AUTOMATIZACIÓN DEL PLACEMENT TEST DE LA UNIDAD DE TRADUCTORES BASADO EN COMPONENTES SW

Lina María Castillo Paredes

Director: Ing. Oscar Mauricio Caicedo

Universidad Del Cauca
Facultad De Ingeniería Electrónica Y Telecomunicaciones
Departamento De Telemática
Línea de Investigación de Ingeniería de Sistemas Telemáticos
Popayán, Mayo de 2004

TABLA DE CONTENIDO

Pág.

INTRODUCCIÓN

La evolución en los esquemas de desarrollo de software se ha centrado en la búsqueda de alternativas que faciliten las etapas de desarrollo e integración del mismo. Como consecuencia, ha surgido un nuevo esquema de desarrollo denominado Desarrollo de Software Basado en Componentes, en el que se destacan aspectos importantes como el reuso del software.

El siguiente documento es la síntesis de un trabajo que se desarrolló en dos fases principales:

La primera fase, incluye una revisión teórica sobre el origen del desarrollo Software basado en Componentes y el impacto que ha tenido en el desarrollo de Software tradicional, tratado en el capítulo I, una comparación de las plataformas empresariales más importantes que se encuentra en el capítulo II, y la descripción detallada de la plataforma J2EE en el capítulo III.

La segunda fase, incluye el desarrollo de una aplicación bajo la plataforma J2EE. Aspectos del modelo del negocio y del modelo de análisis de esta aplicación se encuentran en el IV capítulo.

CAPÍTULO I
PROGRAMACIÓN BASADA EN COMPONENTES

La programación basada en componentes es de gran interés para la comunidad de la ingeniería del software. El objetivo es crear una colección de componentes reusables que puedan ser utilizados para el desarrollo de aplicaciones. En este capítulo, se discutirá los problemas del desarrollo software tradicional y los antecedentes que dieron inicio al desarrollo de componentes como una posible solución y respuesta a la crisis del mismo. Además, se abordarán otros temas relacionados tales como el concepto de software basado en componentes, el concepto de componentes, interfaces, frameworks y componentes, modelos de desarrollo y beneficios y riesgos del uso de componentes.

1.1 ANTECEDENTES

Aunque el desarrollo software es una disciplina relativamente joven, su rápido crecimiento se ve reflejado en la gran cantidad de Sistemas Hardware que contienen diferentes tipos de software. Esta diversidad de software nos ha dado Sistemas Hardware más avanzados y al mismo tiempo nos ha hecho más dependientes de estas nuevas tecnologías. De la misma manera que otros productos hasta antes de la revolución industrial, el software es hoy en día producido en su mayoría de manera individual y empezando desde cero. Como consecuencia, cuando el producto es lanzado al mercado, resulta a menudo pobremente documentado y mucho del conocimiento del software se encuentra aún en la mente de los programadores. Debido a esto, dos de los principales problemas del desarrollo software son la calidad inadecuada y el gran incremento en el costo de desarrollo.

Por otro lado, el creciente desarrollo de hardware demanda una rápida producción de software en tiempos en que la complejidad de los sistemas de software modernos es muy alta. Además, la forma tradicional del desarrollo de programas ha conducido a muchos problemas, entre ellos la dificultad de los clientes para expresar a los programadores lo que se requiere y dado que el tiempo de entrega se hace cada vez más corto, la documentación y las pruebas se ven afectadas. Así, cuando el producto final es entregado, es a menudo de baja calidad. Finalmente, la carencia de documentación significa que es difícil brindar soporte técnico al software y que es virtualmente imposible reusarlo en un nuevo proyecto porque no se esta seguro de cual es su función.

El aspecto económico de la crisis del software es agravado por el hecho de que el costo relativo del software comparado con el costo del hardware a incrementado en un factor de 10:1 [Shaw, 1996]. Una razón significativa para esto es que mientras los precios del hardware han caído dramáticamente, los costos de desarrollo software no puede alcanzar esta reducción debido a su naturaleza de labor intensiva. Como respuesta a la necesidad de buscar alternativas que permitan hacer más fáciles las tareas de desarrollo e integración de software, uno de los más recientes modelos de desarrollo software es el de componentes. El desarrollo de software basado en componentes (CBSD) es un paradigma de producción de software relativamente nuevo, centrado en la construcción de sistemas de software grandes integrados por componentes fácilmente accesibles. Estructurar un sistema con componentes independientes tiene muchas ventajas. Es fácil distribuir los componentes entre varios ingenieros para permitir el desarrollo paralelo; el mantenimiento es mucho más fácil cuando interfaces claras han sido diseñadas para los componentes, porque se pueden hacer cambios locales sin tener efectos desconocidos en el sistema; y, si las interrelaciones de componentes son claramente documentadas y mantenidas a un mínimo, resulta más fácil intercambiar componentes e incorporar nuevos en el sistema. Aspectos como el reuso y la interoperabilidad, muy importantes en la

actualidad, son característicos de este enfoque y puede visualizarse como un modelo de desarrollo bastante prometedor.

1.2 EL CONCEPTO DE SOFTWARE BASADO EN COMPONENTES

El concepto de Desarrollo Software Basado en Componentes (CBSD) considera a todas aquellas aplicaciones que se construyen haciendo uso de componentes software. Una definición presentada para este concepto habla de un enfoque de desarrollo de software en el cual todos los artefactos -desde código ejecutable para la especificación de la interfaz, arquitecturas, modelos de negocios, escalar a aplicaciones completas y descomposición de sistemas en partes- pueden ser construidos por ensamble, adaptación y conexión de componentes existentes dentro de una variedad de configuraciones [D´Souza y Wills, 1998]. Es claro que las características descritas para este enfoque de desarrollo suenan bastante atractivas; de hecho, muchos autores resaltan algunas otras como el reuso de implementaciones e interfaces, la facilidad de mantenimiento y actualización de aplicaciones o la posibilidad de desarrollo en paralelo de diferentes partes de un sistema. Ante este panorama, es importante hacer mención de que al ser el CBSD un área de investigación relativamente joven, existen muchos aspectos que aún no son lo suficientemente robustos. Carencias como el soporte disponible, la selección de componentes, el manejo de excepciones o bien la prueba de integración de componentes son algunos de ellos [Fernández, 1999].

1.3 EL CONCEPTO DE COMPONENTES

La palabra componente nos hace pensar en una unidad o elemento con propósito bien definido que, trabajando en conjunto con otras, puede ofrecer alguna funcionalidad compleja. Transportando este concepto al contexto de ingeniería de software, específicamente de desarrollo basado en componentes, un componente es la pieza elemental de este enfoque de desarrollo, de esta forma, a partir de componentes existentes pueden llegarse a construir aplicaciones completas.

Los componentes son el corazón del CBSD y son definidos en varias formas desde diferentes puntos de vista. A continuación se da una recopilación de definiciones acerca del concepto de componentes.

En términos formales, un componente es una pieza de software que cumple con dos características: no depende de la aplicación que la utiliza y, se puede emplear en diversas aplicaciones [Liskov, 2000].

Según los asistentes a la ECOOP96[1], un componente es una unidad de composición con interfaces definidas contractualmente y dependencias contextuales explícitas solamente. Un componente de software puede desplegarse independientemente y estar sujeto a composición por terceros [Szyperski et al., 2002].

Una definición más orientada a un enfoque de desarrollo, plantea que un componente es un paquete coherente de una implementación de software que puede ser independientemente desarrollado e intregado, tiene interfaces explícitas y bien especificadas para informar los servicios que provee y los servicios que espera de otros, puede estar constituido por otros componentes, quizás mediante la especialización de algunas de sus propiedades, sin que esto signifique modificar los componentes mismos [D´Souza y Wills, 1998].

Un componente software es un fragmento de un sistema software que puede ser ensamblado con otros fragmentos para formar piezas mas grandes o aplicaciones completas [Brown, 1999].

Un componente es una implementación que: (a) realiza un conjunto de funciones relacionadas, (b) puede ser independientemente desarrollado, intregado e instalado, (c) tiene un conjunto de interfaces para los servicios proporcionados y otro para los servicios requeridos, (d) permite tener acceso a

[1] 10th European Conference for Object-Oriented Programming. Linz, Austria, July 8-12, 1996.

los datos y al comportamiento sólo a través de sus interfaces, (e) opcionalmente admite una configuración controlada [IBM-WebSphere, 2001]).

Un componente es algo que se puede componer junto con otras partes para formar una composición o ensamblaje [OMG, 2001].

Desde el punto de vista de la tecnología COM de Microsoft, un componente es “una pieza de software compilada, la cual ofrece un servicio” [Ciupke y Schmidt, 1996].

Un componente es “una unidad de despliegue independiente. Una unidad compuesta de terceras partes, y no tiene estado de persistencia” [Gamma et al., 1994].

Un componente es definido también como “una colección de objetos que cooperan entre si, con límites claramente definidos para otros objetos o componentes” [Szyperski y Pfister, 1996]. Así, los objetos dentro de un componente son típicamente entrelazados fuertemente, mientras la interacción a través de los límites del componente es relativamente débil.

Componentes son “unidades de software que son independientes del contexto en el dominio técnico y conceptual” [Ciupke y Schmidt, 1996].

De acuerdo con [Szyperski et al., 2002] las propiedades que un componente como mínimo debe tener son:

- *Es una entidad comerciable*. Un componente es una pieza binaria, encapsulada, e independiente de software, que cualquiera puede comprar en el mercado abierto.

- *No es una aplicación completa*. Un componente puede ser combinado con otros componentes para formar una aplicación completa. Está diseñado para hacer un conjunto limitado de tareas en un dominio de la aplicación.

- *Puede ser usado en combinaciones impredecibles*. Típicamente, los componentes pueden ser combinados con otros componentes de la misma familia usando conectores (“plug and play”).

- *Tiene una interfaz bien definida*. Como un objeto clásico, un componente puede solamente ser manipulado a través de su interfaz. Esto es, cómo el componente expone su función al mundo exterior.

- *Es un objeto interoperable*. Un componente puede ser invocado como un objeto a través de espacios de direcciones, redes, lenguajes, sistemas operativos, y herramientas. Es una entidad de software independiente del sistema.

Del análisis de las ideas presentadas en las definiciones de [Liskov, 2000] y [Szyperski et al., 2002] y, de acuerdo con lo que plantea [Fernández, 1999], se pueden identificar características en común:

- *Orientación al reuso*, lo que nos hace pensar en algo que se puede tomar y utilizar más de una vez.

- *Interoperabilidad*, el componente no puede solo usarse de forma aislada, sino que puede integrarse con otros y trabajar conjuntamente.

- *Función significativa*, el componente tiene una función bien definida, identificable y diferenciable.

- *Encapsulamiento*, los componentes ocultan detalles acerca de su implementación.

Tomando la definición de [D´Souza y Wills, 1998], se pueden enriquecer algunas de las características presentadas antes, considerando también:

- Una lista de interfaces proveídas y requeridas, las cuales representan mecanismos indispensables para la interoperabilidad de los componentes.

- Una especificación externa, medio que permite principalmente identificar la función del componente.

- El código de validación, necesario para verificar, cuando se utiliza más de un componente, para saber si se conectaron correctamente.

- El código ejecutable del componente.

De lo expuesto anteriormente se puede concluir que: un componente debe ser capaz de conectarse ("plug and play"[2]) con otros componentes para que este pueda ser compuesto en tiempo de ejecución sin re-compilación. Además, los componentes deberían separar su interfaz de su implementación, ser capaces de interoperar sobre una arquitectura predefinida, y sus interfaces deberían estar estandarizadas ya que estas son ampliamente reusadas. En resumen, un componente es una pieza autónoma y reusable de software independiente de cualquier aplicación.

1.3.1 Niveles de Granularidad de un Componente

Los niveles de granularidad de un componente definen las diferentes categorías de componentes:

- *Componente distribuido*. Esta es considerada la granularidad más baja de un componente. Es una forma específica del concepto componente usual en la industria, por ejemplo, este debe ser implementado como un *Enterprise JavaBean*, un componente CORBA o DCOM.
- *Componente de negocio*. Un componente que implementa un concepto de negocio autónomo simple. Este usualmente consiste de uno o más componentes distribuidos que juntos manejan los diferentes aspectos de

[2] «Conecte y Use» Este término se utiliza para señalar que un dispositivo puede ser instalado sin necesidad de recurrir a un programa de instalación o software adicional.

distribución requeridos por los componentes de negocio. Esto es, el componente de negocio es un objeto simple que debe físicamente ser desplegado a través de dos o quizá más máquinas.

- *Sistema de componentes de negocio*. Un grupo de componentes que cooperan para entregar un conjunto consistente de funcionalidad requerida por una necesidad de negocio específica.

1.3.2 Objetos y Componentes

Las nociones de instanciación, identidad y encapsulación conllevaron a la noción de objetos. En contraste con los componentes, un objeto es una unidad de instanciación; tiene estado, y encapsula su estado y comportamiento. Esto significa que los objetos no pueden ser parcialmente instanciados. Como un objeto tiene estado individual, este también tiene una identidad única para identificarse a pesar de los cambios de estado. Como los objetos son instanciados es necesaria la construcción que describa el espacio de estado, el estado inicial y el comportamiento de un nuevo objeto. Este plan es llamado una “clase” y esta existe antes de que los objetos existan [Gamma et al., 1994].

Típicamente, un componente se conforma a partir de objetos y por lo tanto debería normalmente contener una o más clases [D´Souza y Wills, 1998; Fernández, 1999]. Sin embargo, no es necesario para un componente contener clases. Los componentes deben contener (pero no necesariamente) procedimientos, variables globales (estáticas), etc. Los objetos creados en un componente pueden dejar el componente y convertirse visibles a los clientes de componentes [Szyperski, 1999].

Los componentes pueden depender de otros componentes. Las super clases de una clase no necesariamente necesitan residir en el mismo componente como la clase en si misma. Los objetos dentro de un componente simple pueden acceder a las implementaciones de otros. Sin embargo, el acceder a implementaciones desde el exterior del componente debe ser prevenida.

Los componentes llevan instancias que actúan en tiempo de ejecución. En el caso más simple, un componente es una clase y las instancias que lleva son objetos de esa clase. Sin embargo, la mayoría de componentes (COM o JavaBeans) consistirán de muchas clases. Un Java Bean esta externamente representado por una clase simple, y por esto es un tipo simple de objeto que representa todas las instanciaciones posibles o usos de ese componente. Un componente COM es más flexible. Este puede presentarse a los clientes como una colección arbitraria de objetos cuyos clientes solamente ven un conjunto de interfaces no relacionadas. En JavaBeans o CORBA, múltiples interfaces son últimamente combinadas en una clase implementada. Esto facilita un manejo apropiado de casos importantes tales como componentes que soportan múltiples versiones de una interfaz, donde la implementación exacta de un método particular compartido por todas estas versiones necesita depender de la versión de la interfaz que el cliente esta utilizando.

Mientras los componentes capturan la naturaleza estática de un fragmento software, los objetos capturan su naturaleza dinámica. Si se considera todo como “dinámico” se podría eliminar esta distinción. Sin embargo, es un principio de la ingeniería del software tratar de fortalecer la descripción estática de sistemas tanto como sea posible. Este es el papel de arquitecturas que asignan a los componentes su lugar. El papel de los objetos es capturar la naturaleza dinámica de los crecientes sistemas construidos a partir de componentes.

