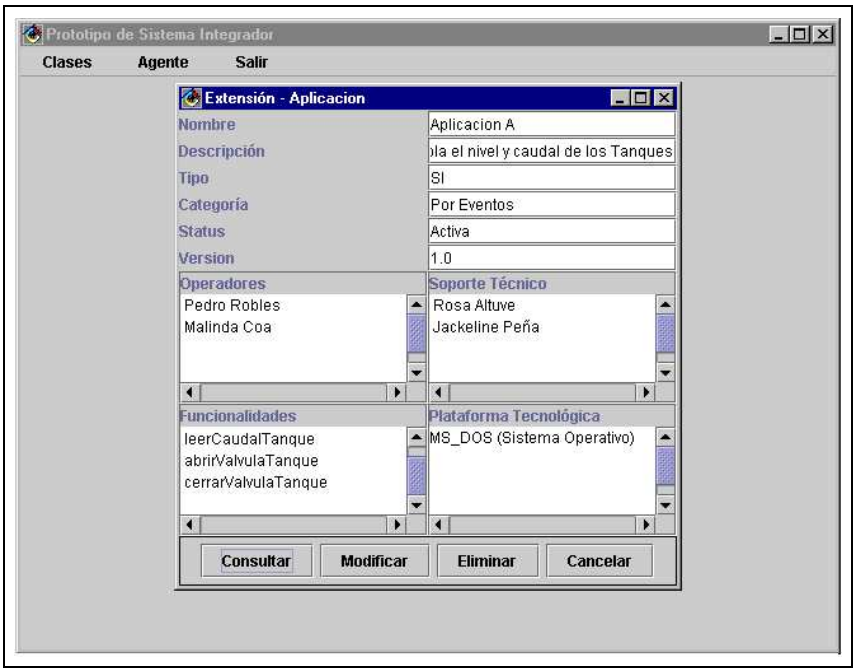


Figura 5.14: Menú **Clases** - Opción **Extensión**. Ejemplo de la clase **Aplicacion**

Menú Agente

La opción de menú más importante del prototipo es la de **Agente**, pues en ella se emplazan las características del sistema integrador. En la figura 5.15 se observa que ella posee opciones que permiten poner en **Ejecución** el AI, que representa a la UP cuyas aplicaciones se están integrando, además de *soportar la administración de las RNs (crear, modificar o eliminar las mismas)*

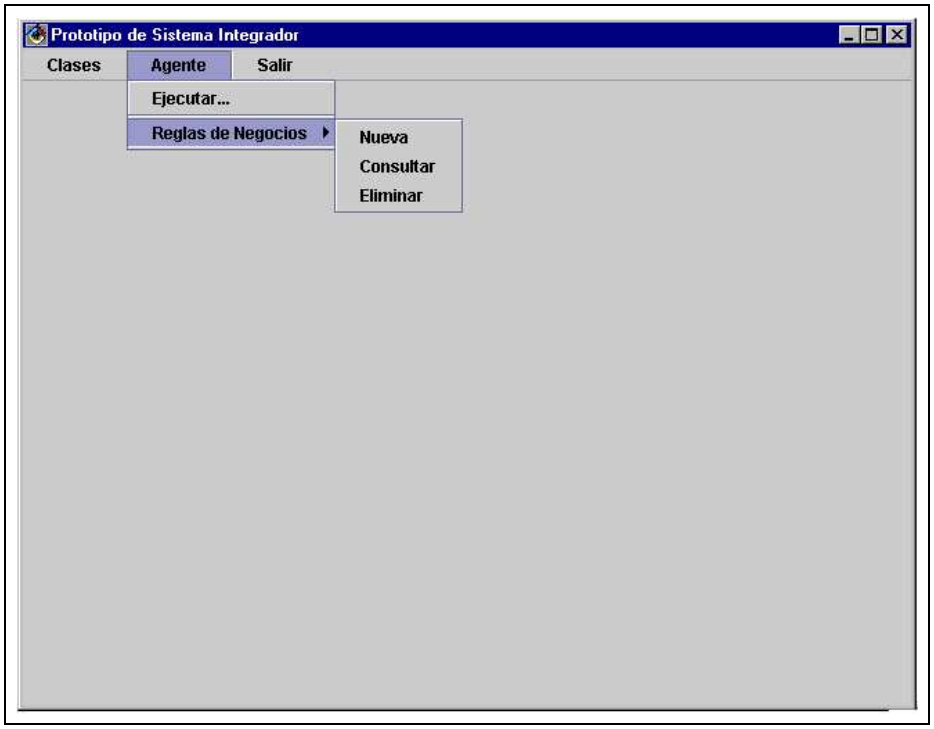


Figura 5.15: Menú **Agente** de Prototipo

En la figura 5.16 se muestra la ventana donde se puede observar la ejecución del AI. En ella aparecen columnas en donde se da a conocer toda la información relacionada con las acciones que ejecuta el AI, a cada instante de tiempo, en pro de integrar las aplicaciones de la UP que él representa. La información presentada en estas columnas representan lo siguiente:

- ***Regla***: Identificación o nombre de la RN que el AI está utilizando.
- ***Objeto de Negocio***: ON manejado por el AI en la RN.
- ***Estado Anterior***: Estado previo del ON al disparo de una regla
- ***Estado Actual***: Estado posterior del ON al disparo de una regla

- ***Acción Realizada***: Acción o acciones ejecutadas por el AI

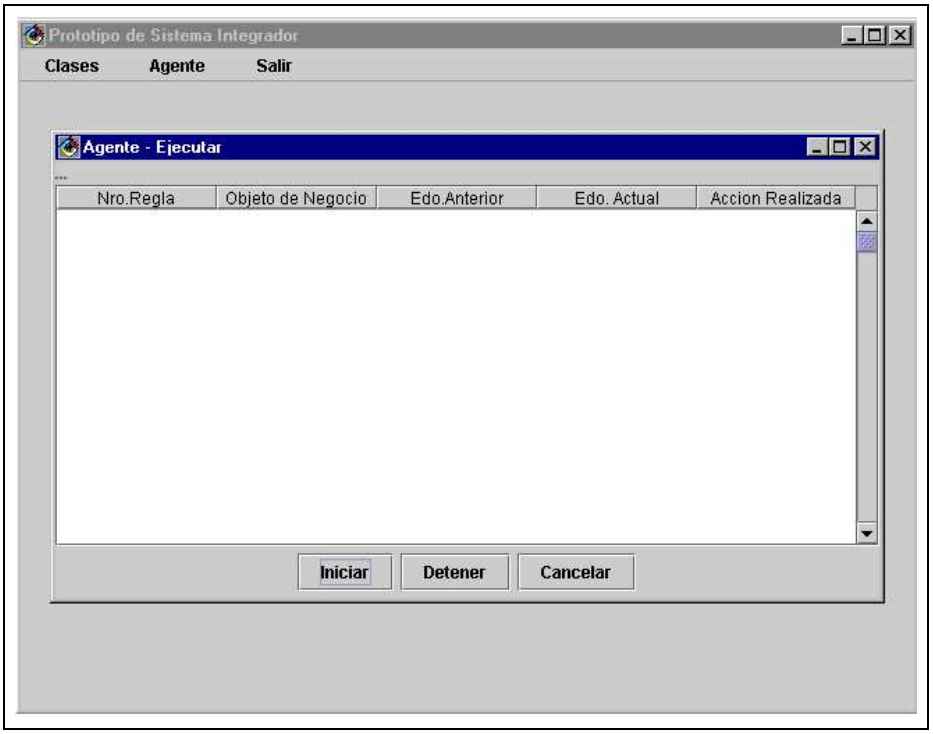


Figura 5.16: Menú **Agente** - Opción **Ejecutar**

Como ya se mencionó, el menú **Agente** también posee la opción **Reglas de Negocios** a través de la cual se pueden administrar las RNs. La figura 5.17 muestra la interfaz del prototipo por medio de la que se puede crear una RN, en la que se encuentran los componentes necesarios para la creación de una RN con estructura *SI-ENTONCES*. En la figura 5.18 se muestra un ejemplo de esto.

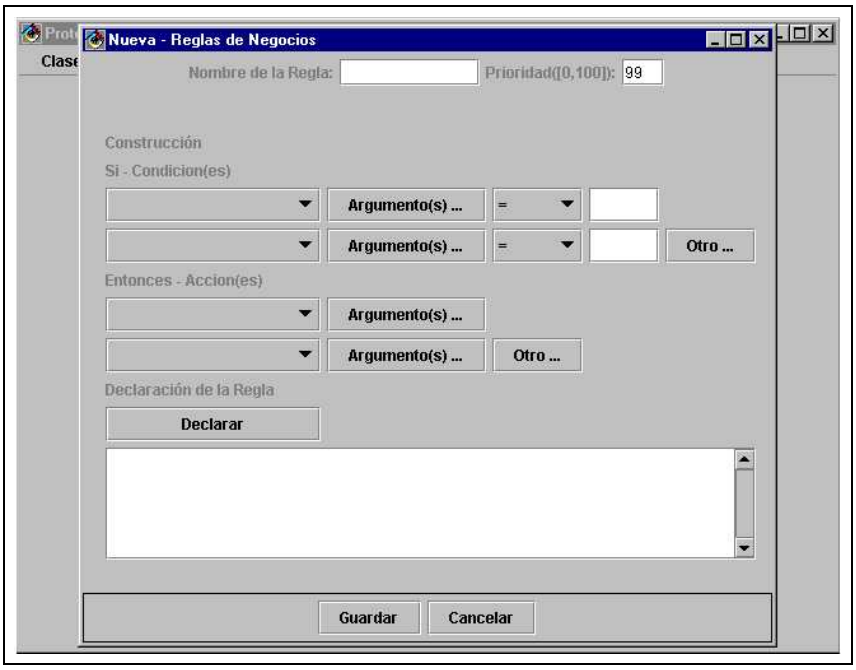


Figura 5.17: Menú **Agente** - Opción **Reglas de Negocios**. Crear una RN

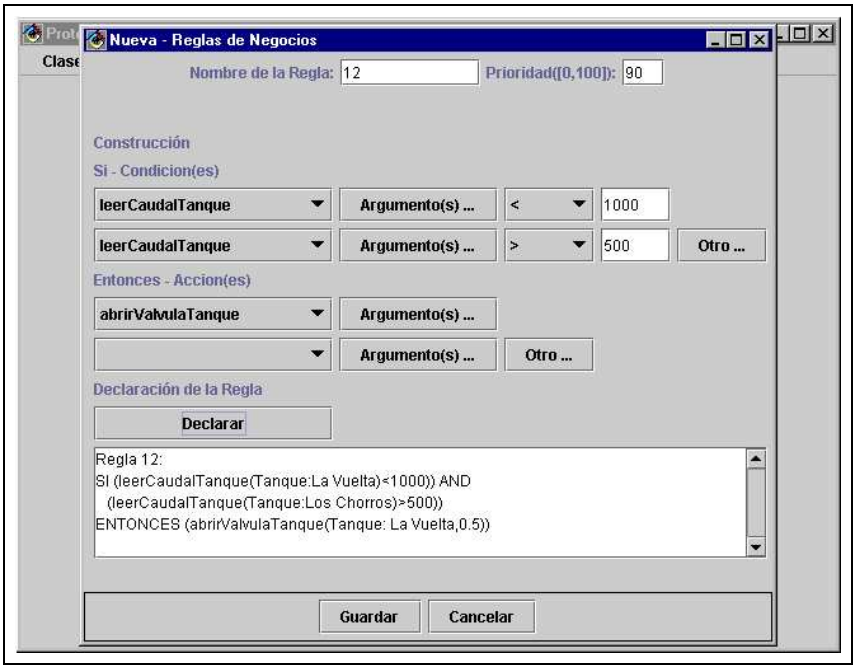


Figura 5.18: Menú **Agente** - Opción **Reglas de Negocios**. Ejemplo de la creación de una RN.