Algunas preguntas importantes a resolver son:

Es un objeto un componente? Los componentes son artefactos de software, y representan el trabajo de los desarrolladores de software y sus herramientas; los objetos son instancias individuales identificables creadas en sistemas en operación por la ejecución de código que es una parte de un componente. En el sentido estricto, un objeto no es un componente. Partes del código en un componente pueden definir los formularios para objetos, por ejemplo usando

clases de la programación orientada a objetos. Este es el código del componente que es reusado. Dicho esto, un componente, cuando es desplegado y ejecutado, a menudo será manifestado como una colección de objetos; y puede algunas veces ser tratado como un objeto de amplia granularidad. Así, algunas veces se puede utilizar el término “componente” de una manera más flexible para referirse al objeto, o conjunto de objetos, el cual manifiesta un uso particular de un componente en una aplicación. Aun si un componente no es implementado con la tecnología de objetos, una simple unidad de amplia granularidad, cuando se esta ejecutando, puede usualmente ser modelada como un objeto.

Es una clase un componente? Una clase como tal, no es una unidad contenida en si misma como es requerido por un componente. En particular, un componente debe incluir descripciones explícitas de las interfaces que éste implementa, y las interfaces que éste espera desde otros componentes. En general, un componente podría implementar sus interfaces por exposición directa de estas a los clientes, o por implementación de clases que proveen estas interfaces.

Como se ha descrito, existen algunos aspectos comunes entre los objetos y los componentes que pueden hacer que estos se entiendan como conceptos equivalentes. Aunque hay similitudes, existen marcadas diferencias que permiten entenderlos como entidades diferentes, aunque no por ello excluyentes una de otra. Con el propósito de diferenciar a los componentes de los objetos, a continuación se presentan algunas características:

- Los componentes tienen persistencia, los objetos solo utilizan memoria.

- Los componentes utilizan otras formas de comunicación como *eventos*, en lugar de limitarse al uso de mensajes como los objetos.

- La granularidad de los componentes es mayor que la de los objetos, un componente puede presentarse como varios objetos de diferentes clases.

- Los componentes definen el conjunto de interfaces utilizadas en más detalle.

- El paquete que contiene el componente, incluye la especificación de las interfaces proveídas y requeridas, mientras que en los objetos estas especificaciones se concentran en las operaciones proporcionadas.

En conclusión, un objeto componente es un objeto soportado por un componente identificado. Así, los componentes y objetos permitirán la construcción de software de la siguiente generación.

1.4 INTERFACES

Interfaces son los puntos de acceso a un componente. Para ser precisos, una interfaz es una colección de operaciones que son usadas para especificar un servicio de un componente [Kruchten, 1998]. En una interfaz, cada semántica de una operación es especificada. Esta especificación sirve para que los proveedores implementen la interfaz y para que los clientes la usen. La interfaz de un componente es importante para la composición y utilización de componentes por usuarios, permitiendo encontrar componentes apropiados entendiendo su propósito, funcionalidad, uso y restricciones [Bent, 1994]. Los componentes pueden implementar (exportar) una o mas interfaces y también pueden usar (importar) interfaces de otros componentes. Por lo tanto, interfaces exportadas corresponden a servicios que un componente provee, e interfaces importadas corresponden a servicios que un componente necesita para implementar los servicios a exportar [Bent, 1994].

Las interfaces permiten a los componentes heterogéneos interoperar. En general, la aproximación a las interfaces de un componente es sintáctica, lo cual es poco útil para resolver problemas en corto tiempo. Problemas de semántica relacionados con dependencias e interacciones de contexto pueden ser de importancia. La interfaz define un estándar que los vendedores de componentes tienen que proveer y que los usuarios de componentes pueden

esperar; es decir, una interfaz puede ser comparada con un contrato [Meyer, 1997]. Por lo tanto, como mínimo, las interfaces de componentes deberían proveer un contrato que estipule los servicios que un componente ofrece y los servicios que este requiere para cumplir sus obligaciones. Los contratos involucran al menos dos partes: un vendedor que provee los bienes y un comprador quien consume los bienes en una forma particular. Existen buenos y malos contratos. Un buen contrato debería ser claro, completo y conciso. Un mal contrato es ambiguo, carece de puntos importantes, o presenta detalles irrelevantes.

1.5 FRAMEWORKS Y COMPONENTES

Los componentes son piezas que pueden ser conectadas en un sistema. Si esto ocurre, debe haber algo en qué conectarlos. El objetivo de los componentes software no es solamente resolver cómo diseñar y construir una pieza independiente y útil de software con una interfaz bien definida, sino también asegurarse de que hay sockets de software en el cual un componente compatible se acomode cuando es desplegado. En este contexto, el socket incluye la infraestructura técnica y otros componentes requeridos para que el componente opere correctamente. El *framework* puede ser entendido en sentido de socket como: un ambiente para componentes software con capacidad “plug and play”.

Antes de definir el concepto de frameworks es necesario conocer el concepto de patrones. Los patrones fueron inicialmente definidos como patrones de arquitectura [Appleton, 2000]. Los patrones involucran una descripción general de soluciones recurrentes a un problema periódico. Además, un patrón hace mas que solo identificar una solución; éste también explica por qué la solución es necesaria. Un patrón debe desempeñar las siguientes tareas:

- *Resuelve un problema*. Los patrones capturan soluciones, no solo principios abstractos o estrategias.

- *Es un concepto demostrado*. Los patrones capturan soluciones basados en experiencia, no en teorías o especulaciones.

- *La solución no es obvia*. Muchas técnicas de solución de problemas (paradigmas de diseños de software o métodos) tratan de derivar soluciones desde principios. Los mejores patrones generan una solución a un problema indirectamente.

- *Describe una relación*. Los patrones no solo describen módulos, también describen profundas estructuras de sistemas y mecanismos.

Cuando se trata de documentar una solución/diseño software, diferentes tipos de patrones a diferentes niveles de abstracción pueden ser usados. Los *patrones de arquitectura* expresan un esquema para los sistemas software. Este provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y guías para organizar las relaciones entre ellos. *Patrones de diseño* proveen un esquema para refinar los subsistemas o componentes de un sistema software, o las relaciones entre ellos. Este describe la estructura comúnmente recurrente de componentes de comunicación que resuelven un problema de diseño general en un contexto particular. Un *patrón idiomático* es un patrón de bajo nivel para un lenguaje de programación específico. Este describe como implementar aspectos particulares de componentes o las relaciones entre ellos usando las características del lenguaje dado.

La diferencia entre estos tres tipos de patrones esta en sus correspondientes niveles de abstracción y detalle. Los patrones de arquitectura son estrategias de alto nivel que conciernen a componentes de alta escala y las propiedades globales y mecanismos de un sistema. Los patrones de diseño son tácticas de escala media que le dan cuerpo a algunas de las estructuras y comportamientos de las entidades y sus relaciones. Ellos no influencian la estructura general de un sistema y en lugar de esto definen arquitecturas micro de subsistemas y componentes. Los patrones idiomáticos son técnicas de

programación de lenguajes específicos y paradigmas específicos que llenan detalles externos e internos de bajo nivel del comportamiento o estructura de un componente.

Los frameworks están estrechamente relacionados con los patrones. Ellos definen ciertos grupos de participantes y relaciones entre estos que podrían ser reusados en cualquier situación isomórfica. En comparación a los patrones, los frameworks definen una gran unidad de diseño y son mas especializados [Gamma et al., 1994]. Los frameworks también son definidos como unidades para compartir y reusar arquitecturas. Parece existir una distinción entre frameworks y frameworks de componentes. Mientras los frameworks describen una situación típica y reusable en general, los frameworks de componentes definen un “Tablero de circuitos” con slots vacíos en los cuales los componentes podrían ser insertados para crear una instancia de trabajo. Los requerimientos son especificados indicando a los componentes como conformarse para ser capaces de hacer el trabajo en el circuito. En la noción de frameworks formales se ha visto que el comportamiento del circuito (framework) puede ser especificado en términos de pre y post-condiciones del framework, invariantes e instanciaciones, en lo que participaran los componentes y las relaciones estáticas entre estos. Lo que sigue es formalizar la relación entre componentes y en más detalle especificar qué interfaces deberían tener los componentes para interactuar con otros componentes. Los slots abiertos podrían llenarse con componentes atómicos (cajas negras [Alexander, 1979]) o con otros frameworks que podrían estar compuestos de otros componentes. Así, un framework de componentes es definido como una colección de reglas e interfaces que gobiernan la interacción de componentes conectados en el framework, el cual contiene un diseño abstracto para solucionar una familia de problemas relacionados [Monday et al., 2000].

Un framework de un componente define tareas para componentes cargados dinámicamente y desarrollados independientemente, en lugar de tareas para clases que son entrelazadas en una aplicación monolítica. Un framework de un

componente debe proveer interfaces, posiblemente junto con algunos procedimientos. En contraste a los frameworks de aplicaciones, los frameworks de componentes son frameworks de cajas negras; por ejemplo, frameworks que pueden ser usados sin acceder a su código fuente y no necesitan ser una librería de clases. De hecho, un framework de un componente no debe contener código en absoluto; este debe ser solo una colección de interfaces. Esta es una gran diferencia con antiguas interpretaciones del término "framework", donde la existencia de código actual desempeñó un papel más importante. En los frameworks de componentes, el código actual, en la mayoría de los casos, existe para controlar la ejecución.

Un ejemplo de framework de componente bien definido es el ambiente de desarrollo de Visual Basic de Microsoft. Los formularios son construidos añadiendo componentes (controles) a un formulario originalmente vacío ("Tablero de Circuitos"). Después de esto, el desarrollador a través de un lenguaje de programación (Basic) añade variaciones al comportamiento del componente. Esto es hecho a través de una interfaz de mensajes que ocurre entre componentes y eventos que es aplicada a los componentes. El formulario en si mismo es también considerado como un componente. El framework es construido alrededor de las estructuras de una aplicación de ventanas típica.

1.6 MODELOS DE DESARROLLO

El modelo de desarrollo basado en componentes, ha adquirido el respaldo de compañías importantes, por un lado está Microsoft con su modelo COM+/DCOM y por otro lado empresas como Sun con su modelo de Java Beans, y HP, ORACLE e IBM que manejan sus propios estándares.

A continuación se describen tres de los modelos de desarrollo de Componentes más populares actualmente, COM+/DCOM, CORBA y Enterprise Java Beans.

1.6.1 COM y DCOM

El Component Object Model (COM), es la propuesta de Microsoft en materia de

componentes. Aunque inicialmente fue propuesto para resolver problemas en la construcción de documentos con OLE para Windows 3.1, este modelo ha sido enriquecido con el propósito de permitir la construcción de aplicaciones a partir de componentes. COM es esencialmente un esquema de integración, de modo que los componentes que se construyen para trabajar bajo este modelo, deben describir su comportamiento bajo las consideraciones de este esquema, el cual es popularmente conocido como estándar binario. Este estándar permite que las interfaces de los componentes se presenten a los clientes como un conjunto de apuntadores a tablas en memoria, llamadas tablas virtuales de funciones o vtables. El llamado de funciones entre componentes se realiza utilizando estos apuntadores, ocultándose así detalles de implementación, lo cual permite que componentes que estén escritos en diferentes lenguajes de programación de los que se pueden incluir a *C++, Small Talk, Ada, VisualBasic, Delphi* o *Power Builder*, puedan comunicarse. Dadas las características de las vtables, las interfaces no pueden definirse utilizando herencia múltiple, porque se debe considerar la posibilidad de tener varias vtables y más de un apuntador por interface. Una interfaz COM es una colección de métodos lógicamente relacionados que expresan cierta funcionalidad. Todas las interfaces COM derivan de IUnknown, y se identifican mediante un Identificador de Interfaz único a nivel global, Interface ID (IID). Las interfaces COM se definen utilizando un lenguaje especial denominado Interface Definition Language (IDL). La compilación de estas interfaces, produce una especie de librerías que contienen información sobre los metadescriptores del objeto, de sus interfaces, de las estructuras definidas por el usuario y de los elementos referidos por el componente, así como un mapa de memoria para las operaciones públicas.

Distributed COM, mejor conocido como DCOM, es una extensión de COM que se implementa como una respuesta a la necesidad de aplicaciones distribuidas, proporcionando capacidades para el trabajo con componentes que residen en diferentes computadoras. DCOM utiliza como mecanismo de comunicación los llamados a procedimientos remotos (RPC), los cuales son transparentes para el cliente.

1.6.2 CORBA

CORBA, *Common Object Request Broker Architecture*, es una infraestructura abierta para el manejo de objetos distribuidos estandarizada por el *Object Management Group* (OMG), [Vinoski, 1997]. En CORBA, un objeto se considera como una instancia de una clase que encapsula operaciones, atributos y excepciones. CORBA permite que los objetos puedan comunicarse con otros sin necesidad de preocuparse por donde están localizados o por quien han sido diseñados. Para esto, se han considerado algunos elementos importantes como: un IDL y una API que provee al programador la interacción cliente-servidor entre componentes con la implementación especifica de un *Object Request Broker* (ORB), el cual es una especie de infraestructura en CORBA que permite que los objetos puedan comunicarse.

Un componente expone un conjunto de interfaces, implementadas utilizando un lenguaje de definición llamado OMG IDL, que son las mismas que definen las operaciones de los objetos. Actualmente existen estándares para el mapeo de una IDL a lenguajes de programación como C, C++, Java, Smalltalk, o Ada. Una vez que la interfaz se ha definido, se procede a generar dos archivos especiales, un archivo denominado stub para el cliente y, un archivo skeleton para el servidor, estos archivos son una especie de pegamento entre el cliente y el servidor con el ORB que le permiten acceder a sus interfaces.

El ORB, provee un mecanismo para la comunicación transparente entre clientes y objetos, actuando como un intermediario para realizar la comunicación ya sea local o remota entre componentes. El ORB simplifica la programación distribuida desligando del cliente detalles sobre la invocación de los métodos. Para ello, intercepta la solicitud, encuentra el objeto que corresponde a dicha solicitud, invoca al método correspondiente y regresa el resultado. Al delegarse al ORB todas estas responsabilidades, el cliente no tiene que preocuparse por detalles sobre la localización del objeto, sobre el

lenguaje de programación en el cual el objeto ha sido implementado, del sistema operativo, u otros aspectos propios del sistema.

Desde una perspectiva general, las principales ventajas de CORBA son [Vinoski, 1997]:

- *Heterogeneidad*. El uso de IDL permite definir las interfaces de manera independiente del lenguaje, y los objetos CORBA pueden distribuirse en plataformas muy diversas.

- *Modelo de objetos*. La interacción entre objetos está definida de modo que es indiferente el protocolo de red subyacente.

- *Integración de lo ya existente*. La especificación de CORBA no hace referencia a la implementación; es posible integrar protocolos y aplicaciones existentes, como DCE o COM.

- *Enfoque orientado a objetos*. CORBA (empezando por IDL) está diseñado según el paradigma de la orientación a objetos, lo que hace que las aplicaciones implementadas sobre CORBA se beneficien también de este enfoque.

1.6.3 JavaBeans

Actualmente uno de los modelos de desarrollo de software basado en componentes con mayor aceptación, es el propuesto por Sun Microsystems, conocido como JavaBeans. Java Beans es una API implementada para la construcción y uso de componentes escritos en Java, los cuales son comúnmente llamados Beans. Esta API es proporcionada por SUN como una herramienta visual que permite la carga, utilización, modificación, así como también interconexión de Beans, con el propósito de construir nuevos Beans, applets o aplicaciones completas.

Retomando la idea de [Vanhelsuwe, 1998], una forma útil para describir a un Bean es comparándolo con una especie de caja negra; una unidad de la cual

se conoce su funcionalidad pero no su implementación o estructura interna. Siendo un Bean un elemento de software sobre el cual solo se conoce su función, es preciso contar con algún mecanismo que permita la comunicación con él. Ese mecanismo de comunicación consiste en una interfaz, a partir de la cual, se puede acceder a los elementos principales de un Bean: sus métodos, propiedades y eventos.

En Java Beans, los métodos implementados en un Bean, especialmente los definidos con el prefijo set o get, son un medio para interactuar con él. Los métodos pueden entenderse como servicios con efectos específicos que el usuario puede invocar en algún momento. Las propiedades son conceptualmente equivalentes a lo que en el software tradicional se conocen como atributos, estos consisten en un conjunto de características que un usuario puede leer o modificar haciendo uso de los métodos de acceso. Los eventos son utilizados por el Bean como un medio de comunicación con otros Beans. Un evento se presenta como un cambio de estado que el componente puede notificar a su ambiente. El evento lleva información de quién lo genera, así mismo puede llevar datos y aún objetos que migran de una clase a otra.

Puesto que existen otros modelos de desarrollo basados en componentes, en JavaBeans se han establecido algunas capacidades con el propósito de distinguirlo de otros. La característica más importante, es por supuesto, que el componente este escrito en Java, además de soportar las siguientes capacidades:

- *Introspección,* mecanismo mediante el cual se pueden descubrir las propiedades, métodos y eventos que un Bean contiene.

- *Soporte a propiedades*, la posibilidad de conocer las características de los atributos y la capacidad de poder ser modificados en tiempo de diseño.

- *Soporte a eventos**,*** actividades como generación, escucha o respuesta a eventos con el propósito de comunicarse con otros Beans.

- *Persistencia,* capacidad para salvar y recuperar las especializaciones realizadas a un Bean.

El modelo propuesto por Sun Microsystems, permite interrelacionar los componentes construidos bajo sus propias consideraciones con componentes de otros lenguajes y modelos, destacando principalmente la compatibilidad con el estándar de CORBA. Aunque los Beans han sido pensados para ser simples procesos locales, recientemente se ha propuesto una alternativa para trabajar con componentes remotos diseñadas para correr en un servidor y ser invocados por clientes, esta propuesta es conocida como Server Beans o Enterprise Java Beans (EJB) [Roman et al., 2001].

El estándar EJB es una arquitectura de componentes para componentes desplegables del lado del servidor, escritos en Java. Es un acuerdo entre componentes y servidores de aplicaciones que permiten a cualquier componente, correr en cualquier servidor de aplicación. Los componentes EJB, cuentan con mecanismos que les permiten ofrecer nuevos y mejores servicios entre los que destacan los de seguridad, manejo de transacciones, concurrencia y persistencia. Físicamente, el estándar EJB es dos cosas en una [Roman et al., 2001]:

- *Una especificación*, que presenta las reglas de compromiso entre los componentes y los servidores de aplicación.

- *Un conjunto de interfaces Java*. Componentes y servidores de aplicación que deben hacerse para estas interfaces.

De lo expuesto anteriormente, se puede observar las grandes diferencias que existen entre los estándares Java Beans y Entrerprise Java Beans. Un Java Bean es un componente utilizado en Java que permite agrupar funcionalidades para formar parte de una aplicación, esto puede ser, un "Java Bean" agrupando información personal, datos sobre un pedido, requerimientos de ordenes, etc.; un Enterprise Java Bean también agrupa funcionalidades para una aplicación,

sin embargo, a diferencia de un Java Bean un Enterprise Java Bean es un componente de despliegue, lo que implica que existe un ambiente de ejecución, éste ambiente es precisamente un EJB Container, el cual forma parte de un servidor de aplicaciones java. Un Java Bean requiere ser integrado con otros componentes para que éste sea funcional, mientras un Enterprise Java Bean a través de un contenedor EJB puede ser activado (deployed).

1.7 VENTAJAS Y DESVENTAJAS DEL USO DE COMPONENTES

La llegada del software de componentes es uno de los más importantes desarrollos nuevos en la industria del software desde la introducción de lenguajes de alto nivel. El software de componentes combina las ventajas del software personalizado y del software en general. Esto hace que las soluciones sean las más evolucionadas, de mantenimiento rápido, que puedan ser extendidas a través del tiempo y que puedan ser modernizadas incrementalmente. Algunas de las ventajas de esta tecnología se exponen a continuación:

VENTAJA	EXPLICACIÓN
Facilita el trabajo distribuido	Las aplicaciones de software están divididas de acuerdo con la funcionalidad y dependencia, lo que hace más fácil identificar el personal requerido para el desarrollo de cada componente.
Facilita el mantenimiento	Los sistemas son formados por componentes con interfaces claras entre ellos. Esto hace a los problemas más tratables, que en sistemas construidos monolíticamente.
Desarrollo Paralelo	Una vez la dependencia haya sido definida, grupos de trabajo pueden trabajar simultáneamente en diferentes partes del sistema
Extensibilidad	Los componentes pueden ser reemplazados por otros, ofreciendo la misma interfaz, sin afectar el resto del sistema.

Tabla 1. Ventajas del Uso de Componentes

Algunos de los problemas inevitables en el CBDS son cómo tratar con aspectos no funcionales de comunicación, cooperación y coordinación, incorporados en una arquitectura/framework de componentes. La separación de requerimientos no funcionales, la cual da a un componente más independencia contextual, posiblemente permitiendo el reuso a través de un gran rango de contextos, debería ser clara. Otro problema importante es el problema de clase base de frágil sintaxis, que es la incompatibilidad causada por las versiones incorrectas de componentes. La actual producción agitada de nuevos frameworks CORBA, ActiveX/COM y Java es peligrosa porque su definición a menudo parece desarrollarse más rápido que la experiencia necesaria. Además, los problemas de estandarización y certificación complican aun más el difícil proceso de desarrollo. Algunas de las desventajas de esta tecnología se exponen a continuación:

DESVENTAJA	EXPLICACIÓN
Tiempo	Los grupos que dependan de componentes no terminados, tiene que esperar hasta que éstos sean desarrollados. Los grupos separados de desarrollo se rigen por la política de "no publicar mientras no se pruebe" debido a los costos de integración continua.
Dificultad de Integración	Desde que los componentes hayan sido desarrollados separadamente, éstos podrían fallar en la integración debido a problemas de cambio de interfaces, cambios de ambiente, etc.

Tabla 2. Desventajas del Uso de Componentes

1.8 COMPONENTES COMERCIALES (COMMERCIAL OFF-THE-SHELF (COTS))

El termino COTS, como sucede con muchos otros términos en el campo de la informática (como por ejemplo Internet), surge desde el Ministerio de Defensa de los Estados Unidos [Oberndorf, 1997]. Históricamente hablando, el término

COTS se remonta al primer lustro de los años 90, cuando en Junio de 1994 el Secretario de Defensa americano, William Perry, ordenó hacer el máximo uso posible de especificaciones y estándares comerciales en la adquisición de productos (hardware y software) para el Ministerio de Defensa. En Noviembre de 1994, el Vicesecretario de Defensa para la Adquisición y Tecnología, Paul Kaminski, ordenó utilizar estándares y especificaciones de sistemas abiertos como una norma extendida para la adquisición de sistemas electrónicos de defensa. A partir de entonces, los términos “comercial”, “sistemas abiertos”, “estándar” y “especificación” han estado muy ligados entre sí (aunque un término no implica los otros), estando muy presentes en estos últimos años en la Ingeniería del Software Basada en Componentes (ISBC).

Como sucede para el caso de los componentes software, en la literatura tampoco existe una definición concreta y comúnmente usada para el término COTS. Una definición híbrida del término “componente COTS” la podemos encontrar utilizando, por un lado, la definición de “componente” de Szyperski [Szyperski, 1998], y por otro lado la definición de elemento “COTS” del Software Engineering Institute (SEI), que dice lo siguiente: Un elemento COTS se refiere a un tipo particular de componente software, probado y validado, caracterizado por ser una entidad comercial, normalmente de grano grueso y que reside en repositorios software, y que es adquirido mediante compra o alquiler con licencia, para ser probado, validado e integrado por usuarios de sistemas, [Brown y Wallnau, 1998].

1.8.1 Limitaciones del desarrollo de software con componentes COTS

Sin embargo, el uso que se hace de la definición de componente COTS conlleva una serie de limitaciones. En primer lugar, aunque es cierto que desde 1994 se están utilizando prácticas para la utilización de componentes comerciales en procesos de desarrollo, la realidad es que muchas organizaciones encuentran que el uso de componentes COTS conlleva un alto riesgo y esfuerzo de desarrollo, para controlar su evolución y mantenimiento

dentro del sistema [Dean y Vigder, 1997]. Estos problemas se deben en cierta medida, a que las organizaciones utilizan procesos y técnicas tradicionales para el desarrollo basado en componentes, pero no para componentes comerciales.

Otro inconveniente, es que los fabricantes de componentes COTS tampoco documentan de forma adecuada sus productos para que puedan ser consultados por usuarios desarrolladores que necesitan conocer detalles de especificación del componente, como información acerca de sus interfaces, protocolos de comunicación, características de implantación (tipos de sistemas operativos y procesadores donde funciona, lenguaje utilizado, dependencias con otros programas, etc).

Por último, dado que no existen formas globalmente conocidas para documentar los componentes COTS, también son inexistentes los repositorios que almacenen especificaciones de componentes COTS, y por tanto, en mayor medida, no se puede pensar en procesos de mediación que permitan por un lado a los “fabricantes de componentes COTS” poder anunciar sus productos, y por otro a los “usuarios de componentes COTS” poder buscar los componentes COTS que necesitan.

CAPÍTULO II
PLATAFORMAS PARA APLICACIONES DISTRIBUIDAS

El disponer de componentes software no es suficiente para desarrollar aplicaciones, ya provengan estos de un mercado global o sean desarrollados a la medida para la aplicación. Un aspecto crítico al momento de construir sistemas complejos es el diseño de la estructura del sistema, y por ello el estudio de la Arquitectura Software se ha convertido en una disciplina de especial relevancia en la ingeniería del software [Shaw, 1996]. La reutilización de arquitecturas de software se presenta dentro un marco de trabajo (MT, o framework), que se puede definir como un diseño reutilizable de todo o parte de un sistema, representado por un conjunto de clases abstractas y la forma en la cual sus instancias interactúan. Un Marco de Trabajo Distribuido es un MT diseñado para integrar componentes y aplicaciones software en ambientes distribuidos, permitiendo la modularidad y reutilización en el desarrollo de nuevas aplicaciones [Fayad y Schmidt, 1997]. En la terminología de la programación orientada a componentes, este término es sinónimo de Plataforma de Componentes Distribuidos (PCD). Este tipo de marcos ofrecen un entorno de desarrollo y de ejecución de componentes software que permite aislar la mayor parte de las dificultades conceptuales y técnicas que conlleva la construcción de aplicaciones basadas en componentes. En este capítulo, se dará una descripción de las tres plataformas empresariales más importantes: J2EE, .NET y CORBA, junto con las ventajas y desventajas de la adopción de cada una de ellas.

2.1 LA EMPRESA HOY

Empresa significa una organización de negocios, y las aplicaciones empresariales son aquellas aplicaciones software que facilitan varias actividades en una empresa, como gestión de recursos y catálogos,

procesamiento de facturas, etc. Hoy en día, la construcción de estas aplicaciones está cambiando. Algunos de los factores que contribuyen a este cambio son [Allamaraju et al., 2001]:

- *Diversidad de las necesidades de información.* En una empresa, la información es creada y consumida por varios usuarios de diferentes formas, dependiendo de la necesidad específica.
- *Complejidad de los procesos de negocio*. La mayoría de los procesos de negocio empresariales involucran captura, procesamiento y compartimiento de información compleja, lo que implica técnicas y requerimientos arquitecturales complejos, para construir aplicaciones empresariales.
- *Diversidad de aplicaciones*, debido a la naturaleza compleja de los procesos de negocio empresariales, es común encontrar en una empresa, un gran número de aplicaciones construidas varias veces para satisfacer las necesidades de diferentes procesos de negocio.

Estos factores son muy comunes, y las empresas invierten grandes cantidades de dinero para construir y gestionar aplicaciones que afronten estos retos. Así, la adopción de nuevas tecnologías tiene que convertirse en un factor clave, para que las empresas exploten su habilidad de conocer el valor de la información.

2.2 TIPOS DE ARQUITECTURAS EMPRESARIALES

Antes de describir las arquitecturas existentes para el desarrollo de aplicaciones distribuidas, es necesario considerar los tipos de aplicaciones distribuidas contemporáneas (arquitecturas 2-capas, 3-capas, n-capas). A pesar de que estos tipos de arquitecturas son muy comunes en la empresa de hoy, es importante notar que estos tipos surgen debido a la llegada de las plataformas hardware más baratas para clientes y servidores, y tecnologías de red.

2.2.1 Arquitectura 2-capas

En una aplicación tradicional 2-capas, el procesamiento de carga es entregado al PC cliente, mientras el servidor simplemente actúa como un controlador de tráfico entre la aplicación y los datos. Los sistemas cliente-servidor típicos, están basados en esta arquitectura, por lo cual hay una clara separación de los datos y la lógica de presentación/negocio. A pesar de que esta arquitectura nos permite compartir datos a través de la empresa, tiene muchos inconvenientes, como:

- La lógica de negocio puede ser compleja y como esta se ejecuta en el cliente obliga a disponer de potente hardware para su ejecución.
- Hay un gran acoplamiento en los componentes. Un pequeño cambio en el cliente obliga a distribuir la aplicación a todos los puntos donde se ejecuta la aplicación.
- Cada cliente tiene que abrir una conexión con el servidor.
- La cantidad de datos que se envía a través de la red suele ser bastante grande ya que no se ha procesado en el servidor.
- El servidor representa un punto de fallo único. No existe el concepto de *clustering*[3].
- El servidor es accedido públicamente por lo que existen riesgos de seguridad.
- La extensibilidad es muy pobre ya que la lógica de negocio y el cliente se encuentran altamente acoplados.

2.2.2 Arquitectura 3-capas

Una aplicación 3-capas está dividida en tres capas separadas lógicamente, cada una con un conjunto de interfaces bien definidas. La primera capa, corresponde a la capa de presentación y básicamente consiste en una interfaz

de usuario gráfica. La segunda capa, corresponde a la capa de negocio, que consiste en la lógica de negocio o de presentación, y la tercera capa corresponde a la capa de datos, que contiene los datos necesarios para la aplicación.

2.2.3 Arquitectura n-capas

En esta arquitectura, la lógica de la aplicación está lógicamente dividida por funciones, no físicamente. Este tipo de sistemas pueden soportar un número de diferentes configuraciones. Una arquitectura n-capas se descompone en:

- *Una interfaz de usuario*, que maneja la interacción del usuario con la aplicación.
- *La lógica de presentación*, que define lo que la interfaz de usuario debe desplegar y cómo deben ser tratadas las solicitudes de usuario.
- *La lógica de negocio*, que modela las reglas de negocio de la aplicación, a menudo a través de la interacción con los datos de la aplicación.

Otras características generales de estos sistemas de n-capas son:

- Tienen una buena fiabilidad y disponibilidad ya que el servidor de aplicaciones ofrece sistemas de clustering que aseguran que el sistema esté disponible. No existe un único punto de fallo.
- La lógica de negocio está separada del resto de capas. Un cambio a la lógica de negocio no implica la modificación de todos los clientes. Asimismo la capa de presentación nos permite modificar la interfaz de la aplicación sin tener la necesidad de realizar cambios en los clientes.
- El rendimiento suele ser muy bueno. El servidor de aplicaciones ofrece automáticamente soporte de pool de objetos y conexiones, cachés, balanceo de carga, gestión de múltiples instancias, etc.