Ejemplo de Prueba

Con el fin de probar el prototipo implementado, se tomó como ejemplo de prueba una porción de la lógica del negocio de la Empresa Aguas de Mérida. Esta es una empresa de servicio cuya misión es proporcionar agua potable al Estado Mérida, con participación del Gobierno Regional y las Alcaldías del Estado, utilizando para ello las fuentes que proporciona la naturaleza, procesando el agua y devolviéndola al ambiente. Todas estas actividades las debe realizar de manera que sea una empresa autosostenible, manteniendo el costo de los servicios al público dentro de unos parámetros aceptables para el mismo. Para asegurar esta misión, la empresa tiene que optimizar sus procesos reduciendo los costos operativos: materiales, energía, y haciendo uso óptimo del recurso humano disponible en la empresa [36].

En el Anexo D, se presenta el ejemplo con el que probó el prototipo. Las figuras 5.19 y 5.20 capturan distintos instantes de ejecución del AI. En las ventanas de ejecución[1] se muestra las reglas[2] utilizadas por el AI en diferentes instantes de tiempo, el ON involucrado en el proceso de negocio que se está llevando a cabo, su estado anterior y actual (antes y después de la ejecución de la RN, respectivamente), y la acción ejercida por el AI como sistema integrador. En dichas figuras se observa que el AI utiliza las RNs denominadas regla_1, regla_3, regla_4 y regla_5[3] en la integración de aplicaciones, que involucran el control del nivel de los tanques llamados "La_Vuelta" y "La_Hechicera", quienes se representan a través de ONs. Las acciones ejercidas por el AI implican el cierre o la abertura de válvulas de entrada de agua en los tanques en determinadas proporciones, de acuerdo a las especificaciones de las RNs respectivas.

En JESS, el motor de inferencia trabaja con encadenamiento hacia adelante y además permite el uso de diferentes estrategias de resolución de conflictos. Cada regla tiene una propiedad llamada *salience*, que define su prioridad. Las reglas activadas de mayor prioridad, se disparan primero seguidas de aquellas de menor prioridad. A través de esta prioridad, se puede forzar el disparo de reglas. Si se tienen varias reglas con la misma prioridad, ellas son disparadas de acuerdo a la estrategia de resolución de

[1] Descrita en la sección anterior

[2] Recuérdese, por el capítulo 4, que tanto los antecedentes como los consecuentes de las RNs, están conformados por funcionalidades de las aplicaciones que informan sobre los procesos internos y las capacidades de las mismas

[3] Únicas reglas almacenadas para el momento

conflictos activa. JESS posee dos estrategias: *en profundidad* (estrategia por defecto), donde las reglas recientemente activadas se disparan antes que otras con la misma prioridad, y *en amplitud*, en donde las reglas se disparan en el orden en que son activadas. A pesar de esto, en el prototipo no se le asignan prioridades a las reglas pues se forzaría el disparo de las mismas en un orden particular, lo que tendría un impacto negativo en la ejecución del sistema.

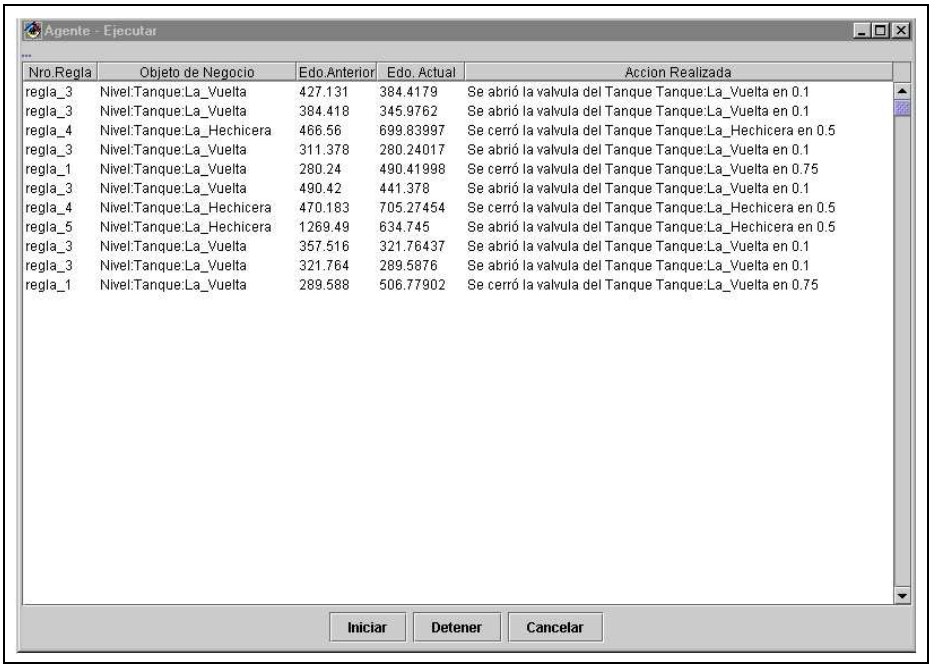


Figura 5.19: Ejecución del AI

Por otra parte, en la figura 5.21 se observa que el AI interactuando con otro AI, particularmente enviándole información sobre algo que está ocurriendo dentro él y que es de interés para el otro AI. Según especificaciones del modelo propuesto, un AI tiene conocimiento sobre la información que él genera que puede ser de interés para otros AIs. Esta es una acción denominada *comunicar* (ver figura 4.10), en la que el AI le envía información a un agente externo. Específicamente, en la figura 5.21 el AI le está enviando información a un agente llamado "Otro Agente", referida al momento de la abertura de válvulas de los tanques.

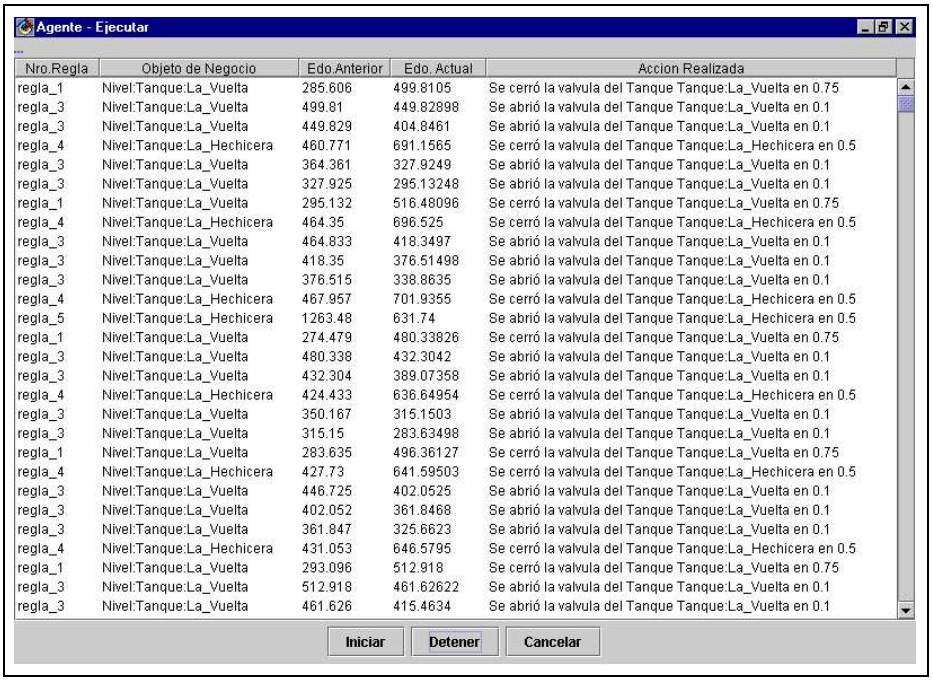


Figura 5.20: Ejecución del AI

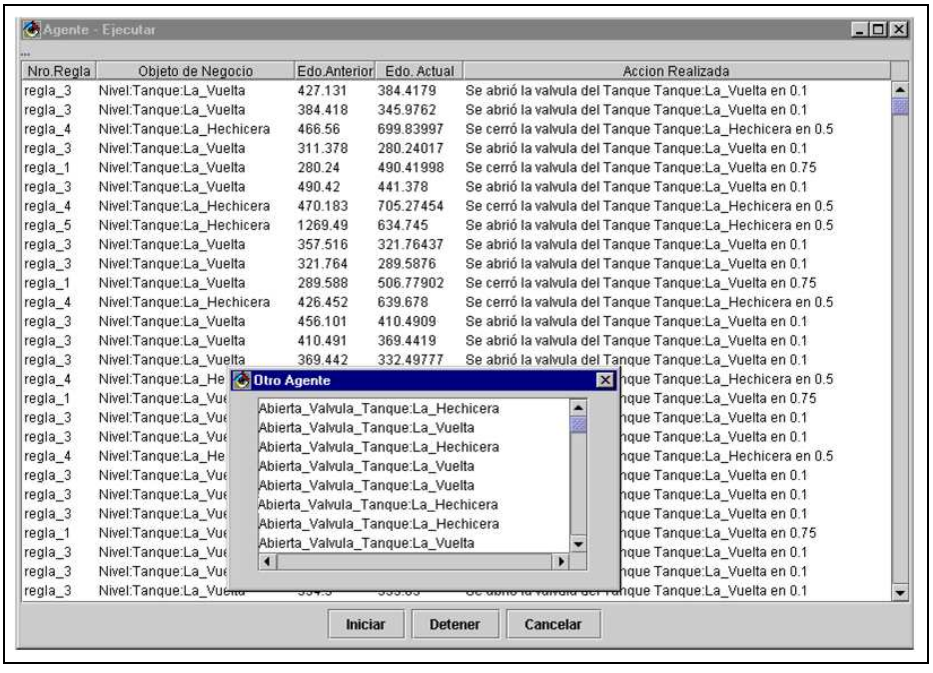


Figura 5.21: Ejecución del AI

5.5. Conclusiones

El desarrollo de un prototipo para el Agente Integrador que permita que las aplicaciones heterogéneas presentes en una UP compartan información, ha demostrado que es posible llegar a la construcción de tales sistemas integradores como parte del proceso de Automatización e Integración de Procesos de Producción Continua. Esto conlleva a una mejora en la velocidad y eficiencia en los procesos de toma de decisiones dentro de un complejo industrial.