---

[3] La tecnología de *clustering,* es un servicio estándar de Windows NT Server, Enterprise Edition, y lleva las capacidades y rendimiento de los centros de datos a una cantidad mayor de instalaciones de clientes.

- La extensibilidad es excelente. Existe poco acoplamiento entre las diferentes capas.

- La seguridad se encuentra centralizada. Por lo tanto, la seguridad suele implicar gran cantidad de sistemas y tiene una arquitectura compleja.

2.2.4 Arquitectura empresarial

Una posible definición abstracta de arquitectura empresarial sería: El estudio de sistemas empresariales complejos desde el punto de vista de su estructura. Una plataforma de desarrollo empresarial ha de ofrecer una serie de servicios a los arquitectos y desarrolladores encaminados a facilitar el desarrollo de aplicaciones empresariales, al tiempo que ofrece la mayor cantidad posible de funcionalidades a los usuarios. Normalmente una plataforma de desarrollo empresarial tiene los siguientes requisitos:

- *Escalabilidad*. Para ofrecer una buena escalabilidad tanto horizontal como vertical de modo que si aumenta la carga del sistema se pueda añadir servidores o ampliar los existentes sin que sea necesario realizar modificaciones.

- *Mantenibilidad*. Para permitir añadir y modificar los componentes existentes sin que se modifique el comportamiento del sistema.

- *Disponibilidad*. Para tener el soporte de arquitecturas tolerantes a fallos, sistemas de redundancia, etc., que aseguren que el sistema estará siempre disponible.

- *Extensibilidad*. Para hacer posible añadir nuevos componentes y capacidades al sistema sin que se vean afectados el resto de componentes.

- *Manejabilidad*. los sistemas deben ser fácilmente manejables y configurables.

- *Seguridad*. Para tener buenos sistemas de seguridad tanto a nivel de autenticación, como de autorización y de transporte.

- *Rendimiento*. Para ofrecer automáticamente soporte de clustering, balanceo de carga, pools de objetos, pools de conexiones, cachés, y en general mecanismos que permitan aumentar el desempeño de manera transparente al usuario.

La arquitectura empresarial es básicamente una arquitectura n-capas (definida en el punto 2.2.3). En la actualidad las plataformas de desarrollo empresarial más importantes son CORBA, .NET, y J2EE.

2.3 PLATAFORMA COMMON OBJECT BROKER ARCHITECTURE (CORBA)

CORBA es un estándar publicado por el *Object Management Group* (OMG) para una red de objetos distribuida y heterogénea. En el año 2000, el consorcio OMG comenzó la publicación de la versión CORBA 3 que ofrece soluciones en tres áreas: integración en Internet, calidad de servicio, y una arquitectura de componentes para CORBA [Siegel, 2000; Tary y Bukhres, 2001]. CORBA aporta un conjunto de especificaciones que, al ser incorporadas a las aplicaciones distribuidas, ofrecen una forma de conseguir la interoperabilidad entre sistemas distribuidos de naturaleza heterogénea. Para ello CORBA se centra en principios fundamentales: *la separación entre interfaz e implementación, la independencia de localización* y *la independencia del fabricante, y la integración de sistemas a través de la interoperabilida*d.

En CORBA todos los componentes son objetos. Cada objeto se puede implementar con un lenguaje de programación distinto, y ejecutarse sobre cualquier plataforma hardware y sistema operativo. Para OMG, un objeto es una entidad que encapsula una funcionalidad a través de una interfaz. Un objeto ofrece servicios a través de sus operaciones y atributos que son visibles mediante su interfaz. Esta entidad permite a los clientes solicitar la realización de operaciones en un objeto con independencia de su localización. El elemento que permite la transparencia de localización y de acceso a los objetos es el *Object Request Broker* (ORB), que podría ser el equivalente en software al bus

que interconecta componentes. Las interfaces de los objetos se especifican en un lenguaje especialmente definido para este fin, IDL, que forma parte del estándar CORBA.

2.3.1 Terminología básica de CORBA

- *Objeto CORB*A: Entidad virtual capaz de ser localizada por un ORB y de recibir peticiones por parte de un cliente.
- *Objeto destino (target*): Objeto CORBA al que se dirige una petición por parte de un cliente.
- *Client*e: Entidad que realiza peticiones sobre un objeto CORBA.
- *Servido*r: Aplicación en la que existen uno o más objetos CORBA.
- *Petició*n: Invocación de una operación sobre un objeto CORBA por un cliente.
- *Referencia a objet*o: Etiqueta usada para identificar, localizar y direccionar un objeto CORBA de forma única.
- *Sirviente (servant*): Entidad de un lenguaje de programación que implementa uno o más objetos CORBA. Los sirvientes existen dentro del contexto de una aplicación servidor.

2.3.2 Arquitectura

Los componentes de la arquitectura CORBA (presentados en la siguiente ilustración) permiten que una aplicación cliente pueda solicitar la realización de operaciones en un objeto CORBA (objeto implementación en la terminología de CORBA) que reside en un servidor. Las implementaciones de los objetos que están en el lado servidor definen los métodos que implementan las interfaces IDL. Estos objetos pueden estar implementados en diversos lenguajes de programación.

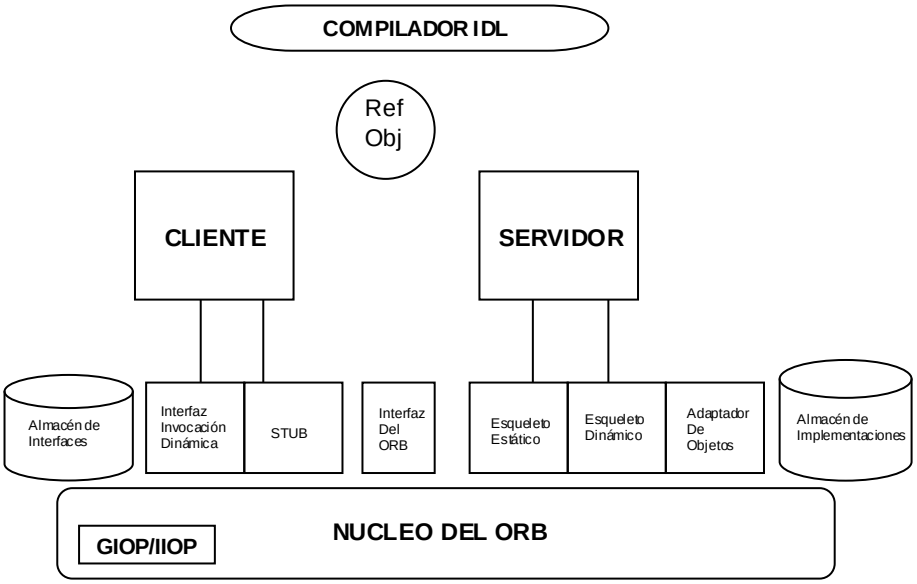


Ilustración 1. Arquitectura CORBA

Descripción de la arquitectura, Lado Cliente:

- *Stub del cliente:* Capa intermedia entre el cliente y el núcleo del ORB. Define cómo los clientes invocan los servicios que proporcionan los objetos servidores; se encargan de codificar la operación y sus parámetros, y enviarla de forma remota.
- *Interfaz de invocación dinámica DII (Dynamic Invocation Interface)*: Permite descubrir en tiempo de ejecución métodos para ser invocados.
- *Almacén de interfaces:* Permite obtener y modificar la descripción de todos los objetos que están registrados en él, métodos que soporta y los parámetros que requiere. No es más que una base de datos distribuida.
- *Interfaz del ORB:* Bibliotecas de servicios locales para realizar labores auxiliares en la aplicación.

Descripción de la arquitectura, Lado Servidor:

- *Esqueleto del servidor:* Proporciona los elementos necesarios para que los clientes invoquen los servicios exportados por el objeto. Tratan de hacer transparente el proceso de comunicación.

- *Esqueleto de interfaces dinámicas DSI (Dynamic Skeleton Interface*): Proporciona un mecanismo de enlazado en tiempo de ejecución para servidores que necesitan manejar peticiones realizadas dinámicamente.
- *Adaptador de objetos:* Elemento que se encarga de gestionar las peticiones que llegan al servidor y que asigna referencias e instancia objetos.
- *Almacén de implementaciones:* Almacén de información acerca de las clases que soporta el servidor, los objetos instanciados y sus *Object Identificators* (OIDs).

2.3.3 Servicios de CORBA

Los servicios de CORBA representan un conjunto de servicios que complementan el funcionamiento básico de los objetos que dan lugar a una aplicación. Son un complemento a la funcionalidad del ORB. El número de servicios se amplía continuamente para añadir nuevas facilidades a los sistemas desarrollados con CORBA. Esto no quiere decir que se encuentren implementados en los ORBs comerciales. Algunos de los principales servicios son:

- *Life cycl*e: define operaciones para crear, copiar, mover y eliminar objetos.
- *Event*s: permite registrarse para recibir eventos que pueden ser producidos por otros objetos.
- *Naming*: permite localizar objetos por un nombre.
- *Trade*r: permite localizar objetos por sus propiedades.
- *Persistenc*e: ofrece una interfaz para almacenar objetos.
- *Transaction*s: proporciona coordinación transaccional.
- *Concurrenc*y: proporciona un gestor de bloqueos.
- *Externalizatio*n: permite obtener y producir datos como *streams.*
- *Securit*y: da soporte a la autenticación, listas de control de acceso,

Confidencialidad, etc.

- *Time*: permite definir y gestionar eventos temporizados.
- *Properties*: permite asociar propiedades a un objeto.
- *Query*: ofrece una interfaz SQL para realizar operaciones en objetos.
- *Licensing*: ayuda a medir el uso de los objetos.
- *Relationship*: permite crear asociaciones dinámicas entre objetos.
- *Collection*: proporciona las interfaces para crear y manipular colecciones de objetos (listas, conjuntos, etc.).

2.3.4 Interoperabilidad en CORBA

El estándar de interoperabilidad de CORBA fue desarrollado para permitir que diferentes ORBs pudieran comunicarse. La interoperabilidad de CORBA ofrece una pasarela de infraestructura que hace que diferentes implementaciones de ORBs sean compatibles. Esta parte del estándar se asienta en la capa de transporte del modelo OSI. La arquitectura de interoperabilidad entre ORBs está basada en el protocolo *General Inter-ORB Protocol* (GIOP) que especifica la sintaxis de transferencia de un conjunto de mensajes estándar para la comunicación entre ORBs basada en un protocolo de transporte orientado a conexión. La especificación consiste en los siguientes elementos:

- *Common Data Representation* (CDR). Define cómo deben codificarse los datos para su transferencia entre clientes y servidores.
- Formato de los mensajes GIOP. Define ocho tipos de mensajes que pueden intercambiar los ORB de la parte del cliente y del servidor. Sólo dos de esos tipos de mensaje son necesarios para realizar la comunicación (Request y Reply), el resto son mensajes de control.
- Condiciones de la capa de transporte**.** Para poder transmitir mensajes GIOP.
    - Debe ser orientado a conexión.
    - Las conexiones son *full-duplex*.
    - Las conexiones son simétricas.

- Comunica si se produce una pérdida de conexión.

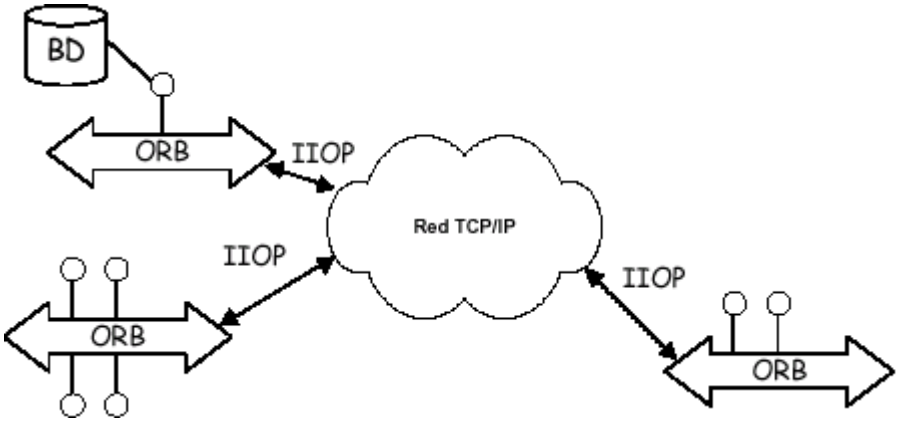


Ilustración 2. Interoperabilidad en CORBA

GIOP especifica la mayoría de los detalles de protocolo que son necesarios para que se puedan comunicar los servidores y los clientes. GIOP es un protocolo abstracto y, por tanto, es independiente de un tipo de transporte particular. El *Internet Inter.-ORB Protoco*l (IIOP) es una implementación concreta del protocolo GIOP sobre el protocolo de transporte TCP/IP. Por ello, IIOP necesita especificar cómo las Interoperable Object References (IOR) codifican la información del direccionamiento de TCP/IP (Ilustración anterior) para que el cliente pueda establecer comunicación con el servidor y enviarle una petición. IIOP es el principal protocolo de interoperabilidad utilizado por CORBA.

2.3.5 El modelo objeto de CORBA

El modelo objeto de CORBA es abstracto y no se implementa en ninguna tecnología particular. Un sistema objeto en CORBA es una colección de objetos que aísla mediante una interfaz bien definida a los que solicitan los servicios (clientes) de los que los ofrecen (servidores). El modelo objeto de CORBA es definido como dos modelos [OMG, 1995]:

- El modelo cliente, que describe la semántica de los objetos, esto es, los conceptos que son significativos para los clientes, tales como identidad

de objeto y creación de objetos, peticiones y operaciones, y tipos y signaturas.

- El modelo de implementación que describe los conceptos relacionados con la implementación de los objetos, incluyendo conceptos como métodos, motores de ejecución y activación de objetos.

El modelo objeto de CORBA es un caso particular de un modelo objeto convencional, donde el cliente envía un mensaje a un objeto [Tary y Bukhres, 2001]. Conceptualmente, el objeto interpreta el mensaje para decidir que servicio ejecutar. En el modelo convencional, un mensaje identifica a un objeto y cero o más parámetros actuales. Como en la mayoría de los modelos más convencionales, se requiere de un primer parámetro diferenciador, que es el que identifica la operación a ejecutar. La interpretación del mensaje por el objeto implica la selección de un método basado en la operación especificada. En el caso de CORBA, la selección del método puede ser llevada a cabo tanto por el objeto como por el ORB.

Algunas de las limitaciones de este modelo son:

- No existe una forma estándar para el despliegue de las implementaciones de objetos.
- Soporte estándar limitado para patrones de programación de servidores CORBA comunes.
- Extensión limitada de la funcionalidad de objetos. Los objetos pueden ser extendidos solamente a través de la herencia.
- La disponibilidad de los servicios de objetos CORBA no está definida por anticipado.
- Gestión del ciclo de vida de los objetos no estandarizada.

Para superar estas limitaciones, la OMG adoptó el CORBA Component Model (CCM).

2.3.6 El modelo de Componentes de CORBA (CCM)

Este modelo extiende el modelo objeto de CORBA para definir características y servicios que permitan a los desarrolladores de la aplicación implementar, gestionar, configurar y desplegar componentes que integran servicios de CORBA comúnmente usados, tales como transacción, seguridad, estado de persistencia, y servicios de notificación de eventos en un ambiente estándar. Adicionalmente, el CCM permite mayor reuso de software para servidores y provee flexibilidad para configuración de aplicaciones CORBA.

Los componentes CCM son bloques de construcción básicos en un sistema CCM. La mayor contribución de CCM se deriva de la estandarización del ciclo del desarrollo del componente utilizando CORBA como su infraestructura middleware.

CCM extiende el IDL de CORBA para soportar componentes. Los desarrolladores de componentes usan definiciones IDL para especificar las operaciones que un componente soporta, así como las interfaces de objetos son definidas en el modelo objeto de CORBA tradicional. Los componentes pueden ser desplegados en servidores de componentes que no tienen conocimiento avanzado de cómo configurar e instanciar esos componentes que han sido desplegados. Sin embrago, los componentes necesitan interfaces genéricas para ayudar a los servidores de componentes que los instalan y gestionan. Los componentes CCM pueden interactuar con entidades externas tales como servicios provistos por un ORB, otros componentes, o clientes a través de puertos, los cuales pueden ser enumerados utilizando mecanismos de introspección. Los puertos permiten mecanismos de configuración estándar para modificar las configuraciones de los componentes. Sin embargo, debe haber un mecanismo para expresar una configuración concreta para un componente, en particular, ellos necesitan designar qué componente(s) deben ser conectados y cómo los eventos fueron editados y recibidos por un componente relativo a cualquier otro. CCM, define una interfaz de configuración de un componentes estándar, llamada Components::StandardConfigurator,

para ayudar a los servidores de componentes a configurar los componentes. Los desarrolladores de componentes pueden extender esta interfaz de configuración para especificar cómo mejorar la flexibilidad de sus implementaciones de componentes. CCM define muchas interfaces para soportar la estructura y funcionalidad de los componentes. Muchas de estas interfaces pueden ser generadas automáticamente a través de herramientas proporcionadas por los proveedores de CCM. El CORBA Component Implementation Framework (CIF) está diseñado para proteger a los desarrolladores de componentes de tareas tediosas, automatizando actividades de implementación de componentes comunes.

CCM define un lenguaje declarativo, el Component Implementation Definition Language (CIDL) para describir las implementaciones y el estado de persistencia de componentes y de los domicilios de componentes. Las implementaciones de componentes dependen en gran medida del estándar CORBA Portable Object Adapter (POA) para enviar solicitudes de clientes entrantes a sus servidores correspondientes. La implementación del modelo de componentes CCM, utiliza la descripción de componentes para crear y configurar la jerarquía POA automáticamente y localizar los servicios comunes definidos por CCM. El modelo de programación de contenedor CCM define un conjunto de interfaces API´s que simplifican la tarea de desarrollar y/o configurar aplicaciones CORBA. Un contenedor encapsula una implementación de un componente y provee un ambiente en tiempo de ejecución para el componente que gestiona.

CCM es modelado estrechamente sobre la especificación EJB. A diferencia de EJB sin embargo, CCM utiliza el modelo objeto de CORBA como su arquitectura de interoperabilidad de objetos subyacente, y no está ligado a un lenguaje de programación particular. Desde que las dos tecnologías son similares, CCM también define el mapeo estándar entre los dos estándares. Sin embargo, un componente CCM puede aparecer como un bean, EJB para clientes EJB y un EJB puede aparecer como un componente CCM utilizando

técnicas puntales (bridging). EJB también soporta CORBA IIOP como su framework de comunicación.

CCM y CORBA están también relacionadas con la familia COM de Microsoft de las tecnologías de middleware. A deferencia de CORBA, COM de Microsoft fue diseñado para soportar un modelo de programación de componentes inicialmente y luego apareció DCOM añadiéndole la habilidad para distribuir objetos COM. La versión más reciente de las tecnologías de Microsoft, COM+, incluye generalmente servicios de negocios tales como el Microsoft Transaction Service (MTS). A diferencia de CORBA y EJB, COM+ está más limitado a las plataformas de Microsoft.

2.4 LA PLATAFORMA .NET

Microsoft .NET es un conjunto de tecnologías de software de Microsoft para conectar su mundo de información, gente, sistemas y dispositivos. Permite un nivel sin precedente de integración de software a través del uso de servicios Web XML: pequeños, discretos, bloques de aplicaciones construidos que se conectan con cada uno—así como a otras aplicaciones grandes—vía Internet. Microsoft .NET contribuirá a impulsar la transformación de Internet, gracias a la adopción de información en XML programable. XML es un estándar industrial compatible definido por la organización World Wide Web Consortium (W3C), que proporciona una forma de separar los datos reales de la vista de presentación de dichos datos. Es un medio de distribuir los datos entre una gran variedad de dispositivos digitales, permitiendo la colaboración e interacción de los sitios Web a través de los servicios Web basados en XML.

2.4.1 Arquitectura de .NET

La plataforma Microsoft .NET está integrada por cinco componentes:

- Servicios del sistema operativo Microsoft Windows®
- .NET Orchestration
- Familia de Microsoft .NET Enterprise Server

- Servicios de componentes .NET (.NET Building Block Services)
- NET Framework

2.4.2 Servicios del sistema operativo Windows

Con Microsoft .NET, los negocios pueden cambiar los procesos clave en servicios Web XML. Los servicios Web XML creados y alojados por una compañía o individuo pueden ser usados por millones de personas en varias combinaciones, para generar experiencias computacionales altamente personales e inteligentes. Este modelo distribuido de cómputo incrementará las demandas en las infraestructuras de servidores. Servidores seguros y escalables, que integren XML, proporcionarán la base para alojar e implementar software y servicios .NET.

Microsoft .NET Enterprise Servers, la familia de Windows 2000 Server y la próxima familia de Windows Server 2003, proporcionan la mejor solución para alojar e implementar servicios Web XML, con su seguridad integrada, su soporte de XML y su capacidad para escalar rápidamente y enfrentar demandas en aumento.

La familia de .NET Enterprise Server incluye:

- Microsoft Application Center 2000 para implementar y administrar aplicaciones Web altamente disponibles y escalables.
- Microsoft BizTalk Server 2000 para desarrollar procesos de negocios en XML a través de aplicaciones y organizaciones.
- Microsoft Commerce Server 2000 para desarrollar rápidamente soluciones escalables de comercio electrónico.
- Microsoft Content Management Server 2001 para administrar el contenido de sitios Web dinámicos de negocios electrónicos.
- Microsoft Exchange Server 2000 para habilitar la mensajería y colaboración a cualquier hora y en cualquier lugar.

- Microsoft Host Integration Server 2000 para pasar datos y aplicaciones a sistemas heredados de mainframe.
- Microsoft Internet Security and Acceleration Server 2000 para una conectividad a Internet segura y rápida.
- Microsoft Mobile Information 2001 Server para habilitar el soporte de aplicaciones por dispositivos móviles como teléfonos celulares.
- Microsoft SharePoint™ Portal Server 2001 para encontrar, compartir y publicar información del negocio.
- Microsoft SharePoint™ Portal Server 2001 para encontrar, compartir y publicar información del negocio.
- Microsoft SQL Server™ 2000 para almacenar, recuperar y analizar datos estructurados de XML.

Entre los servicios ofrecidos por Windows 2000 Server para los sistemas operativos de alto rendimiento se incluyen:

- *Seguridad*, incluido el cifrado y una infraestructura completa de claves públicas.
- *Subsistema XML completo* basado en los estándares de alto rendimiento
- *E/S* con capacidad de ancho de banda extremadamente alta entre el disco y la memoria, así como cualquier conexión TCP/IP.
- *Distribución contínua* a través de la ejecución simultánea de varias versiones de aplicaciones.
- *Compatibilidad* con subprocesos y fibras de alto rendimiento
- Indización, búsqueda y recuperación de contenido.
- *Compatibilidad* con los medios de secuencias de Windows para la colaboración de audio y vídeo.

- *Caché de memoria*
- *Programación de procesos*

La familia de productos Windows Server 2003 representa la próxima generación en la serie de sistemas operativos de Windows Server e integra un poderoso entorno de aplicación para desarrollar servicios Web XML innovadores y aplicaciones perfeccionadas que de manera dramática mejoran la eficiencia del proceso. Las versiones anteriores sirvieron de base para las mejoras realizadas en la administración, la fiabilidad, el rendimiento, la seguridad y los servicios Web XML. La versión de Windows Server 2003 se desarrolla a partir de estos fundamentos y amplía también la biblioteca de características de Windows Server 2003 con la incorporación de las tecnologías avanzadas de colaboración y Microsoft .NET Framework. Además, es el primero de los cuatro integrantes básicos de la familia de productos Windows Server 2003:

Windows Server 2003, Web Edition
Windows Server 2003, Standard Edition
Windows Server 2003, Enterprise Edition
Windows Server 2003, Datacenter Edition

Cada uno de estos productos puede personalizarse con características y funciones para cubrir las necesidades empresariales y de Tecnologías de la Información específicas de cada cliente.

2.4.3 .NET Orchestration

.NET Orchestration hace frente a uno de los retos empresariales más comunes: ser capaz de desarrollar rápidamente los procesos empresariales e integrarlos con los clientes y socios empresariales. Al mismo tiempo, se enfrenta a los retos técnicos que surgen de la expansión de dichos procesos en un gran número de sistemas de software que se ejecutan en varias plataformas de hardware y que, a su vez, se ejecutan en varias ubicaciones de proveedores de

servicios, socios y clientes. La solución de .NET Orchestration ante estos retos consiste en separar la definición de proceso de su implementación subyacente. Los expertos en procesos empresariales crean y administran la evolución de los mismos, mientras que los desarrolladores se centran en la implementación de los servicios compatibles subyacentes.

2.4.4 .NET Enterprise Servers

Los servidores .NET Enterprise Servers constituyen una amplia familia de servidores de Microsoft que permiten crear y administrar rápidamente un sistema empresarial integrado y habilitado para el Web. Diseñados para obtener un rendimiento escalable y eficaz, los servidores .NET Enterprise Servers se crean específicamente por razones de interoperabilidad utilizando los estándares de datos y Web Internet más actuales. Los siete servidores empresariales .NET principales son:

- SQL Server™ 2000
- Internet Security and Acceleration (ISA) Server 2000 (anteriormente Servidor proxy)
- Host Integration Server 2000 (anteriormente SNA Server)
- Exchange 2000 Server y Exchange 2000 Conferencing Server
- Commerce Server 2000
- BizTalk Server 2000
- Application Center 2000

2.4.5 Servicios de componentes .NET

Los servicios Web de componentes son los servicios Web comerciales que ofrece Microsoft y otros proveedores de servicios de Internet. Los desarrolladores podrán utilizar estos servicios en el desarrollo de sus propias aplicaciones y servicios Web. Los servicios Web de componentes .NET de Microsoft básicos incluirán:

- *Los servicios de identidad*, basados en el servicio *Passport*.

- *Los servicios de notificación y mensajería,* son en realidad un único servicio que integra la mensajería instantánea, el correo electrónico, fax, correo de voz y otras formas de notificación.
- *Los servicios de personalización,* controlan el modo de envío de las notificaciones y mensajes, así como el momento en que se produce dicho envío.
- *Los servicios de almacenamiento .NET,* son el equivalente digital de Internet a la caja de seguridad de un banco, en el que el usuario dispone de una llave que le permite controlar el acceso.
- *El servicio de calendario* es un servicio de integración para los calendarios laborales, sociales y particulares que, junto con datos en tiempo real, permitirán que otros servicios Web determinen la disponibilidad del usuario.
- *El servicio de directorio y búsqueda* permite la localización de servicios y usuarios.
- *Los servicios de entrega dinámica,* permiten a Microsoft, los Independent Software Vendors (ISV), socios de soluciones y otros desarrolladores, ofrecer niveles incrementales de funcionalidad y actualizaciones automáticas seguras según demanda, sin necesidad de instalación ni configuración por parte del usuario.

2.4.6 .NET Framework

.NET Framework es la base de las tecnologías de desarrollo .NET. Es un desarrollo multi-lenguaje y un ambiente de ejecución para la construcción, despliegue, y ejecución de servicios web XML, aplicaciones web estándar y aplicaciones cliente inteligente. Además permite la integración de Tecnologías de la información (IT) existentes con nuevas aplicaciones y servicios, y da a los desarrolladores y organizaciones la habilidad para resolver retos de despliegue y operación de aplicaciones en la escala de Internet. .NET Framework está formado por los siguientes componentes básicos:

- Common Type System (CTS). Este sistema soporta el concepto general de clases, interfaces, delegaciones, eventos, referencias y tipo de valores que todos los lenguajes comparten entre ellos, lo que proporciona interoperabilidad entre los lenguajes. CTS viene con un gran conjunto de tipos de datos predefinidos y cada tipo de datos es organizado en un tipo de jerarquía, lo que está estrechamente relacionado con un ambiente de programación orientado a objetos.

- CLS (Common Language Specification). Provee reglas básicas para lograr la integración de lenguajes. CLS es importante en dos formas, este determina el requerimiento mínimo para ser un lenguaje del Framework de .NET, y gestiona los objetos creados en un lenguaje para que sea interoperable con otros lenguajes. Así, la Framework Class Library (FCL) puede ser usada por cualquier lenguaje.

- MSIL (Microsoft Intermediate Language). Net no compila aplicaciones en código nativo, es decir, no compila aplicaciones en código específico para Intel o Mac. La compilación al igual que sucede con Java, se realiza en dos pasos sucesivos. Primero, el código escrito por el programador se compila en el MSIL, del mismo modo que las instrucciones en Java se convierten en bytecodes. Segundo, el CLR (CommonLanguage Runtime) compila en tiempo de ejecución las aplicaciones que están en MSIL y lo pasa a código nativo de la plataforma Intel o Mac.

- *Lenguaje común en tiempo de ejecuciòn (CLR)*. Permite que los lenguajes de programación alcancen paridad funcional y que participen activamente en el entorno de desarrollo de .NET de forma igualitaria. Así, los lenguajes C++®, Visual Basic, C#™ y JScript® ya no son los únicos grandes lenguajes de Microsoft .NET, ya que los socios de Microsoft han implementado otros 20 lenguajes ejecutables en .NET Framework.

- *Clases base .NET.* Se desarrollaron para ofrecer un único grupo común de clases en tiempo de ejecución para todos los lenguajes CLR de .NET y utilizar todas las características del CLR y la plataforma .NET.

- *Transporte de datos y XML.* Los componentes de datos y XML de .NET Framework hacen referencia a la forma en la que se transporta la información entre los componentes, los servicios Web y las aplicaciones, ya sean nuevos o existentes.

2.4.7 Servicios Web

La definición más simple de *servicio Web* es la siguiente: un componente de aplicación programable al que se puede tener acceso a través de protocolos Web estándar. Los servicios Web disponen de las siguientes características:

- Permiten solicitar la descripción de los servicios Web que ofrecen.
- Se definen en función de los formatos y la ordenación de los mensajes con los que son compatibles.
- Los clientes de los servicios Web envían y reciben mensajes utilizando XML.
- Para crear todas estas capacidades se utilizan protocolos de Internet abiertos.