El prototipo desarrollado es de naturaleza genérica, por lo tanto puede ser aplicado en cualquier UP que se encuentre en cualquier nivel de la jerarquía empresarial. Para ello, es necesario conocer a fondo los procesos involucrados, y hacer las modificaciones pertinentes sobre el prototipo; pero en sí, la lógica de diseño y la implantación del mismo no sería alterada mayormente.

Capítulo 6

Conclusiones y Recomendaciones

Un sistema de producción automatizado es aquel donde los procesos de producción son principalmente coordinados y controlados mediante sistemas automáticos. Los modelos de *sistemas de producción* en las industrias son los que describen el comportamiento de un proceso de producción. Entre estos procesos, los más comunes son los de *manufactura*, donde la salida de un producto está asociado con la finalización de una tarea, y los de *producción continua* que trabajan de manera permanente y sus productos resultantes alimentan otros procesos.

Cada uno de estos procesos tiene asociados modelos de automatización. Para los procesos de manufactura, el modelo asociado es el *CIM* (*Computer Integrated Manufacturing*) y para los procesos de producción continua, es la *Pirámide de Automatización*. Ambos modelos, propuestos por la ISO, se caracterizan por ser jerárquicos, estar conformados por un número determinado de niveles y manejar cierto tipo de información en cada uno de ellos.

Otro modelo relacionado a la automatización de procesos continuos es el Enfoque Jerárquico, que también exhibe las características de los anteriores y en el que un *Complejo de Producción Continua* (CPC) es visto como un conjunto de *Unidades de Producción* (UP) que comparten recursos y controlan internamente su comportamiento para lograr sus metas. La noción de UP es recursiva en toda la jerarquía y tanto el CPC, como la empresa pueden ser consideradas UPs.

En la automatización de procesos se hace necesario integrar los distintos niveles jerárquicos, pero hasta ahora las metodologías de integración planteadas están orientadas hacia los procesos de manufactura. Es por ello que se ha propuesto un *Modelo de Referencia de Automatización Integral* (MRAI), que describe de una manera genérica

un método para elaboración de planes estratégicos con miras a la integración de los procesos de negocios, procesos de datos y conocimientos, aplicaciones de software y sistemas de información de una empresa de producción continua. MRAI está basada en la Pirámide de Automatización y extiende la misma para considerar caras llamadas *arquitecturas*, a saber: de Gestión, Tecnologías de Producción, Objetos, Aplicaciones y Tecnologías de Información y Comunicación.

Particularmente, la *Arquitectura de Aplicaciones* de MRAI especifica todas y cada una de las aplicaciones que integran el negocio. Estas aplicaciones, son necesarias para apoyar tanto los procesos físicos, como de toma de decisiones dado que ellas manejan el conocimiento sobre la disponibilidad y capacidad de las UPs, así como la información sobre los indicadores económicos, de ventas, de contabilidad de costos, etc.

Normalmente, las aplicaciones que conforman la Arquitectura de Aplicaciones de una empresa son *heterogéneas*. Ellas provienen de distintos proveedores, poseen distintas plataformas de hardware y software y no tiene capacidad de comunicarse entre sí o integrarse a otros productos lo que, consecuentemente, particiona al ambiente industrial en "*islas de información*". Aún siendo así, estas aplicaciones forman parte de la empresa y deben integrarse en el proceso de automatización.

Dentro de este marco de ideas, cabe señalar entonces la necesidad de una herramienta que permita que estas aplicaciones, aún siendo heterogéneas, "conversen" y puedan compartir información de manera que el proceso de toma de decisiones dentro de la empresa se pueda realizar la manera más efectiva posible, y así solucionar el problema del aislamiento de la información.

Desde una perspectiva más general, actualmente existen técnicas que permiten la integración de aplicaciones heterogéneas. Estas técnicas pueden enfocarse desde dos perspectivas: las que integran aplicaciones existentes dentro de la empresa y aquellas que cumplen la función de integración, pero entre dos o más empresas. Dichos dominios de integración son denominados *Integración Intra-Empresa* e *Integración Entre-Empresas*, respectivamente. En ambos dominios la integración puede ser de varios tipos: a *nivel de datos* -la integración toma lugar entre repositorios de datos-, a *nivel de interfaz de aplicaciones* -integración a través de los servicios proporcionados por las aplicaciones-, a *nivel de métodos* -las aplicaciones se integran a través de un conjunto de métodos

comunes- y a *nivel de interfaces de usuarios* -conocido como *screen scrapping*, la integración es a través de pantallas de usuarios.

Ahora bien, haciendo un enfoque en el dominio de la Integración Intra-Empresa se encuentra que existen diversas arquitecturas que soportan la integración de aplicaciones. Entre ellas se ubica la *integración punto-a-punto*, donde todas las aplicaciones se conectan todas entre sí, pero presenta el inconveniente de tener muchas conexiones cuando el número de aplicaciones aumenta. Como solución a este problema, surge la tecnología *Broker Central de Mensajes* al cual se conectan todas las aplicaciones; si una de ellas cambia sólo una conexión se debe modificar: la del *Broker* de Mensajes. Una extensión de la tecnología anterior es el *Broker* de Procesos que, además, encapsula la lógica del proceso pudiendo ser ésta manipulada a través de una interfaz de usuario.

Por otra parte, se cuenta con varias estrategias para la integración de aplicaciones como son la *transferencia* de componentes de un dominio a otro, la *conexión* de interfaces que sirven como puente entre los sistemas a integrar, la *extensión* de los sistemas de software existentes para incorporarles características de otros y, por último, la *combinación* de características de dos o más sistemas existentes para crear uno nuevo.

Basados en estas arquitecturas y estrategias, se han desarrollado una variedad de soluciones para la integración de aplicaciones que permiten la interoperación o comunicación de la información subyacente en el negocio. Si bien es cierto que estas alternativas ayudan en el proceso de integración, no ofrecen un resolución completa al problema implícito en el mismo.

Entre las soluciones propuestas para la integración de aplicaciones, destacan las basadas en Sistemas *Middleware*. Un *Middleware* es un módulo intermedio que actúa como conductor entre sistemas de software, permitiendo a cualquier usuario comunicarse con varias fuentes de información que se encuentren conectadas en una red. Por medio de él, se pueden conectar entre sí una variedad de productos procedentes de diferentes proveedores.

Existen diferentes categorías de sistemas *Middleware*, pero todas orientadas a un denominador común: las aplicaciones distribuidas. Unas cumplen sus objetivos de integración a través del intercambio de mensajes, por llamadas remotas a procedimientos, acceso a bases de datos estándar, etc.

Particularmente, la categoría de *Middleware para tecnologías orientadas a objetos*

u ORB (*Objects Request Broker*) habilita a los objetos, que conforman una aplicación, estar distribuidos y compartidos, a través de redes heterogéneas. ORB maneja la comunicación y el intercambio de fragmentos de objetos, de distintos vendedores, permitiéndoles esconder a sus clientes detalles de su implementación como lenguaje de programación, sistema operativo, hardware o localización.

Uno de los exponentes de la tecnología ORB es CORBA (*Common Object Request Broker Architecture*). CORBA constituye un conjunto de especificaciones destinadas a facilitar la ya referida interoperación de objetos distribuidos. Uno de los puntos de apoyo de CORBA, para alcanzar sus objetivos, es el lenguaje de definición de interfaces IDL (*Interface Definition Language*) que, como su nombre lo indica, permite definir la interfaz de los objetos a través de la especificación de los métodos que él puede ejecutar o funcionalidades, sus parámetros de entrada, resultados y excepciones de ejecución. La implementación de las interfaces IDL se hace por medio de un lenguaje de programación que facilite dicha implementación.

Como otra posible alternativa a la solución de la integración de ambientes heterogéneos, surgen los *Agentes de Software* que se caracterizan por ser sistemas flexibles y robustos, capaces de decidir por sí mismos qué deben hacer para alcanzar sus objetivos de diseño.

Los agentes han nacido y crecido en el contexto de la comunidad de Inteligencia Artificial y aunque todavía no existe un consenso de lo que es un agente, se han hecho importantes aproximaciones a la definición, diseño y construcción de diversos tipos de agentes. De manera general, un *agente* se puede definir como un sistema que, ubicado en un entorno, percibe su ambiente mediante sensores y responde o actúa de manera autónoma en tal ambiente por medio de efectores.

Algunos investigadores, le han conferido a los agentes un conjunto de características como son la autonomía, sociablidad, reactividad y proactividad. Sin embargo, para otros estas características no son suficientes para describir a los agentes, por lo que ellos le han otorgado otras como movilidad, veracidad, benevolencia, racionalidad y adaptación.

Se han establecido varias arquitecturas concretas para agentes, dependiendo de la manera cómo ellos toman las decisiones: *Basados en Lógica* -a través de deducciones lógicas-, *Reactivos* -transforman directamente situaciones en acciones-, *Creencia/Deseo/Intención* -dependen de la manipulación de estructuras de datos que representan

las creencias, los deseos y las intenciones del agente- y en *Capas* -donde la toma de decisiones se realiza a través de varias capas software-.

Actualmente, los agentes son aplicados en un amplio rango de dominios como aplicaciones industriales, comerciales, médicas, entretenimiento, entre otros. Dentro del dominio de aplicaciones comerciales los agentes se han extendido en el *manejo de procesos de negocios*. A través de un proceso de negocios, una empresa coordina las unidades semi-autónomas que la constituyen, especificando las tareas que deben ejecutarse, así como las decisiones que deben tomarse en la generación de un producto o servicio.