Se agregan cuatro categorías de servicios Web a Internet:

- Servicios Web .NET públicos
- Servicios Web de componentes .NET comerciales
- Servicios Web predeterminados listos para ser utilizados
- Servicios Web desarrollados personalizados

Los servicios Web públicos que aparecerán en Internet se podrán integrar de forma fácil y generalizada tanto en las nuevas soluciones Web, como en las ya existentes. Algunos de estos servicios serán gratuitos, mientras que otros serán compatibles con varios modelos empresariales. A esta última categoría de servicios Web se la denomina *servicios Web de componentes comerciales* y

estarán disponibles desde proveedores de servicios de aplicaciones (ASP) y Microsoft. *Los servicios Web predeterminados* incluyen aquellos servicios desarrollados para uso particular por desarrolladores corporativos y socios de soluciones. Los desarrolladores podrán utilizar y personalizar varios servicios de componentes principales Microsoft .NET en sus propias aplicaciones y servicios, reduciendo el esfuerzo necesario para crear soluciones convincentes. Los servicios Web distribuidos creados con la plataforma Microsoft .NET estarán disponibles tanto en línea como fuera de línea. Un servicio se puede invocar en un equipo independiente no conectado a Internet proporcionado por un servidor local que se ejecute dentro de una empresa, o bien, se puede obtener acceso al mismo a través de Internet. El proceso denominado *agrupación* permite la cooperación y el intercambio de información entre varios servicios a través de la invocación de un proceso que permite a las organizaciones decidir si desean ejecutar su propia infraestructura o alojarla de forma externa sin poner en peligro su control o acceso a los servicios. Por ejemplo, un servicio de directorios corporativos puede agruparse con otro servicio Web de directorios que se ejecute en Internet.

2.4.8 Interfaz de Usuario Web

Desde el punto de vista de la interfaz de usuario, Microsoft .NET ofrecerá a los usuarios y desarrolladores las siguientes tecnologías:

- *Interfaz natural*. Un grupo de tecnologías que dará lugar a la próxima generación de interacciones entre seres humanos y equipos, incluidas la voz, la visión, la escritura manual y la entrada de lenguaje natural a través de un nuevo cuadro de inclusión de datos.
- *Lienzo universal*. Una arquitectura de información de componentes XML que integra la exploración, las comunicaciones y la creación de documentos en un único entorno unificado, lo que permitirá a los usuarios sintetizar e interaccionar con la información de una forma unificada. Su objetivo es transformar Internet de un entorno de sólo lectura a una plataforma de lectura y escritura, permitiendo a los

usuarios crear, buscar, modificar, anotar y analizar información de forma interactiva y desde cualquier lugar del mundo.

- *Agentes de información***.** Gestionan la identidad y los datos del usuario en Internet y proporcionan un mayor control sobre el modo de interacción de los sitios y servicios Web con el usuario. Estos, permitirán que el usuario sólo tenga que crear sus preferencias personales una sola vez para, después, permitir que un sitio o servicio Web las utilice o no.
- *Etiquetas inteligentes (SmartTags).* Extienden la tecnología IntelliSense al contenido Web y permiten tanto al equipo como a los dispositivos del usuario la manipulación inteligente de información desde Internet. Automatiza tareas repetitivas, simplifica la realización de tareas complejas, personaliza el software y facilita el acceso a la funcionalidad de los productos.

Para admitir la presentación transparente de la interfaz de usuario Web en cualquier dispositivo, la plataforma .NET introduce ASP+, la nueva generación de la tecnología de Páginas Active Server de Microsoft (ASP). Esta nueva generación facilita enormemente a los diseñadores y desarrolladores tanto el diseño como el desarrollo de la IU Web a la vez que se benefician de la experiencia existente en el desarrollo de las aplicaciones ASP. Para los administradores del Web, esta nueva tecnología facilita la distribución y administración de las aplicaciones y servicios Web, así como la obtención de una mayor escalabilidad, seguridad y confiabilidad.

2.5 LA PLATAFORMA J2EE

De acuerdo a la definición de Sun de J2EE: “J2EE define un estándar para el desarrollo de aplicaciones empresariales multicapa. J2EE simplifica las aplicaciones empresariales basándolas en componentes modulares y estandarizados, proporcionando un completo conjunto de servicios a estos componentes, y manejando muchas de las funciones de la aplicación de forma automática, sin necesidad de una programación compleja”. J2EE es una

especificación que proporciona un modelo de programación, que consiste en un conjunto de APIs y dirige la manera de construir aplicaciones J2EE. Por otra parte, J2EE especifica cómo debe ser la infraestructura de aplicación sobre la cual puedan correr las aplicaciones. Esta infraestructura de aplicación es proporcionada por los contenedores de las implementaciones de J2EE. La plataforma J2EE es esencialmente un ambiente de servidor de aplicaciones distribuido, un ambiente java que provee lo siguiente:

- Un conjunto de API´s de extensión Java para construir aplicaciones. Estas APIs definen un modelo de programación para aplicaciones J2EE.
- Una infraestructura en tiempo de ejecución para hospedar y gestionar aplicaciones. Este es el servidor en tiempo de ejecución en el cual las aplicaciones residen.

2.5.1 API´s de J2EE

Las aplicaciones distribuidas requieren acceso a un conjunto de servicios empresariales. Servicios típicos incluyen procesamiento de transacciones, acceso a bases de datos, mensajería, etc. Sin embargo, en lugar de tener acceso a estos servicios a través de interfaces propietarias o no estándares, los programas de aplicación en J2EE pueden acceder a estos servicios a través de un contenedor. La especificación de la plataforma J2EE 1.3 incluye un conjunto de extensiones estándar de Java que cada plataforma J2EE debe soportar. Entre estas se encuentran:

- *JDBC2.0*. Esta API añade más eficiencia a la obtención de conexiones, conjunto de conexiones, transacciones distribuidas, etc.
- *Enterprise JavaBeans (EJB) 2.0*. Esta API especifica un framework de componentes para aplicaciones distribuidas multi-capa. Define un estándar para definir componentes del lado del servidor, y especifica una gran infraestructura en tiempo de ejecución para hospedar componentes del lado del servidor.

- *Java Servlets 2.3*. Esta API provee abstracciones orientadas a objetos para construir aplicaciones Web dinámicas.
- *JavaServes Pages (JSP) 1.2*. Esta API enriquece las aplicaciones Web permitiendo el desarrollo de aplicaciones Web manejadas por formularios.
- *Java Message Service (JMS) 1.0*. Provee una API para colas de mensajes, publicación y suscripción de tipos de servicios middleware orientado a mensajes.
- *Java Transaction API (JTA) 1.0*. Permite la implementación de aplicaciones transaccionales distribuidas.
- *JavaMail 1.2*. Provee una plataforma independiente y un framework independiente de protocolo para construir aplicaciones e-mail basadas en Java.
- *JavaBeans Activation Framework (JAF) 1.0.* Esta API es requerida por la API JavaMail para determinar el contenido MIME de un mensaje y determinar las operaciones que pueden ser realizadas en las diferentes partes del mensaje.
- *Java Conector Architecture (JCA) 1.0*. Provee un mecanismo para integrar componentes de aplicaciones J2EE a sistemas de información legales.
- *Java API for XML Parsing (JAXP) 1.1*. Esta API provee abstracciones para parsers XML.
- *Java Authentication and Authorization Service (JAAS) 1.0*. Provee mecanismos de autorización y autenticación para aplicaciones J2EE.

La nueva especificación J2EE 1.4 hace mayor énfasis en los Servicios Web. Las API´s Java API for XML-based RPC (JAX-RPC) y SOAP with Attachments API for Java (SAAJ) proporcionan un soporte de interoperabilidad para servicios Web básicos. Entre las novedades están el API Java API for XML Registries (JAXR) soporta acceso a registros y repositorios, el API de gestión

de aplicaciones J2EE basada en Java Management Extensions (JMX), el API de despliegue de aplicaciones (J2EE Deployment API 1.1) y mejoras en la gestión de la seguridad a través de JACC (Java Authorization Contract for Containers). A todas estas novedades se unen las mejoras de las especificaciones de EJB, JSP, Servlets o JCA entre otras. Esta nueva versión de J2EE se sustenta sobre el JDK 1.4.2 en todas las plataformas para las que está disponible, es decir, Linux, Windows y Solaris. Algunos cambios en esta nueva especificación incluyen el soporte para el despliegue de librerías de clases independiente de cualquier aplicación, y la conversión de DTDs de los descriptores de despliegue a esquemas XML.

2.5.2 Arquitectura de J2EE

Una plataforma de J2EE comercial típica, debe incluir uno o más contenedores. Un contenedor es el software que en tiempo de ejecución proporciona a las aplicaciones J2EE acceso al conjunto de APIs de J2EE y que gestiona los componentes de aplicación desarrollados según las especificaciones de las APIs. Esta gestión de los componentes por parte de los contenedores es, junto con la independencia de plataforma, una de las ventajas más destacables de J2EE. La siguiente ilustración (adaptada de [Allamaraju et al., 2001]) muestra la arquitectura de J2EE en términos de sus contenedores y API´s:

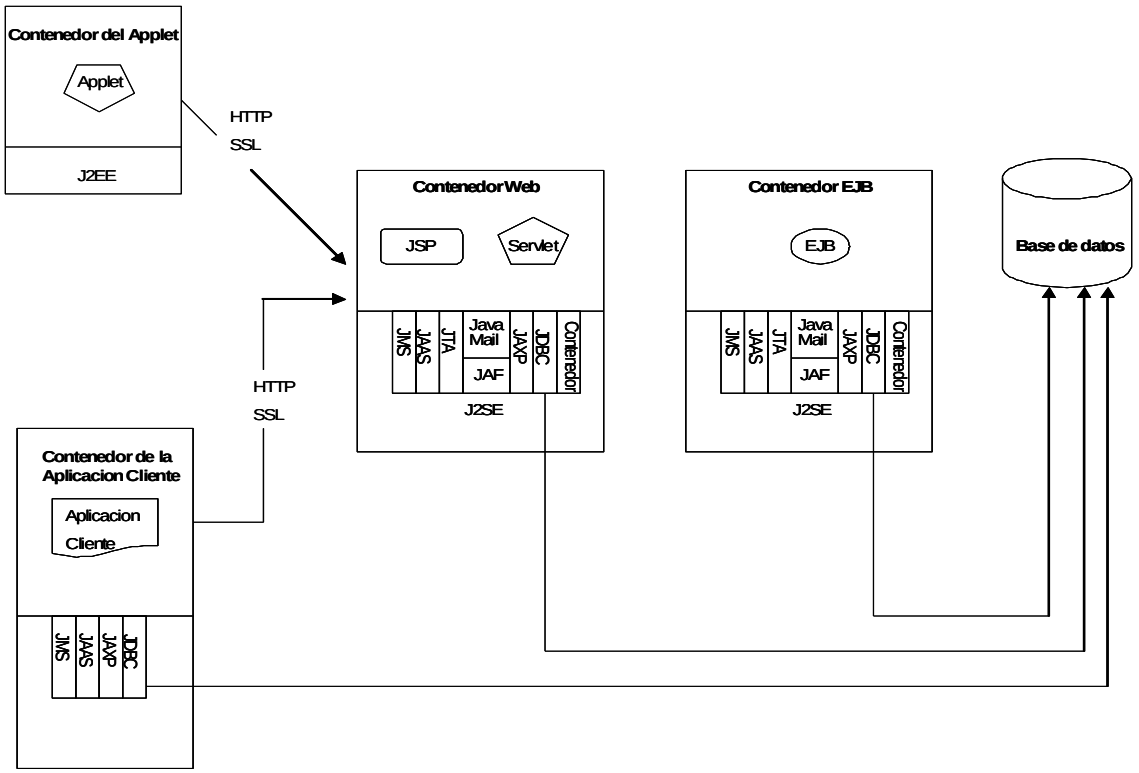


Ilustración 3. Arquitectura J2EE

2.6 CONTENEDORES J2EE

Los contenedores son la interfaz entre un componente y la funcionalidad específica de la plataforma de bajo nivel que soporta el componente. Antes de que un componente web, Enterprise Bean o cliente de la aplicación puedan ser ejecutados, debe ser ensamblado en su contenedor. Los procesos de ensamblaje involucran características de contenedores específicas para cada componente de la aplicación y para la aplicación en sí misma. Las características del contenedor adaptan el soporte subyacente proporcionado por el servidor J2EE, el cual incluye servicios tales como seguridad, gestión de transacciones, conectividad remota, etc. El contenedor también gestiona servicios no configurables tales como el ciclo de vida de Servlets y Enterprise Beans, pooling de recursos de conexión a bases de datos, persistencia de datos y acceso a las API´s de la plataforma J2EE.

Los contenedores que encontramos en la arquitectura J2EE son:

- Un contenedor web para hospedar Java Servlets y páginas JSP.
- Un contenedor para hospedar componentes Enterprise JavaBean
- Un contenedor de applet para hospedar Java applets
- Un contenedor de la aplicación cliente para hospedar aplicaciones Java estándar.

Cada uno de estos contenedores provee un ambiente en tiempo de ejecución para los componentes respectivos. En la ilustración anterior los bloques verticales representan las APIS de J2EE. En ésta arquitectura hay principalmente dos tipos de clientes:

- *Clientes Web*, que normalmente corren en navegadores Web. Estos clientes usan HTTP para comunicarse con el contenedor Web. Los componentes de la aplicación en los contenedores Web incluyen servlets Java y páginas JSP. Estos componentes implementan la funcionalidad requerida por los clientes Web. Los contenedores Web son responsables de aceptar las solicitudes de los clientes Web, y generar respuestas con la ayuda de los componentes de la aplicación.
- *Clientes EJB*. Son aplicaciones que acceden a componentes EJB en contenedores EJB. Hay tres tipos de clientes EJB: La primera categoría son los clientes de la aplicación, que son aplicaciones standalone que acceden a componentes EJB utilizando el protocolo RMI-IIOP. La segunda categoría, son los componentes en el contenedor Web, tales como Servlets Java y páginas JSP que también pueden acceder a los componentes EJB a través del protocolo RMI-IIOP. La categoría final son otros EJB corriendo en el contenedor EJB, que se comunican a través de llamadas al método de java estándar, por medio de una interfaz local.

2.7 COMPARACIÓN DE LAS PLATAFORMAS

CARACTERÍSTICA	CORBA	.NET	J2EE
Sistemas Operativos	Soporte de múltiples sistemas operativos	No soporta múltiples sistemas operativos. El mundo de .NET gira en torno al sistema operativo Windows y aunque se están intentando trasladar partes importantes de la plataforma, como la CLR o C#, a otros sistemas operativos, lo cierto es que estas partes forman una parte ínfima de la totalidad de la plataforma de Microsoft.	Soporte de múltiples sistemas operativos. Al ser una plataforma basada en el lenguaje Java, es posible desarrollar arquitecturas basadas en J2EE utilizando cualquier sistema operativo donde se pueda ejecutar una máquina virtual Java.
Organismos de Control	Controlada por un organismo serio, OMG	Un único dueño. La plataforma .NET está dominada única y exclusivamente por Microsoft. Esto supone un grave problema ya que es una única empresa la que puede añadir y quitar características según crea necesario. Además esto hace que la competencia sea nula y no se estimula la evolución de la plataforma.	Organismo de control. La plataforma J2EE está controlada por el Java Community Process (JCP), un organismo formado por más de 500 empresas, lo que garantiza la evolución de la misma.
Complejidad	CORBA es una plataforma de desarrollo muy compleja, aunque existen capas de abstracción que facilitan el desarrollo de aplicaciones, lo cierto es que desarrollar un simple programa de "Hola Mundo" no es una labor trivial.	Requiere desarrolladores poco experimentados: Bajo la plataforma de desarrollo de Microsoft es posible utilizar lenguajes como VB .NET que hacen muy sencilla la creación de aplicaciones empresariales. De este modo es posible tener un equipo de desarrolladores poco experimentados y que puedan crear fácilmente aplicaciones.	Aunque no es una plataforma tan compleja como CORBA, no existe un VB .NET en Java. La creación de aplicaciones bajo J2EE requiere normalmente desarrolladores más experimentados que los necesarios para desarrollar bajo .NET.