Los Objetos y Reglas de Negocios son dos aspectos importantes que estructuran los procesos de negocios. Un *Objeto de Negocio* (ON) es utilizado para modelar elementos del negocio. Él representa la estructura y el comportamiento de un concepto o algo del mundo real, que es significativo a un dominio particular de negocios como, por ejemplo, un producto, orden de producción, capacidad de producción, etc. Por su parte, las *Reglas de Negocios* (RN) son declaraciones que definen o restringen algún aspecto del negocio. Ellas controlan o regulan los procesos de negocios.

Emplazadas en el contexto del manejo de procesos de negocios y la integración de aplicaciones heterogéneas, se han propuesto arquitecturas basadas en agentes. Una de ellas es la arquitectura ADEPT (*Advanced Decision Envirotment for Process Task*), que constituye una plataforma multiagentes para el manejo de negocios, donde la descripción de un proceso de negocios se transforma en agencias (representada por un agente), quienes se hayan conectadas a un medio de comunicación común. Para la integración de aplicaciones, el último enfoque basado en agentes encontrado en la literatura, propone el alcance de tal integración a través de una infraestructura que puede ser vista desde dos caras: una de la empresa y otra del usuario; cada una de ellas representadas por sus agentes de interfaz y de usuario, respectivamente, quienes ofrecen servicios y se hayan interconectados en red para lograr la integración requerida.

Ubicados en el escenario de un complejo de producción continua (CPC), modelando los mismos como una composición de elementos (UPs), que a su vez son vistos como partes compuestas de componentes (UPs), en la presente investigación, se ha expuesto una arquitectura que trata de lograr la integración de las aplicaciones heterogéneas presentes en cada uno de dichos componentes, todos vistos como unidades de producción genéricas.

Para tal fin, se tomó como base un modelo de clases de objetos que soportan tanto la integración horizontal como vertical de un CPC. Dicho modelo tiene como clase principal una llamada **Aplicación**, que soporta las aplicaciones de software de la UP y que está relacionada con las clases **Objetos de Negocio** y **Reglas de Negocio**, necesarias en este caso para realizar la integración de aplicaciones utilizando CORBA.

Dentro del modelo, como una especialización de la clase **Aplicación**, se desprende la clase **Agente Integrador**. Esta es una aplicación especial que conforma el mecanismo responsable de la sincronización, mantenimiento y control de todas las aplicaciones integradas; constituye el artificio de integración, razón por la cual en torno a ella se erige la arquitectura propuesta.

Como solución general al problema de integración de las aplicaciones de las UPs, se propone que cada una de ellas se encuentre representada por un *Agente Integrador* (AI), estructurado como un agente de software, que cumple con una función interna, integrando las aplicaciones que están dentro de la UP que él representa, y otra externa, permitiendo el intercambio de información entre UPs o con usuarios externos a él. Esta solución es recursiva y puede ser aplicada a todos los componentes de un CPC.

En el entorno presentado, el AI es una entidad autónoma en el sentido de que funciona independientemente de otros agentes, con capacidades de conocimiento y comunicación, y cuyo comportamiento está dirigido por unas reglas (las RNs) que restringen su autonomía interna. Los AIs, localizados en las UPs, exhiben características como reactividad (reacciona ante cambios en su entorno), orientado a un objetivo: integrar aplicaciones, semi-autonomía interna (están condicionados a reglas preestablecidas), capacidad de deliberación (ejecuta acciones en base al conocimiento de su entorno) y, por último, capacidad de cooperación con otros AIs.

Dadas las características deseadas, la arquitectura del AI corresponde a la de un agente híbrido, pues él refleja características reactivas ante los cambios de su entorno y deliberativas, razonando a través de una máquina de inferencia.

La *estructura general del AI* está constituida por niveles, con funciones específicas en cada una de ellos. El primer nivel cuenta con sensores, a través de los cuales el AI puede percibir cambios o eventos que ocurren en su entorno. En el segundo nivel, esta información actualiza las creencias del AI, representada por los modelos que él posee sobre sí mismo y sobre otros agentes. En base a las creencias, capacidades y reglas que

restringen su comportamiento, en su tercer nivel, el AI instancia un plan haciendo uso de un interpretador. Dicho plan contiene las acciones que el AI debe ejecutar. Estas acciones, que corren en hilos independientes, forman parte del cuarto nivel y pueden implicar acciones sobre objetos o la colaboración con otros agentes (quinto nivel).

Sustentado en esta arquitectura, el *AI funciona* de la siguiente manera: sensa los eventos provenientes de las aplicaciones dado que estas, a través de funcionalidades definidas en IDL (CORBA), pueden notificar sobre cambios ocurridos dentro de ellas. Estos eventos pueden o no actualizar los modelos del AI. Posteriormente, haciendo uso de su interpretador y basándose en sus creencias, reglas, estado interno actual y entorno, el AI procede a tomar decisiones de las que se desprenden un subconjunto de reglas a disparar, conformando este subconjunto un plan instanciado que a su vez también es controlado por el AI. Un plan instanciado, puede implicar acciones a realizar sobre un objeto, otro agente, el entorno o estado interno del AI.

La *plataforma de integración* de las aplicaciones de una UP se logra a través de CORBA. Para ellos, es necesario que las funcionalidades de cada una de las aplicaciones de cada UP sean registradas; ellas constituyen las interfaces de las aplicaciones y son definadas en IDL. Estas funcionalidades representan las capacidades que posee cada aplicación y son las que generan los eventos a fin de informarle al AI lo que está ocurriendo dentro de ellas. De esta forma, los datos manejados por una aplicación no son tratados por el AI, sino por la aplicación misma.

En el modelo, las reglas que rigen el comportamiento del AI, es decir las RNs, representan solamente fragmentos de la lógica del negocio y de acuerdo a ellas, se transfiere información de un punto a otro dentro del sistema integrado. Estas reglas se estructuran bajo el paradigma Evento-Condición-Acción-Acción (ECAA) y tanto sus antecedentes como sus consecuentes están conformados por funcionalidades de aplicaciones. Además de esto, es necesario acotar que las RNs son modeladas fuera de las aplicaciones constituyendo, de esta manera, una base de reglas independiente. Hacerlo así tiene ventajas, tales como el hecho que ellas puedan ser compartidas por muchas aplicaciones, una mejor adaptabilidad del negocio en su constante evolución, así como también una mayor flexibilidad, al no tener que alterarse las aplicaciones cuando se modifique la lógica del negocio.

En lo que respecta a los ONs, ellos son utilizados por las aplicaciones para la ejecución de la lógica del negocio. Sólo aquellos ONs útiles para las aplicaciones, serán almacenados en el sistema y se estructuran de manera que un ON este compuesto de otros subONs. Los ONs son creados y mantenidos por las RNs.

Basado en los aspectos anteriores, el *sistema integrado* puede ser visto como una población de AIs, representantes de los componentes UP, que interactúan entre sí con el fin de compartir información. Siendo así, el modelo de integración de una UP pasa a ser una estructura escalable, jerárquica y modular, que puede ser extendida a un CPC y sus componentes.

Con el fin de probar los supuestos planteados en esta investigación, se *desarrolló e implantó un prototipo* experimental, de naturaleza genérica, automatizado funcional y horizontal; para ello, se arrancó con el establecimiento de los requerimientos del sistema, se definieron los objetivos y los requisitos conocidos según las áreas de mayor importancia para el sistema. Luego se hizo un diseño preliminar, sobre el cual se construye el prototipo basado en una interfaz humano-computador compuesto de ventanas, tablas de la base de datos y formatos de entrada y de salida básicos. Posteriormente, se realizaron las pruebas en base a un ejemplo.

Como una primera aproximación, el *AI es modelado como un Sistema Experto* (SE), dada la compatibilidad de características entre ambos sistemas. Un SE está compuesto de una base de hechos, otra de reglas y una máquina de inferencia que, es este contexto, corresponden a los modelos, reglas e interpretador del AI, respectivamente. Además, las capacidades y planes instanciados del AI forman parte de las precondiciones y consecuentes de las reglas utilizadas por un SE.

Tomando en cuenta las características del sistema, el prototipo ha sido desarrollado en Java e implementando la "inteligencia" o capacidad de razonamiento del AI a través de un paquete de sistemas expertos denominado JESS (*Java Expert System Shell*), el cual es compatible con Java.

A través de la implantación y prueba del prototipo *se demuestra la posibilidad de construcción de estos sistemas integradores* como parte del proceso de Automatización de un Complejo de Producción Continua. Para futuras investigaciones, se recomienda incorporarle a estos Agentes Integradores capacidades como la de aprendizaje, de manera que ellos puedan aprender sobre cómo resolver situaciones a partir de experiencias o

situaciones a las que hayan estado expuestos en un pasado, y movilidad, de manera que posean facultades para "viajar" a través de *host* externos y buscar información fuera de la organización. De igual manera, se sugiere la aplicación de los conceptos establecidos para el AI, a un caso de la vida real.

Bibliografía

[1] Edi, intercambio electrónico de datos. 2000. http://www.monografias.com/EDI, Intercambio Electrónico de Datos.htm.

[2] Enterprise application integration (eai). 2000. http://www.peterindia.com/EAIOverview.html.

[3] Messaging oriented middleware. Technical report, Software Engineering Institute, Carnegie Mellon, 2000. http://www.sei.cmu.edu/str/descriptions/mom-body.htm.

[4] Middleware -software technology review. Technical report, Software Engineering Institute, Carnegie Mellon, 2000. http://www.sei.cmu.edu/str/descriptions/middleware-body.htm.

[5] Objects request broker. Technical report, Software Engineering Institute, Carnegie Mellon, 2000. http://www.sei.cmu.edu/str/descriptions/orb-body.htm.

[6] Remote procedure call. Technical report, Software Engineering Institute, Carnegie Mellon, 2000. http://www.sei.cmu.edu/str/descriptions/rpc-body.htm.

[7] Transaction processing monitors. Technical report, Software Engineering Institute, Carnegie Mellon, 2000. http://www.sei.cmu.edu/str/descriptions/tpm-body.htm.

[8] Tutorial de java. 2001. http://programacion.net/java/.

[9] J. Barrios. *Estudio de Estructuras, Componentes, Interrelaciones, Metodologías y Tecnologías asociadas a los Sistemas de Información.* Universidad de Los Andes, 2001.

[10] I. Besembel y E. Chacón. Objetos y reglas de negocios en la integración y automatización de procesos de producción continua. 2001. Universidad de Los Andes.

[11] E. Chacón. Redes de control de procesos: Un soporte para los sistemas de automatización integral. *Conferencia ISA-Occidente*, Octubre 1996.

[12] E. Chacón, I. Besembel, F.Ñarciso, J. Montilva, and E. Colina. An integration arquitecture for the automation of a continuous production complex. *ISA Transactions 42.Journal of the American Institute of Physics*, 2002.

[13] E. Chacón y G. Sarrasin de. Automatización integral de sistemas de producción continuos: Control y supervisión. *Universidad de Los Andes*, Julio 1998.

[14] F. Chacón. Integración de software heterogéneo a través de sistemas de información web(siw): Arquitectura y método de desarrollo. Master's thesis, Universidad de Los Andes, Junio 1999.

[15] CIM Reference Model Committee. *A reference model for computer integrated manufacturing (CIM): A description from the viewport of industrial automation.* Taller Internacional de Pardue sobre Sistemas de Computación Industrial, 1991.

[16] Attachmate Corporation. Tecnología de adaptación de pantallas. Octubre 2000. http://attachmate.com.mx/article/0,1012,34859,00.html.

[17] A. Dagotto. Portales de próxima generación. 2002. http://www.ebizlatam.com/.

[18] T. Davenport. Process innovation: Reengineering work through information technology. Technical report, Harvard Business School Press, 1992.

[19] Secretaría del Consejo Superior de Informática. Ministerio de Administraciones Públicas. Generadores de sistemas basados en conocimiento. 1998. Madrid. España.Disponible: http://www.map.es/csi/silice/Gensiscon1.html.

[20] Secretaría del Consejo Superior de Informática. Ministerio de Administraciones Públicas. Sistemas middleware. 1999. Madrid. España.Disponible: http://www.map.es/csi/silice/Global14.html.

[21] G. Doumeingts, B. Vallespir, M. Zanettin, and D. Chen. Cim grai integrated methodology, a methodology for designing cim systems. Technical report, Universidad de Bordeaux, Francia, Mayo 1992.

[22] M. Enric. Diseño de un agente que habita en internet. Master's thesis, Facultat d'Informàtica de Barcelona, Febrero 1997.

[23] B. Faltings. Intelligent agents: Software technology for the new millennium. 2002. http://www-lsi.upc.es/ ia/agentes/a001Faltings.pdf.

[24] OMG Business Object Domain Task Force. Business object concepts. OMG Document: bom/99-01-01. Disponible: http://www.omg.org/.

[25] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. 1996. http://www.msci.members.edu/franklin.

[26] E. Friedman-Hill and SandiaÑational Laboratories. Jess - the java expert system shell. 2000.

[27] CONICIT Proyecto G-97000824. Integración de software heterogéneo. 1999. [Página Web del Proyecto].Disponible: http://www.centauro.ing.ula.ve/itsh/.

[28] E. Gottesdiener. Business rules: Show power, promise. *Application Development Trends*, 4(3), 1997.

[29] M. Hammer and J. Champy. Reengineering the corporation. Technical report, Herper Collins, New York, 1993.

[30] L. Hidalgo. Inteligencia artificial y sistemas expertos. Technical report, Facultad de Ciencias Económicas y Empresariales (ETEA) - Universidad de Córdoba, 1998.

[31] N. Jennings, P. Faratin, and T.Ñorman. Implementing a bussiness process management system using adept: A real-world case study. *Int. Journal of Applied Artificial Intelligence*, 14(5):421–465, 2000.

[32] N. Jennings and M. Wooldrige. Software agents. *IEEE Review*, January 1996.

[33] N. Jennings and M. Wooldrige. Applications of intelligent agents. *Agent Technology: Foundations, Applications, and Markets (eds. N.R. Jennings and M. Wooldridge)*, 3(28), 1998.

[34] P. Johannesson. Application and process integration - concepts, issues and research directions. Technical report, Departament of Computer Science - Stockholm University, 2000.

[35] K. Kosanke, F. Vernadat, and M. Zelm. Cimosa: Enterprise engineering and integratión. *Computer in Industry*, 40, 1999.

[36] LASDAI. Estudio preliminar de factibilidad para el desarrollo de un sistema integral de automatización e información para la empresa aguas de mérida. facultad de ingeniería de la universidad de los andes. 1999.

[37] D. Linthicum. Eai application integration exposed. SoftwareMag.com, Feb-Mar 2000. http://www.softwaremag.com/archive/2000fec/EAI.html.

[38] Z. Maamar, D.Kettani, and N. Sahli. Software agents for enterprise application integration. *OPPLA'2000*, 2000. http:/www.jeffsutherland.org/oppsla200/zakaria/zakaria.htm.

[39] P. Maes. Artificial life meets entertaiment: life like autonomous agent. *Comunications of the ACM 38*, 11, 1995.

[40] A. Molina. Arquitectura de objetos integrados para la automatización industrial (modelo aoiai). Master's thesis, Universidad de Los Andes, Noviembre 1999.

[41] J. Montilva. An integration method applied to the design of a data/knowledge model for multimedia and spatial applications. Technical report, University of Leeds-School of Computer Studies, January 1993.

[42] J. Montilva. An object oriented approach to business modeling in information systems development. *Proceeding of the III World Multiconference on Systemics, Cybernetics and Informatics -SCI97*, 2:358–364, Agosto 1999.

[43] J. Montilva, E. Chacón, y E. Colina. Metas: Un método para la automatización integrada en sistemas de producción continua. *Información Tecnológica-12*, 6:147–156, 2001.

[44] A. Pacheco. Sistemas expertos. 2000. http://www.socrates.itch.edu.mx/ apacheco/ ai/definic.htm.

[45] PERA. Pera reference model for cim. *ISA Publication*. http://www.pera.net.

[46] J. Pimentel. *Communications Networks for Manufacturing*. Prentice Hall-Englewood Clifs, 1990.

[47] H. Roesler and L. Pierson. Open communications based on international standards. *Control Engeneering*, 5(39), 1992.

[48] S. Rusell and Norvig P. *Artificial Intelligence: A Modern Approach*. Prentice-Hall Inc, 1995.

[49] R. Shelton. Business objects: Modeling with business patterns. *Data Management Review*, 1996.

[50] Rational Software. Uml. 2000. http://www.rational.com/uml.

[51] P. Solivares. Desarrollo cliente/servidor: Ubicación de las reglas de negocio. *Revista Profesional para Programadores (RPP)*, 1997.

[52] K. Taveter. Agent-oriented business rules: Deontic assignments. *VTT Information Technology*, Finland. 2001.

[53] UTFSM. Ingeniería del software. introducción. 2001. http://grulla.hispalinux.es/enunciados/introduccion.pdf.

[54] G. Weiss. Multiagents systems: A modern approach to distributed artificial inteligence. Technical report, Massachusetts Institute of Technology, 2000.

[55] E. Yourdon, K. Whitehead, J. Thomann, K. Oppel, and P.Ñevermann. Mainstream objects: An analysis and design approach for business. Yourdon Press, 1996.

Parte I

Apéndices

Apéndice A

La Orientación a Objetos

La orientación a objetos (OO) es un enfoque para el desarrollo de sistemas programados, que permite la representación del dominio de una aplicación en forma natural y directa, en término de los objetos que intervienen en dicha aplicación.

Características de la OO

Las características fundamentales de la OO son: abstracción, encapsulación, herencia y polimorfismo.

Abstracción. Por medio de la abstracción conseguimos no detenernos en los detalles concretos de las cosas, sino generalizar y centrarse en los aspectos que permitan tener una visión global del tema.

Encapsulación. Esta característica permite ver un objeto como una caja negra en la que se ha introducido de alguna manera toda la información relacionada con dicho objeto. Esto permite manipular los objetos como unidades básicas, permaneciendo oculta su estructura interna.

Herencia. La herencia es el mecanismo para compartir automáticamente atributos y métodos entre clases y subclases de objetos.

Polimorfismo. Es la propiedad que indica, literalmente, la posibilidad de que una entidad tome muchas formas. En términos prácticos, el polimorfismo permite referirse a objetos de clases diferentes mediante el mismo elemento de programa y realizar la misma operación de diferentes formas, según sea el objeto al que se hace referencia en ese momento.

Objetos

Los conceptos de la OO se basan en la manipulación de representaciones abstractas de los objetos plasmadas en su definición y su comportamiento. Un **objeto** se define entonces como la representación de algo que se describe mediante una *estructura* y un *comportamiento*.

La **estructura** de un objeto describe aquellas características de interés presentes en el objeto y que sirven para plasmar el estado de ese objeto, siendo el **estado** de un objeto el conjunto de valores actuales almacenados en su estructura.

El **comportamiento** del objeto está representado por una serie de operaciones, funciones o métodos que modifican o no el estado del objeto, haciendo que ocurra un cambio de estado en el mismo, el cual representa el comportamiento del objeto en la realidad. Así, el comportamiento del objeto está dado por sus cambios de estado.

Mensajes y Métodos

Los objetos realizan acciones cuando ellos reciben mensajes. El **mensaje** es esencialmente una orden que se envía a un objeto para indicarle que realice alguna acción. Los objetos se comunican entre sí enviando mensajes. Los mensajes tienen una contrapartida denominada **métodos**, que son los procedimientos que se invocan cuando un objeto recibe un mensaje.

Clases

Una **clase** es la descripción de un conjunto de objetos; consta de atributos y métodos que resumen las características comunes de dicho conjunto. Se pueden definir muchos objetos de la misma clase. Las clases son similares a modelos o plantillas que describen cómo se construyen ciertos tipos de objetos. Cada vez que se crea un objeto a partir de una clase estamos creando lo que se llama *instancia* de esa clase. Por consiguiente, los objetos no son mas que instancias de una clase. Una instancia es una variable de tipo objeto. En la figura A.1 se muestra la representación gráfica de una clase y un ejemplo.

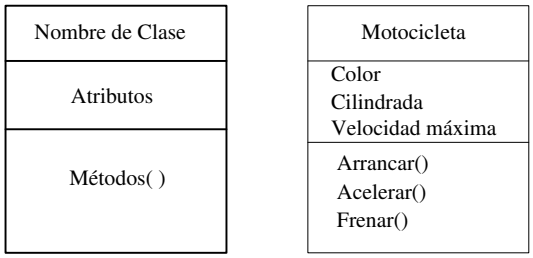


Figura A.1: Representación gráfica de las clases y un ejemplo.