Lenguajes de Programación	Soporte de múltiples lenguajes	Soporte de múltiples sistemas lenguajes. Con .NET es posible desarrollar aplicaciones utilizando simultáneamente varios lenguajes de programación.	La plataforma J2EE depende exclusivamente del lenguaje Java. Sólo se puede utilizar este lenguaje para desarrollar aplicaciones lo que puede suponer un gran problema si nuestro equipo no dispone de los conocimientos suficientes o tiene otras preferencias.
Evolución	En el año 2000, el consorcio OMG comenzó la publicación de la versión CORBA 3 que ofrece soluciones en tres áreas: integración en Internet, calidad de servicio, y una arquitectura de componentes para CORBA.	Con sólo un año en el mercado, apenas ha salido algún proyecto importante desarrollado con esta tecnología. Su inmadurez hace que probablemente deba pasar algún tiempo hasta que sea realmente productiva.	Creada en el año 1997 como respuesta a la tecnología MTS de Microsoft, J2EE tiene ya seis años de vida y una gran cantidad de proyectos importantes a sus espaldas.

Ilustración 4. Matriz Comparativa de CORBA- .NET-J2EE

CAPÍTULO III
DESARROLLO DE APLICACIONES CON J2EE

Desde su introducción, hace más de dos años, La plataforma Java 2 Enterprise Edition (J2EE), ha establecido rápidamente un nuevo modelo para el desarrollo de aplicaciones distribuidas. Este se basa en componentes bien definidos que pueden automáticamente tomar ventaja de servicios de plataformas sofisticadas. Estos componentes pueden ser desarrollados de acuerdo a estándares, combinados en aplicaciones, desplegados en una variedad de productos servidores compatibles, y reusados para alcanzar un máximo de productividad. El éxito de la plataforma J2EE se debe en gran parte al éxito de este modelo. Diferentes implementaciones de la plataforma J2EE deben proporcionar características distinguibles que mejoren su funcionamiento en áreas particulares. Este capítulo describe las tecnologías y algunas consideraciones de diseño necesarias para el desarrollo de las aplicaciones J2EE.

3.1 APLICACIONES MULTICAPA DISTRIBUIDAS

La plataforma J2EE usa el modelo de aplicaciones multicapa distribuidas. La Lógica de la aplicación es dividida de acuerdo a la función de los componentes y éstos son instalados en diferentes máquinas dependiendo de la capa a la que el componente de la aplicación pertenece.

3.2 TECNOLOGIAS J2EE

Las tecnologías J2EE proveen los mecanismos necesarios para la construcción de grandes aplicaciones empresariales distribuidas. Esta gran colección de tecnologías pueden ser clasificadas de acuerdo a su uso en:

- *Tecnologías de componentes.* Estas tecnologías son utilizadas para contener la parte más importantes de la aplicación: La lógica del negocio. Hay tres tipos de componentes: páginas JSP, servlets y Enterprise Java Beans.
- *Tecnologías de servicios*. Estas tecnologías proveen los componentes de la aplicación que soportan servicios para una funcionalidad eficiente.
- *Tecnologías de comunicaciones.* Estas tecnologías proveen los mecanismos para la comunicación entre las diferentes partes de la aplicación, ya sean locales o remotas.

3.2.1 TECNOLOGÍAS DE COMPONENTES

En cualquier aplicación, el elemento más importante es el modelado de la lógica de negocio a través del uso de componentes. Estos componentes deben ser dependientes de su contenedor para muchos servicios tales como gestión del ciclo de vida, multihilo (threading), seguridad. Esto nos permite concentrarnos en cumplir con los requisitos de la funcionalidad de negocio sin centrarse en la semántica de bajo nivel.

La plataforma J2EE provee tres tecnologías para el desarrollo de componentes:

Componentes web. Estos pueden ser categorizados como cualquier componente que responde a una petición HTTP. Entre estos se encuentran:

- *Servlets:* son programas del lado del servidor que permiten que lógica de la aplicación sea embebida en procesos HTTP de solicitud-respuesta. Los servlets proveen un medio para extender la funcionalidad de los servidores web para generar contenido dinámico en HTML, XML, u otros lenguajes web.
- *JavaServer Pages (JSP)*. Proveen una forma de embeber componentes que trabajan juntos para generar la página que es eventualmente enviada al cliente. Una página JSP puede contener HTML, código java y componentes JavaBeans. Son de hecho, una extensión del modelo de programación de servlets. Cuando un usuario solicita una página JSP, el contenedor web compila la página JSP en un servlet, luego invoca el servlet y retorna el contenido resultante al navegador web. Una vez el servlet ha sido compilado desde la página JSP, el contenedor web puede simplemente retornar el servlet sin tener que volverlo a compilar. De esta forma, las páginas JSP proveen un mecanismo poderoso y dinámico para ensamblado de páginas que se benefician de muchas ventajas de la plataforma Java. Mientras los servlets son puro código java, las páginas JSP son documentos basados en texto hasta que el contenedor web los compila en los servlets correspondientes, lo que permite al programador concentrarse en la presentación de la aplicación.

Componentes Enterprise JavaBean. Los Enterprise Java Beans (EJB) son los componentes del modelo de desarrollo de aplicaciones empresariales de Sun. Un Enterprise Java Beans es el elemento de lógica o de persistencia, que acompañado de los componentes para armar la presentación, Servlets y Java Server Pages, forman lo que son los componentes java programables del modelo J2EE. Un EJB tiene al menos una clase y dos interfaces, la implementación, la interfaces Home y la de los servicios del bean. La interfaz de servicios puede ser de dos tipos desde EJB 2.0: remota o local. La diferencia es que la remota utiliza RMI/IIOP para hacer llamadas a los métodos, y la local no. Obviamente la local necesita que los objetos estén en la misma máquina virtual. La implementación soporta los métodos de negocio. La interfaz

home por un lado es una fábrica de objetos que permiten acceder al componente y por ello aquí se definen los métodos de creación, búsqueda y destrucción del EJB. Estos objetos de negocio, toman tres formas básicas:

- *Beans de sesión.* Son utilizados para manejar las interacciones de entidades y otros beans de sesión, accesos a recursos, y generalmente realizan tareas en beneficio del cliente. Encontramos dos tipos: *Los beans de sesión con estado,* son objetos utilizados para representar una interacción del cliente con el sistema. Estos ejecutan las solicitudes de los clientes en la aplicación, accesando a una base de datos etc, y cuando las operaciones de los clientes son completadas, el bean es destruido, es decir, el bean existe mientras exista la sesión del cliente. *Los beans de sesión sin estado*, como todos los beans de sesión, no son objetos de negocio persistentes. No representan datos en la base de datos. En su lugar, representan procesos de negocio o tareas que son realizadas en beneficio del cliente que los usa. Los métodos del negocio en un bean de sesión sin estado actúan como los procedimientos tradicionales en un monitor de procesamiento de transacciones legales. Cada llamada a un método de negocio es independiente de las llamadas anteriores.
- *Beans de entidad.* Son utilizados para representar datos en una base de datos. Proporcionan una interfaz orientada a objetos a los datos que normalmente serían accedidos mediante el JDBC u otro API. Más que eso, los beans de entidad proporcionan un modelo de componentes que permite a los desarrolladores de beans enfocar su atención en la lógica de negocio del bean, mientras el contenedor tiene cuidado de manejar la persistencia, las transacciones, y el control de accesos. Hay dos tipos de beans de entidad: Persistencia Manejada por el Contenedor (CMP) y Persistencia Manejada por el Bean (BMP). Con CMP, el contenedor maneja la persistencia del bean de entidad. Las herramientas de los fabricantes se usan para mapear los campos de entidad a la base de datos y no se escribe ningún código de acceso a las bases de datos en la clase del bean.

Con BMP, el bean de entidad contiene código de acceso a la base de datos (normalmente JDBC) y es responsable de leer y escribir su propio estado en la base de datos.

- *Beans de manejo de mensajes*. El propósito de estos beans es procesar mensajes recibidos vía Java Message Service(JMS). Cuando un cliente o una aplicación envía un mensaje vía JMS, el contenedor invoca el bean de manejo de mensajes apropiado para procesar el mensaje.

3.2.2 TECNOLOGIAS DE SERVICIOS

Algunos de los servicios de J2EE para los componentes de la aplicación son gestionados por los mismos contenedores, permitiendo de esta manera a los desarrolladores concentrarse en la lógica del negocio. Sin embargo, algunas veces los desarrolladores ven necesario invocar algunos de estos servicios utilizando las diferentes tecnologías de servicios, como:

- *Java Database Connectivity (JDBC)*. Esta API (un estándar para conectividad de bases de datos) permite al desarrollador conectarse a bases de datos relacionales. Además permite consultas transaccionales, recuperación y manipulación de datos desde una conexión JDBC. J2EE añade una extensión al núcleo de la API JDBC para dar características avanzadas tales como pooling de conexión[4] y transacciones distribuidas.

- *Java transaction API (JTA)*. Esta API es un medio para trabajar con transacciones y especialmente con transacciones distribuídas, independiente de la implementación del gestor de transacción, el Java Transaction Service (JTS).

- *Java Naming and Directory Interface (JNDI).* EL papel de esta API dentro de la plataforma J2EE es proporcionar el medio para hacer operaciones estándar sobre un recurso de servicio de directorio tales como LDAP, Servicios de directorio Novell, o servicios de directorio

[4] Pooling de conexión= conjunto de conexiones

Netscape. Una aplicación J2EE utiliza JNDI para buscar interfaces para crear, entre otras cosas, EJB´s y conexiones JDBC.

- *Java Message Service (JMS).* Proporciona la funcionalidad para enviar y recibir mensajes a través del uso del middleware orientado a mensajes (MOM).

- *Java Conector Architecture (JCA).* Es un estándar a través del cual las aplicaciones J2EE pueden acceder a una variedad de aplicaciones legales, especialmente sistemas de planeación de recursos empresariales tales como SAP R/3 y PeopleSoft. Esta especificación define una arquitectura simple en la cual los fabricantes de servidores de aplicaciones J2EE y de sistemas legales colaboran para producir componentes plug and play que permiten el acceso al sistema legal sin tener que saber nada específico de cómo trabajar con éste.
- *Java Autentication and Autorization Service (JAAS).* Esta API provee un mecanismo para conceder permisos basados en quién ejecuta el código.

3.2.3 TECNOLOGIAS DE COMUNICACIÓN

Son las tecnologías que proveen los mecanismos para que varios componentes y servicios de una aplicación J2EE se comuniquen unos con otros. Entre estas tecnologías tenemos:

Protocolos de Internet. Permiten la comunicación entre solicitudes de clientes y respuestas de servidores. Se destacan:

- *Hypertext Transfer Protocol (HTTP):* es un protocolo genérico, del nivel de aplicación, que tiene muchos usos más allá de las simples capacidades de hipertexto. Este trabaja en la base solicitud/respuesta. Un cliente envía una solicitud para el servidor en la forma de un método de solicitud, URI (Unifrom Resource Identifier), y versión del protocolo, seguido por un MIME, información del cliente, y posible contenido del cuerpo sobre una conexión con un servidor. El servidor responde con

una línea de estado seguida por un mensaje MIME, entidad de meta-información y posible contenido de la entidad cuerpo.

- *Transmisión Control Protocol over Internet Protocol (TCP/IP):* actualmente son dos protocolos separados que están combinados en una entidad simple. IP es el protocolo que asegura que los datos sean recibidos por ambos puntos finales en una comunicación sobre Internet. TCP es el protocolo que sigue las huellas de los paquetes y se asegura que éstos sean ensamblados en el mismo orden en que fueron enviados y que estén libres de errores. TCP e IP trabajan juntos para mover datos alrededor de la Internet.
- *Secure sockets Layer (SSL):* Este protocolo utiliza criptografía para encriptar el flujo de información entre el cliente y el servidor.

Protocolos de objetos remotos. Son mecanismos que permiten la utilización de componentes remotos. Entre estos tenemos:

- *Remote Method Invocation (RMI).* Es uno de los mecanismos principales en las aplicaciones de objetos distribuidos que permite el uso de Interfaces para definir objetos remotos. Luego podemos llamar métodos de estos objetos remotos como si fueran locales.
- *RMI-Inter ORB Protocol (IIOP).* Es una extensión de RMI sobre el protocolo IIOP, el cual permite definir una interfaz remota de cualquier objeto remoto para que pueda ser implementada en cualquier lenguaje que soporte mapeo OMG y ORB.
- *Java Interface Definition Language (IDL).* A través del uso de este, un cliente Java puede invocar llamadas a métodos sobre objetos CORBA. Estos objetos CORBA no necesitan estar escritos en Java, pero deben implementar una IDL.

3.3 EJB COMO UNA SOLUCION EMPRESARIAL

El estándar Enterprise Java Bean (EJB) es una arquitectura de componentes desplegables del lado del servidor en Java. Es un acuerdo entre componentes

y servidores de la aplicación que permite a cualquier componente correr en cualquier servidor de aplicación. Los componentes EJB son desplegables y pueden ser importados y cargados en un servidor de aplicación, que almacena estos componentes. Físicamente, un EJB es dos cosas en una:

- Una especificación. Esta especificación define las reglas del acuerdo entre los componentes y los servidores de aplicación.
- Un conjunto de interfaces Java. Los componentes y servidores de aplicación deben adaptarse a estas interfaces. Desde que todos los componentes sean escritos para las mismas interfaces ellos significan lo mismo para el servidor de aplicación.

EJB es específicamente utilizado para resolver problemas empresariales, y debe realizar cualquiera de las siguientes tareas:

- Hacer la lógica del negocio. Ejemplos de esto incluye: computarizar los impuestos de una tarjeta de compra, asegurarse de que el administrador tiene autorización para aprobar la orden de compra, etc.
- Acceder a una base de datos: Por ejemplo, transferencia de dinero entre dos cuentas bancarias, llamadas a procedimientos almacenados para recuperar un tiquete en un sistema de soporte de ventas, etc.
- Acceder a otros sistemas. Ejemplos incluyen, llamadas a sistemas legales de alto desempeño escritos en COBOL que computariza el factor de riesgo para una nueva cuenta de seguro, llamadas a Enterprise beans para permitir la integración de aplicaciones a través de JCA.

Para conseguir un despliegue y ejecución de un EJB exitosos, es necesario más que un solo servidor de aplicación y componentes. De hecho, EJB promueve la colaboración de diferentes grupos. Cada grupo es un experto en su propio campo y es responsable de una parte clave para el despliegue exitoso del EJB. Debido a esto, el tiempo total requerido para construir una clase empresarial de despliegue es muy reducido.

3.3.1 EL PROVEEDOR DEL BEAN

Este suministra los componentes empresariales, o enterprise beans. Los enterprise beans no son aplicaciones completas; son componentes desplegables que pueden ser ensamblados en soluciones completas.

3.3.2 EL ENSAMBLADOR DE LA APLICACION

Es el arquitecto de la aplicación completa. Este grupo es el responsable del entendimiento de cómo varios componentes se acomodan juntos y escribe la aplicación que combina los componentes.