Relaciones entre clases

Las relaciones entre clases juegan un papel importante en el modelo de objetos. Las clases, al igual que los objetos, no existen de modo aislado. Por esta razón existirán relaciones entre clases y por ende, entre objetos. Dichas relaciones pueden indicar alguna forma de compartición, así como algún tipo de conexión semántica. A continuación se definen los tipos de relaciones que pueden existir entre clases.

Relaciones de generalización/especialización (es-un)

Las clases se pueden organizar en estructuras jerárquicas. La *herencia* es una relación entre clases donde una clase comparte la estructura o comportamiento, definida en una (*herencia simple*) o más clases (*herencia múltiple).*

Se denomina *superclase* a la clase de la cual heredan otras clases. De modo similar, una clase que hereda de una o más clases se denomina *subclase*. Una superclase representa una *generalización* de las subclases. Una subclase de la clase dada representa una *especialización* de la clase ascendente. La clase derivada o subclase **es-un** tipo de la clase base o superclase. En la figura A.2 se muestra un ejemplo de estos conceptos.

Relaciones de asociación

Una **asociación** representa la relación estructural entre objetos de clases diferentes. La mayoría de las asociaciones son relaciones binarias aunque pueden existir relaciones ternarias o *n-arias*. En esencia, una asociación representa qué objetos de dos clases tienen un enlace (relación) entre ellos, significando, por ejemplo, que «unas clases conocen de otras clases», «están conectadas a», «por cada objeto x hay un objeto y», etc.

La **multiplicidad** representa el numero de objetos (*rango*) que indica cuántos objetos se pueden enlazar. El rango puede ser cero-a-uno (0..1), cero-a-muchos (0..* o

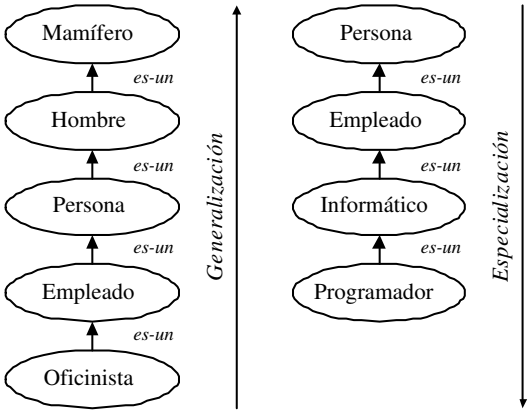


Figura A.2: Relación de jerarquía *es-un*.

sólo *)[1], uno-a-muchos (1..*), cuatro (4), seis a doce (6..12), o incluso varios números (2,5,7,8,9-15). Si no se especifica la multiplicidad, se considera uno (1) por omisión. La multiplicidad se muestra en los extremos de la asociación, en las clases donde es aplicable. Además de la multiplicidad, enl enlace entre clases muestra en cada extremo un **rol** (*papel*). Cada rol puede tener un nombre que indica cómo es visualizada la clase por otra.

Una asociación puede ser otra clase en sí misma, una **clase asociación**. La clase asociación no se conecta a ninguno de los extremos de la asociación, sino que se conecta a la propia asociación, y es utilizada para añadir información extra en un enlace. La asociación es similar a una clase normal; puede tener atributos, operaciones y otras asociaciones.

En la figura A.3 se presenta un ejemplo de estos conceptos. En dicho ejemplo se hayan asociadas dos clases: *Empresa* y *Persona*; a través del enlace navegable «trabaja-para» y «emplea a» (indicado por el sentido de la flecha) . En el ejemplo: una *Empresa* visualiza a *Persona* como un *empleado*; similarmente *Persona* visualiza a *Empresa* como *empleador*. Cada rol indica multiplicidad de su clase. Una *Persona* puede trabajar para muchas (*) *Empresas* y una *Empresa* puede emplear a muchas (1..*) *Personas*. La clase asociación *Trabajo*, muestra detalles de la relación entre *Empresa* y *Persona*.

Relación de agregración (tiene-un)

La **agregación** es un caso especial de asociación. El agregado indica que la relación entre clases es un tipo de *todo-parte*. Un ejemplo de un agregado es un avión que consta

[1]el símbolo * indica «muchos»

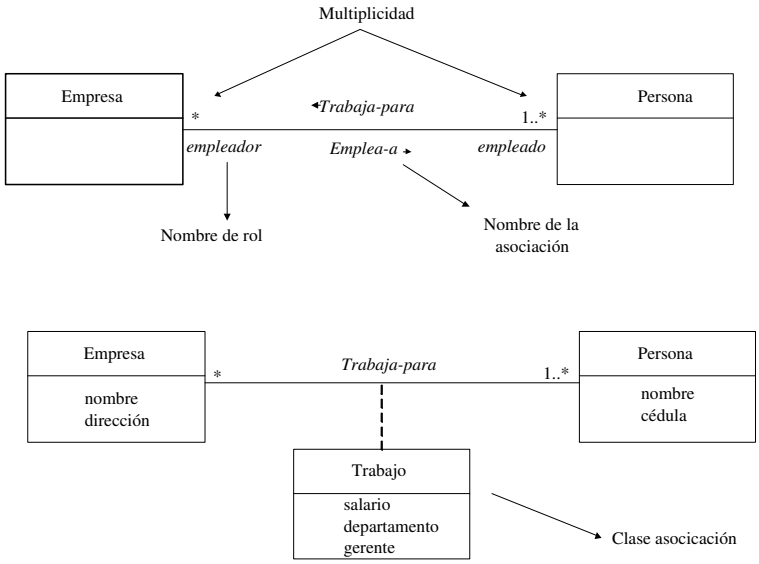


Figura A.3: Asociaciones con nombres, multiplicidad, roles y clases asociación.

de motores, alas, asientos, etc. Las palabras o frases que se utilizan para identificar agregados son *tiene-un*, *es-parte-de*, *consta-de*, *contiene*, etc., y señalan una relación *todo-parte* entre las clases implicadas.

La figura A.4 muestran ejemplos de agregaciones. La parte (a) del ejemplo, es una agregación normal: una *Universidad* tiene varias *Facultades* o *Escuelas*, de modo que se puede suprimir o añadir alguna Facultad y seguirá existiendo la Universidad. Esta es una característica de la agregación: las *partes* (*Facultades*) componen el *todo* (*Universidad*). La multiplicidad del *todo* es uno (1).

Existen diferentes tipos de agregaciones: *agregado compartido* y *agregado composición*. Una *agregación compartida* es un caso especial de la agregación normal, en el que las partes pueden ser partes en cualquier todo (figura A.4-(b)) Así, un equipo se compone de diferentes miembros (personas). Una persona puede ser miembro de muchos equipos, es decir, las personas son las partes compartidas. La agregación es compartida si la multiplicidad en el todo es distinta de uno (1). Una *agregación de composición* (o sólo *composición*) es aquella que *posee* («contiene») a sus partes y entraña una fuerte dependencia de propiedad. Las partes viven en el todo, de modo que se destruyen cuando se destruye el todo (figura A.4-(bc)). La multiplicidad en el lado del *todo* debe ser cero o uno (0..1), pero la multiplicidad en el lado *parte* puede ser un intervalo.

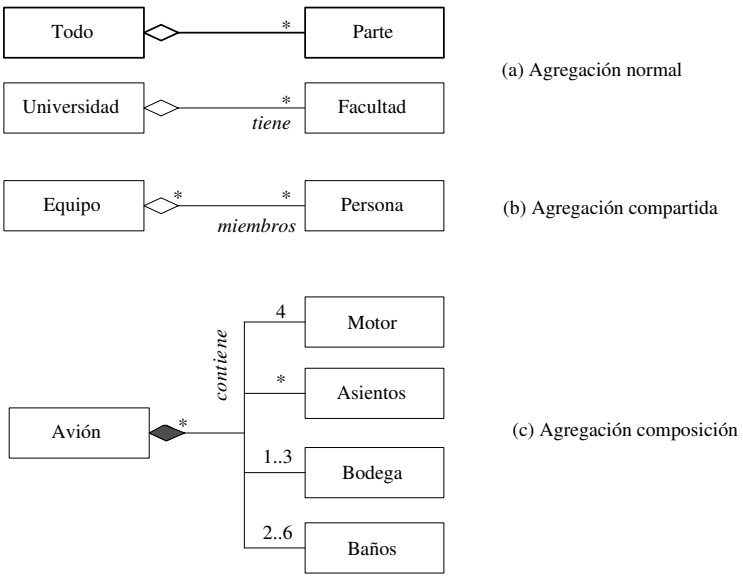


Figura A.4: Agregaciones.

Relación de uso

Una relación de **uso** implica una asociación entre clases. A veces una clase debe *hacer uso* de otra clase, pero no necesita incorporar esa clase en su propia estructura. Tales asociaciones se llaman relaciones de «uso». Normalmente una relación de «uso» significa que la **clase utilizadora** envía mensajes a la **clase utilizada**, pero la clase utilizada (usada) no se almacena en una de las variables de la instancia de la clase utilizadora. En su lugar, la clase utilizada se pasa a la clase utilizadora por alguna tercera parte, como un argumento de uno de los métodos de la clase utilizadora. La notación gráfica de la relación de uso se muestra en la figura A.5.

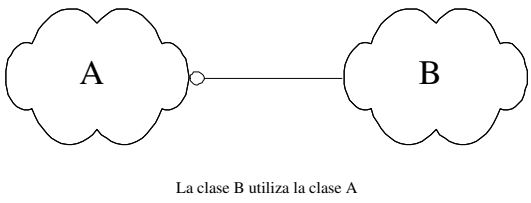
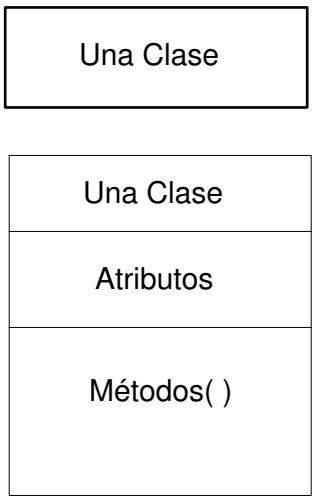


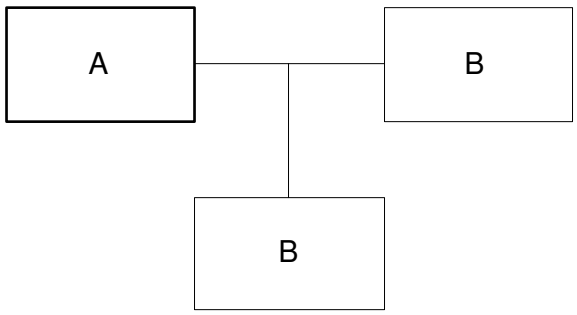
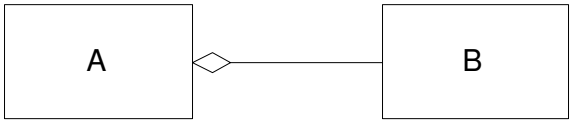
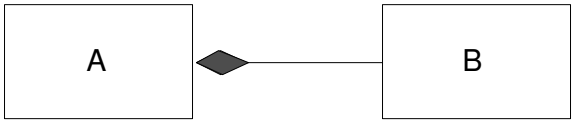
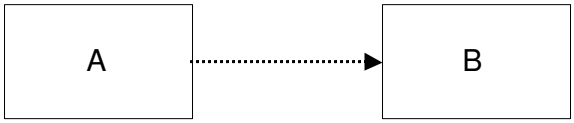
Figura A.5: Relación de uso.

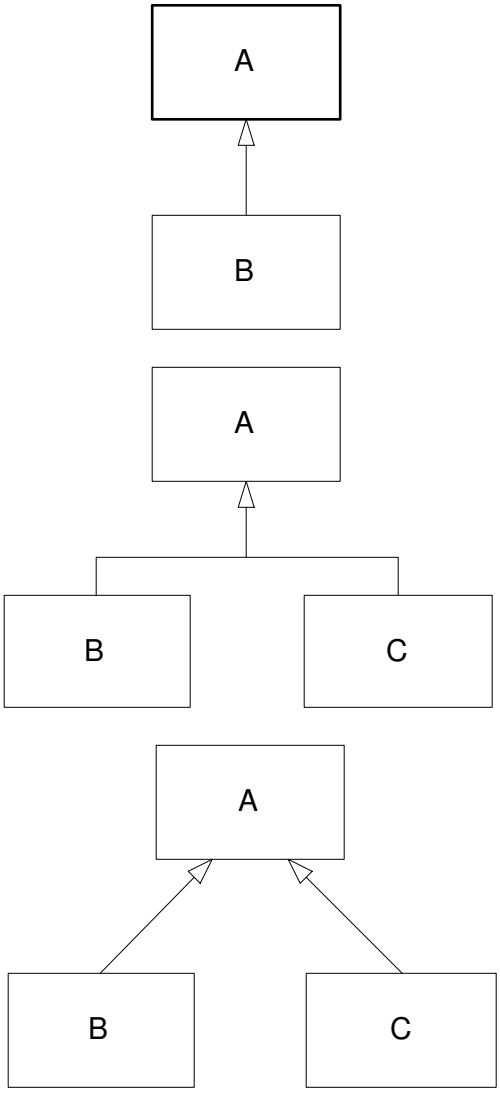
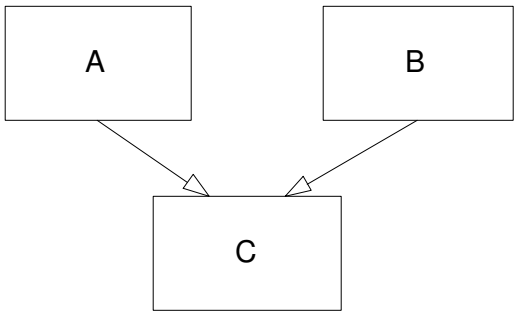
Apéndice B

Notaciones Básicas UML

En este apéndice se muestran algunas de las notaciones de modelado básicas de UML. Las mismas se anexan como posible referencia para la comprensión de los modelos UML del sistema integrado, mostrados en el capítulo 4.

Elemento de Modelado	**Representación UML**
Clases	
Visibilidad	+ público # protegido - privado
Asociación	

Elemento de Modelado	Representación UML
Multiplicidad	1 0..1 * 3..5 , 7 , 15 3..*
Clase Asociación	
Agregación	
	
Dependencia	

Elemento de Modelado	Representación UML
Herencia Simple	
Herencia Múltiple	

Apéndice C

Atributos y Métodos del Modelo de Clases

A modo de referencia, en este apéndice se da una breve descripción de los atributos que forman parte de las clases del modelo de implementación del Agente Integrador. El listado de métodos de las clases no se incluye, por ser éste muy extenso. Sin embargo, se anexa un CD-ROM que contiene todas las clases (donde se encuentran todos los atributos y métodos) de la implementación de prototipo.

Clase Personal

nombre: Nombre de la persona
usuario: Nombre de Usuario de la persona necesario para su identificación en el dominio
passwd: Password de la persona
telefono: Teléfono de la persona

Clase Mantenimiento

Hereda atributos de la clase **Personal**

Clase Operaciones

Hereda atributos de la clase **Personal**

Clase Tecnologia

nombre: Nombre de la Tecnología

descripción: Descripción de la Tecnología en el dominio del negocio

fabricante: Nombre del fabricante de la tecnología

tipo: Tipo de Tecnología

fechaCompra: Fecha de Compra de la Tecnología

fechaInstalacion: Fecha de instalación de la Tecnología dentro del sistema de negocios

Clase Hardware

ram: Cantidad de memoria RAM requerida por la Tecnología

dd: Cantidad en Megabytes requeridos por la Tecnología

procesador: Tipo y/o velocidad del procesador requerida por la tecnología

video: Características o especificaciones de video requeridas

Hereda atributos de la clase **Tecnología**

Clase Software

sistOper: Sistema Operativo donde corre la tecnología

versión: Versión o nivel de parches que debe poseer el Sistema Operativo como requerimiento de la tecnología

Hereda atributos de la clase **Tecnología**

Clase Aplicacion

nombre: Nombre de la aplicación del dominio

descripción: Descripción de la aplicación del dominio

tipo: Tipo de aplicación = GIS, SBD, SI, SIW, etc.

categoría: Categoría de la aplicación = por eventos, periódica, intervención manual

status: Status de la aplicación = A: activa, S: suspendida, D: deshabilitada

versión: Versión de la Aplicación

Clase AgenteIntegrador

Hereda atributos de la clase **Aplicación**

Clase InterfazCORBA

Transacción: Nombre de la función

numArg: Número de Argumentos que utiliza la función

tipoRes: Tipo de resultado que devuelve la función. Puede ser float, double, short...

status: Mantiene el status de la función. Puede ser = Falso:no activo, Verdadero: activo

Clase Argumento

nomArg: Nombre del Argumento

tipoArg: Tipo de dato del argumento en IDL, puede ser = float, double, long, short,...

dir: Indica la dirección de uso del argumento, puede ser = in, out, inout

Clase Regla

nomRegla: Nombre de la Regla *fechaCreacion*: Fecha de creación de la Regla

fechaUltMod: Fecha de última modificación de la Regla

Clase Actuacion

Hereda atributos de la clase **Regla**

Clase Estado

Hereda atributos de la clase **Regla**

Clase Evaluacion

Hereda atributos de la clase **Regla**

Clase Observacion

Hereda atributos de la clase **Regla**

Clase Sensado

Hereda atributos de la clase **Regla**

Clase Visualización

Hereda atributos de la clase **Regla**

Clase Condicion

opComparacion: Operador de comparación = =, <>, <=, <, >, >=
valor: Valor de comparación, depende del tipo de retorno de la función

Clase CondicionCompuesta

opLogico: Operador lógico = y, o,

Clase ObjetoDeNegocio

nombre: Nombre del objeto generado por una aplicación del sistema integrado
tipoObj: Tipo del objeto
descripción: Descripción del objeto de negocio
longitud: Longitud del objeto
info: Contenido del objeto en secuencias de bytes

Apéndice D

Ejemplo de Prueba para el Prototipo

Para el ejemplo, se toma una parte (figura D.1)[1] del diagrama de producción de la empresa Aguas de Mérida (figura D.2)[2]. A partir de dicha porción, se establecieron aplicaciones, funcionalidades de las mismas -con sus argumentos-, reglas y objetos de negocios; todos ellos ficticios, a fin de proporcionarle datos al prototipo desarrollado en esta investigación.

Aplicaciones:

A. Control de tanques (nivel, caudal)

B. Control de fuentes de captación (caudal)

C. Control de Bombas (presión)

D. Control de fuentes de producción (caudal, producción)

E. Control de zonas de abastecimiento (consumo, producción)

Funcionalidades de las aplicaciones:

La descripción de los argumentos de las funcionalidades pertenecientes a las aplicaciones, se muestran en el cuadro D.1. A continuación se presentan las funcionalidades, ideadas, para cada una de las aplicaciones nombradas anteriormente.

[1] Mostrada al final de éste apéndice
[2] Igual que 1

Argumento	Descripción
idT	Identificación del Tanque
idFC	Identificación de la Fuente de Captación
idB	Identificación de la Bomba
idFP	Identificación de la Fuente de Producción
idZ	Identificación de la Zona de Abastecimiento
pp	Proporción de apertura/cierre de válvulas o compuertas, aumento/disminución de la producción o presión de las bombas

Tabla D.1: Argumentos de las funcionalidades de las aplicaciones

Aplicación A (Control de tanques):

- leerNivelTanque(idT)
- leerCaudalTanque(idT)
- abrirValvulaTanque(idT,pp)
- cerrarValvulaTanque(idT,pp)

Aplicación B (Control de fuentes de captación):

- leerCaudalFuente(idFC)
- abrirCompuertaFuente(idFC,pp)
- cerrarCompuertaFuente(idFC,pp)

Aplicación C (Control de bombas):

- leerPresionBomba(idB)
- leerCaudalBomba(idB)
- aumentarPresionBombeo(idB,pp)
- disminuirPresionBombeo(idB,pp)

Aplicación D (Control de fuentes de producción):

- leerCaudalPlanta(idFP)
- leerProducciónPlanta(idFP)
- leerCloroPlanta(idFP)
- abrirValvulaPlanta(idFP,pp)
- cerrarValvulaPlanta(idFP,pp)
- aumentarProduccionPlanta(idFP,pp)
- disminuirProduccionPlanta(idFP,pp)
- aumentarCloroPlanta(idFP,pp)
- disminuirCloroPlanta(idFP,pp)

Aplicación E (Control de zonas de abastecimiento):

- leerConsumoZona(idZ)
- leerCaudalZona(idZ)

Reglas de Negocios

Tanques:

- ***Si*** nivel del tanque "La Vuelta" >1500 m3
  entonces cerrar la válvula de entrada del tanque "La Vuelta" en 3/4
- ***Si*** nivel del tanque "La Vuelta" <300 m3
  entonces abrir la válvula de entrada del tanque "La Vuelta" en 3/4
- ***Si*** caudal de entrada del tanque "La Vuelta" >900 l/s y la válvula de entrada del tanque "La Vuelta" esta abierta $<\frac{1}{2}$
  entonces cerrar la válvula de entrada del tanque "La Vuelta" en 1/4 y disminuir la producción de la fuente de producción "Burgoin" en 1/5.

- ***Si*** caudal de entrada del tanque "La Vuelta" <300 l/s y la válvula de entrada del tanque "La Vuelta" esta cerrada $<\frac{1}{2}$
  entonces abrir la válvula de entrada del tanque "La Vuelta" en 1/4 y aumentar la producción de la fuente de producción "Burgoin" en 1/5.

- ***Si*** nivel del tanque "La Hechicera" >700 m3
  entonces cerrar la valvula de entrada del tanque "La Hechicera" en 1/2

- ***Si*** nivel del tanque "La Hechicera" <300 m3
  entonces abrir la valvula de entrada del tanque "La Hechicera" en 1/2

- ***Si*** caudal de entrada del tanque "La Hechicera" >6 l/s y la válvula de entrada del tanque "La Hechicera" esta abierta $<\frac{1}{2}$
  entonces cerrar la válvula de entrada del tanque "La Hechicera" en 1/4 y disminuir la presión de bombeo "Los Chorros" en 1/5

- ***Si*** caudal de entrada del tanque "La Hechicera" <2 l/s y la válvula de entrada del tanque "La Hechicera" esta cerrada $<\frac{1}{2}$
  entonces abrir la válvula de entrada del tanque "La Hechicera" en 1/4 y aumentar la presión de bombeo "Los Chorros" en 1/5

Fuentes de Captación:

- ***Si*** caudal de salida de la fuente "Rio Mucujun" >600 l/s
  entonces cerrar compuertas de la fuente "Rio Mucujun" en 9/10

- ***Si*** caudal de salida la fuente "Rio Mucujun" <300 l/s
  entonces abrir compuertas de la fuente "Rio Mucujun" en $\frac{1}{2}$

Fuentes de Producción:

- ***Si*** caudal de entrada de la fuente producción "Burgoin" >1100 l/s
  entonces cerrar válvula de entrada de la fuente de producción "Burgoin" en 3/10

- ***Si*** caudal de entrada de la fuente producción "Burgoin" <800 l/s
  entonces abrir válvula de entrada de la fuente de producción "Burgoin" en 4/5

- ***Si*** nivel de producción de la fuente producción "Burgoin" >1300 l/s
  entonces disminuir la producción de la fuente de producción "Burgoin" en 1/4

- ***Si*** nivel de producción de la fuente producción "Burgoin" <900 l/s
  entonces aumentar la producción de la fuente de producción "Burgoin" en 1/5

Bombas:

- ***Si*** caudal de entrada de la estación de bombeo "Los Chorros" >40 l/s
  entonces cerrar válvula de la estación de bombeo "Los Chorros" en 1/3

- ***Si*** caudal de entrada de la estación de bombeo "Los Chorros" <10 l/s
  entonces abrir válvula de la estación de bombeo "Los Chorros" en 1/2

Zonas de Abastecimiento:

- ***Si*** caudal de entrada a la zona de abastecimiento "Sector Los Chorros" >35 l/s
  entonces cerrar válvula de la estación de bombeo "Los Chorros" en 1/9 y disminuir presión de la estación de bombeo "Los Chorros" en $\frac{1}{4}$

- ***Si*** caudal de entrada a la zona de abastecimiento "Sector Los Chorros" <20 l/s
  entonces abrir válvula de la estación de bombeo "Los Chorros" en 1/7 y aumentar presión de la estación de bombeo "Los Chorros" en 5/8

- ***Si*** consumo real de la zona de abastecimiento "Sector Los Chorros" >35 l/s
  entonces aumentar la producción de la fuente de producción "Burgoin´´ en 1/5

- ***Si*** consumo real de la zona de abastecimiento "Sector Los Chorros" <20 l/s
  entonces disminuir la producción de la fuente de producción "Burgoin" en 1/5

Reglas de Negocios especificadas de acuerdo a las funcionalidades de las aplicaciones

Tanques

- ***Si*** (leerNivelTanque("La Vuelta") >1500)
  entonces (cerrarValvulaTanque("La Vuelta", $\frac{3}{4}$))

- ***Si*** (leerNivelTanque("La Vuelta") <300)
  entonces abrirValvulaTanque("La Vuelta", $\frac{3}{4}$)

- ***Si*** (leerCaudalTanque("La Vuelta") >900)
  entonces (cerrarValvulaTanque("La Vuelta", 1/4)) AND
  (disminuirProduccionPlanta("Burgoin",1/5))

- ***Si*** (leerCaudalTanque("La Vuelta") <300)
  entonces (abrirValvulaTanque("La Vuelta", 1/4)) AND
  (aumentarProduccionPlanta("Burgoin",1/5))

- ***Si*** (leerNivelTanque("La Hechicera") >700)
  entonces (cerrarValvulaTanque("La Hechicera", 1/2))

- ***Si*** (leerNivelTanque("La Hechicera") <300)
  entonces abrirValvulaTanque("La Hechicera", 1/2)

- ***Si*** (leerCaudalTanque("La Hechicera") >6)
  entonces (cerrarValvulaTanque("La Hechicera", 1/4)) AND
  (disminuirPresionBombeo("Los Chorros", 1/5))

- ***Si*** (leerCaudalTanque("La Hechicera") <2)
  entonces (abrirValvulaTanque("La Hechicera", 1/4)) AND
  (aumentarPresionBombeo("Los Chorros", 1/5))

Fuentes de Captación:

- ***Si*** (leerCaudalFuente(Rio Mucujun") >600)
  entonces (cerrarCompuertaFuente(Rio Mucujun", 9/10))

- ***Si*** (leerCaudalFuente(Rio Mucujun") <300)
  entonces (cerrarCompuertaFuente(Rio Mucujun", 1/2))

Fuentes de Producción

- ***Si*** (caudalPlanta("Burgoin") >1100)
  entonces (cerrarValvulaPlanta("Burgoin",3/10))

- ***Si*** (caudalPlanta("Burgoin") <800)
  entonces (abrirValvulaPlanta("Burgoin",4/5))

- ***Si*** (produccionPlanta("Burgoin") >1300)
  entonces ((disminuirProduccionPlanta("Burgoin",1/4))

- ***Si*** ((produccionPlanta("Burgoin") <900)
  entonces ((aumentarProduccionPlanta("Burgoin",1/5))

Bombas:

- ***Si*** leerCaudalBomba("Los Chorros") >40)
  entonces (cerrarValvulaBombeo("Los Chorros",1/3)

- ***Si*** (leerCaudalBomba("Los Chorros") <10)
  entonces (abrirValvulaBombeo("Los Chorros",1/2))

Zonas de Abastecimiento:

- ***Si*** (leerCaudalZona("Sector Los Chorros") >35)
  entonces (cerrarValvulaBombeo("Los Chorros",1/9)) AND
  (disminuirPresionBombeo("Los Chorros",$\frac{1}{4}$))

- ***Si*** (leerCaudalZona("Sector Los Chorros") >20)
  entonces (abrirValvulaBombeo("Los Chorros",1/7)) AND
  (aumentarPresionBombeo("Los Chorros",5/8))

- ***Si*** (leerConsumoZona("Sector Los Chorros") >35)
  entonces (aumentarProduccionPlanta("Burgoin",1/5))

- ***Si*** (leerConsumoZona("Sector Los Chorros") <20)
  entonces (disminuirProduccionPlanta("Burgoin",1/5))

Objetos de Negocios

Los Objetos de Negocios son los siguientes: Tanques, Fuentes de Captación, Fuentes de Producción, Bombas, y Zonas de Abastecimiento.

En el prototipo, ellos se almacenan con los atributos que se muestran en la tabla D.2. Estos atributos también son considerados Objetos de Negocios. Por ejemplo, en tabla se observa que el objeto Tanque posee los atributos Nivel y Caudal; en el sistema, ambos atributos también son almacenados como Objetos de Negocios que son sub-objetos del objeto Tanque.

Objeto de Negocio	Atributos
Tanque	Nivel Caudal
Fuente de Captación	Caudal
Fuente de Producción	Caudal Producción
Bomba	Presión
Zona de Abastecimiento	Consumo Producción

Tabla D.2: Objetos de Negocios y sus atributos

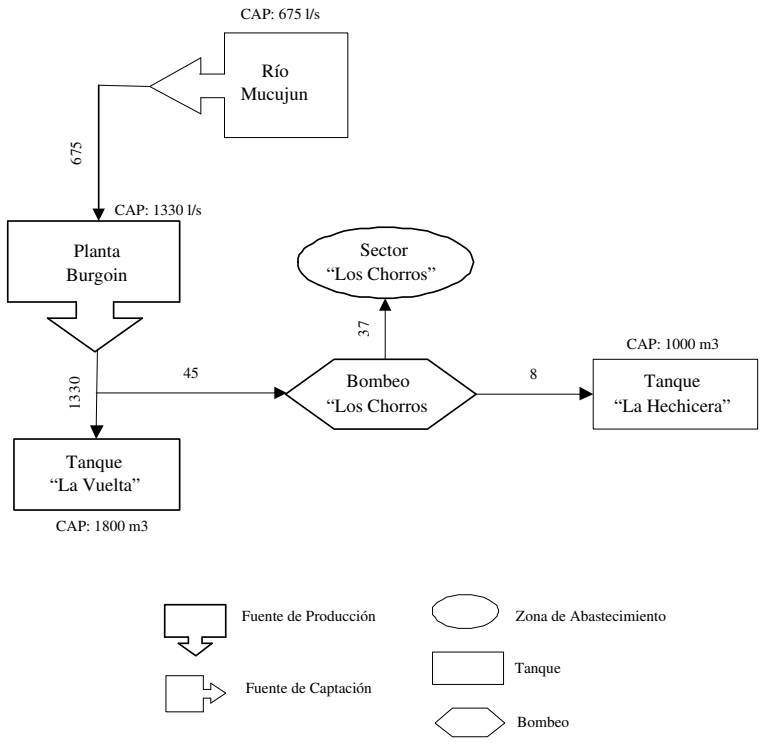


Figura D.1: Porción del Diagrama de Aguas de Mérida utilizada en el Ejemplo de Prueba

Figura D.2: Diagrama de Aguas de Mérida