El ensamblador de la aplicación podría hacer una o todas de las siguientes tareas:

- Desde el conocimiento del problema empresarial, decidir que combinación de componentes existentes y nuevos enterprise beans son necesarios para proporcionar una solución efectiva.
- Proporcionar una interfaz de usuario (Swing, servlet/JSP, aplicación/applet o Web Service wrapper).
- Escribir nuevos beans empresariales para resolver algunos problemas específicos para su problema empresarial.
- Escribir el código de integración que mapea los datos entre los componentes proporcionados por diferentes proveedores de beans. Luego, los componentes trabajarán juntos para resolver un problema empresarial, especialmente si diferentes vendedores escriben los componentes.

3.3.3 EL DEPLOYER DEL BEAN

Son las personas conscientes de los requerimientos operacionales específicos, entienden cómo desplegar los beans en servidores y cómo adaptarlos a un ambiente específico.

Algunas de sus tareas incluyen:

- Asegurar el despliegue con un Firewall y otras medidas de protección

- Integración con un servidor LDAP para listas de seguridad, tales como Lotus Notes o Microsoft Active Directory
- Escoger el hardware que proporciona le nivel de rendimiento requerido
- Proporcionar redundancia de Hardware y otros recursos para confiabilidad y tolerancia a fallos

3.3.4 EL ADMINISTRADOR DEL SISTEMA

Es el que gestiona la supervisión de la estabilidad de la solución operacional. Se encarga del mantenimiento y monitoreo del sistema desplegado y debe hacer uso del monitoreo en tiempo de ejecución y de las herramientas de gestión que proporciona el servidor EJB.

3.3.5 EL PROVEEDOR DEL SERVIDOR Y DEL CONTENEDOR

El proveedor del contenedor suministra el ambiente en tiempo de ejecución en el cual el bean vive. El contenedor proporciona servicios de middleware para los beans y los gestiona. Ejemplos de proveedores de contenedores son BEA´s WebLogic, Servidor de aplicación iplanet de iplanet, WebSphere de IBM, Oracle 9i de Oracle, Jrun de Macromedia, PowerTier de Persistence, el servidor de aplicación JBoss libre distribución, etc. El proveedor del servidor es el mismo que el proveedor del contenedor.

3.3.6 EL FABRICANTE DE HERRAMIENTAS DE DESARROLLO

Para facilitar los procesos de desarrollo de componentes, debe existir una forma estandarizada para construir, gestionar y mantener componentes. En el ambiente de los EJB hay muchos Ambientes de Desarrollo Integrales (IDEs) que ayudan a la construcción rápida y depuración de componentes. Ejemplos son, Visual Café de Webgain, VisualAge para Java o JBuilder de Borland.

3.4 DESARROLLO DE APLICACIONES J2EE

La especificación J2EE plantea en los procesos de desarrollo y despliegue de las aplicaciones:

- *Desarrollo de los componentes de la aplicación*. Durante este paso se modelan las reglas de negocio en la forma de componentes de la aplicación.
- *Composición de los componentes de la aplicación en módulos*. Durante este paso, los componentes de la aplicación son empaquetados en módulos. Esta fase también incluye la provisión de los descriptores de despliegue para cada módulo.

Un módulo es usado para empaquetar uno o más componentes de la aplicación del mismo tipo. Aparte de los componentes de la aplicación, cada módulo también incluye un descriptor de despliegue que describe la estructura del módulo. Hay tres tipos de módulos en J2EE:

- *Módulos web*. Un módulo web es un unidad desplegable formada por Servlets, páginas JSP, librerías JSP, archivos de librería JAR, documentos HTML/XML, y otros recursos públicos tales como imágenes, archivos de la clase applet, etc. Un módulo web es empaquetado en un archivo Web, llamado archivo WAR. El descriptor de despliegue está contenido en un archivo web.xml.
- *Módulos EJB*. Un módulo EJB es una unidad desplegable consistente de EJB´s, librerías de archivos JAR asociados, con un descriptor de despliegue (ejb-jar.xml) en el directorio META-INF de el archivo JAR.
- *Módulos Java*. Un módulo java es un grupo de clases cliente java empaquetadas en archivos JAR. El descriptor de despliegue para un módulo java es un archivo application-client.xml.
- *Composición de la aplicación a partir de los módulos*. Este paso integra múltiples módulos en aplicaciones J2EE. Esto requiere ensamblado de uno o más módulos en una aplicación J2EE, soportándola con el descriptor de los archivos.
- *Despliegue de la aplicación*. En este paso la aplicación empaquetada es desplegada e instalada en el servidor de la aplicación J2EE.

3.5 CONSIDERACIONES DE DISEÑO PARA APLICACIONES J2EE

La llegada del e-commerce e Internet ha cambiado la forma en que la mayoría de las aplicaciones empresariales son definidas y desarrolladas. Los requerimientos son más exigentes que antes y los tiempos de entrega más cortos. Para diseñar y desarrollar aplicaciones sobre estas condiciones se necesita:

- Un framework estandarizado, sobre el cual las aplicaciones pueden ser construidas y desplegadas. Este debe proveer niveles apropiados de funcionalidad y automatizar la creación de un estándar para conectar la aplicación al framework.
- Un conjunto de guías prácticas para usar el framework. Los desarrolladores de software no tienen una infinita cantidad de tiempo para dedicarlo al entendimiento de la filosofía del diseño o aprender la forma más eficiente de utilizar las API´s. Lo que ellos necesitan es guías para ayudarlos a escribir buenas aplicaciones usando el framework.

En términos Java, el framework para el desarrollo de aplicaciones distribuidas es J2EE. Las características y funcionalidad de la plataforma permiten la creación de aplicaciones basadas en componentes escalables, distribuidas y flexibles.

El uso del término arquitectura en el mundo del desarrollo software es sujeto de algunos debates. En términos software, este es usado de forma intercambiable con el término diseño. La arquitectura de una aplicación puede definir:

- Los componentes que hacen las tareas de negocio de la aplicación
- El tipo de interacción de estos componentes
- Los servicios usados por estos componentes
- Otras características o capacidades que manejan los requerimientos no funcionales de la aplicación

Adicionalmente, la arquitectura de una aplicación provee un framework en el cual pueden ser tomadas decisiones de diseño detalladas. Una decisión de diseño no es buena o mala, ésta depende del contexto y de los factores internos y externos que influyen en el diseño. Esto es válido para el modelamiento de la solución propuesta y el mapeo del modelo en la plataforma subyacente.

3.5.1 ELABORAR LOS REQUERIMIENTOS

Una vez los requerimientos funcionales han sido definidos, puede ser construido un modelo para que encapsule los procesos de negocios y las principales entidades de negocio. UML, puede ser utilizado para capturar los elementos principales del sistema que son descritos por los requerimientos funcionales.

La fase inicial del análisis y diseño involucran la creación de un modelo que mapea el problema a ser resuelto. Lo primero para la creación de este modelo son los casos de uso que describen las funciones del sistema deseado. Todas las soluciones reflejan el contexto del diseño y uno de los principales aspectos a tener en cuenta es que el modelo debe ser muy simple para que pueda ser bien entendido.

3.5.2 ELABORAR EL CONTEXTO

Una vez el modelo del dominio del problema ha sido construido, un modelo del dominio de la solución debe ser deducido de éste. Este modelo será sujeto a las fuerzas del mundo real del desarrollo de una aplicación tales como las capacidades de las herramientas y ambientes aplicados, la topología de la infraestructura en la cual la aplicación es desplegada, y las herramientas disponibles para el diseño. La solución que se deduce desde un modelo básico diferirá dependiendo de el “terreno” en el cual será **desplegado. Un** patrón que es prevalente en este ambiente debe sugerir una solución coherente a un problema particular.

3.5.3 ESCOGER Y DEFINIR UNA ARQUITECTURA

Diseñar sistemas distribuidos es difícil. Es importante tener un buen entendimiento de que va a hacer el sistema y que tipo de compromisos pueden ser hechos. El tipo de middleware distribuido caracterizado por J2EE ayuda a los diseñadores y desarrolladores proporcionando mucha de la infraestructura requerida para el desarrollo de sistemas distribuidos. La necesidad de componentes distribuidos es sólo una de las típicas causas que evoluciona el modelo de dominio idealizado en un diseño concreto que puede ser implementado en un conjunto particular de máquinas utilizando un lenguaje particular y un ambiente en tiempo de ejecución. Una vez la arquitectura es conocida, una serie de decisiones de diseño pueden ser tomadas para que mapeen exitosamente la arquitectura en sistemas subyacentes, plataformas y frameworks.

3.5.4 APLICAR PATRONES

Un patrón es una solución comprobada para un problema en un contexto dado. En términos software, los patrones son esencialmente el resultado de la experiencia ganada por practicantes de lo que trabaja bien cuando especifican, diseñan e implementan software. Los patrones de J2EE, apunta a sistemas empresariales n-capas y son categorizados basados en la capa en la cual ellos residen, particularmente Presentación, de Negocios e integración. Todos los contextos de una aplicación tienen algunos grados de exclusividad, por consiguiente, las implementaciones de los patrones deben ser adaptados para satisfacer nuestros requerimientos específicos.

CAPÍTULO IV
ANÁLISIS Y DISEÑO DE UNA APLICACIÓN SOBRE J2EE

J2EE ofrece una arquitectura, que establece un contexto tecnológico para el diseño y desarrollo de aplicaciones distribuidas en el marco del uso de componentes como estrategia de construcción. Este esquema responde a los retos actuales de las compañías comerciales en cuanto a crecimiento escalable, reutilización de lógica del negocio y distribución del procesamiento (procesamiento cooperativo). Sin embargo, las metodologías de desarrollo orientadas a objetos existentes abordan la problemática de la fabricación de software de manera general, sin incluir dentro de sus artefactos alguna aproximación hacia una arquitectura de base como J2EE. Por tanto existe un vacío cuando se trata de llevar a cabo un proceso de desarrollo para construir aplicaciones específicas para J2EE. Esto ha hecho que la construcción de software para estos ambientes se realice con una gran incertidumbre, lo cual ha originado el fracaso de los proyectos en muchas ocasiones. Estos fracasos no sólo han dependido de la carencia de un proceso de desarrollo adaptado a las características de la arquitectura, sino también ha influido la no utilización de patrones de diseño que permitan el uso adecuado de la plataforma. A continuación se describe la adopción de una metodología para el desarrollo de una aplicación sobre J2EE.

4.1 METODOLOGIA DE DESARROLLO

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos de software. Además, detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla.
Hay un gran número de factores que repercuten en la persona que trabaja dentro de un entorno de desarrollo software. Los cambios en el sistema operativo, el lenguaje de programación, la organización del proyecto, o los estándares establecidos para los diferentes aspectos del ciclo de vida de un proyecto pueden influir tanto en el trabajador como en la cantidad de trabajo que puede realizar.

Los programadores tradicionales argumentan que la aplicación de una metodología supone una gran carga. Es cierto, pero si no se emplea una metodología pueden surgir los siguientes problemas:

- Resultados impredecibles.
- Detección tardía de errores.
- La introducción de nuevas herramientas afectará perjudicialmente al proceso.
- Cambios de organización también afectarán al proceso.
- Resultados distintos con nuevas clases de productos.

Para garantizar el éxito en el desarrollo de la aplicación para la automatización del Placement Test, se adoptó la metodología de desarrollo Proceso Unificado (UP) junto con Unified Modeling Language (UML) como lenguaje de modelamiento, ya que son herramientas conocidas y manejadas por la desarrolladora del Proyecto.

4.2 AUTOMATIZACIÓN DEL PLACEMENT TEST-ESPECIFICACIÓN DE REQUISITOS

4.2.1. ESENCIA DEL SISTEMA

Los exámenes de clasificación para los cursos de inglés ofrecidos por el programa de formación en inglés PFI de la universidad del Cauca, se basan en la filosofía de los Paper Based Test PBT en los cuales los datos escritos deben ser procesados posteriormente para procesos de evaluación y generación de resultados. Esto genera inconsistencia en los resultados obtenidos debido a errores en el procesamiento de los datos, pérdida de datos por exposición de los formularios del examen a condiciones adversas y demora en la entrega de resultados. La automatización del Placement Test permitirá la realización en Web del Placement Test (examen utilizado por el PFI para la clasificación en los cursos ofrecidos por este programa), la gestión de la información del examen, de las personas que presenten el examen, de los administradores del sistema, y de los formularios del Placement Test. Adicionalmente el sistema realiza la clasificación automática en los niveles de inglés de las personas que presenten el examen, y estos resultados podrán ser consultados vía Web, por medio de una aplicación desarrollada bajo la plataforma J2EE.

El sistema ha sido desarrollado para el Programa de Formación en Inglés PFI, cuya labor se enfoca en brindar a la comunidad académica de la Universidad del Cauca cursos básicos de inglés. El ingreso a los diferentes niveles de inglés ofrecidos por este programa, va acompañado de un proceso de clasificación, que consiste en la presentación de un examen (el Placement Test) en un formato de papel y en un cálculo de los resultados de cada examen presentado para su posterior clasificación en los diferentes niveles de inglés.

Por esta razón el sistema de automatización del Placement Test es de gran utilidad para el PFI, al soportar la presentación del exámen vía Web y al generar resultados de clasificación automáticamente, traduciéndose esto en un ahorro de tiempo significativo, ahorro de papel, y confianza en los resultados obtenidos.

El sistema está diseñado para que exista un administrador principal para el que el sistema se convierte en una herramienta de gestión para el control de usuarios y de información, un administrador que realiza procesos de consulta de información y un candidato que son todas las personas que van a presentar el Placement Test.

4.2.2. DESCRIPCIÓN GENERAL DEL SISTEMA

El sistema de automatización del Placement Test bajo la plataforma J2EE consiste en una aplicación Web con objetivos claramente diferenciados:

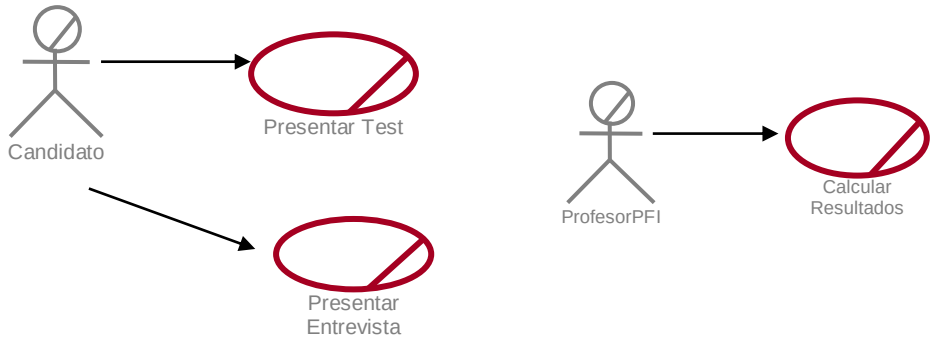
- Desarrollar una herramienta que genere formatos de exámenes de clasificación que se adapten al formato original del Placement Test, con un componente adicional que es la introducción de la sesión de Escritura.
- Permitir la presentación y consulta de resultados del exámen a través de la Web.
- Realizar automáticamente la clasificación en los diferentes niveles de inglés.
- Soportar algún mecanismo seguro de autenticación para restringir el acceso de usuarios al sistema.
- Permitir que toda la información del sistema sea gestionada por un administrador a través de la Web.

4.2.3. GLOSARIO

- **Placement Test:** examen de clasificación a los niveles de inglés ofrecidos por el Programa de Formación en Inglés de la universidad del Cauca.
- **Formulario Test:** formato en papel del Placement Test. Contiene 4 sesiones: Escucha, Gramática, Vocabulario y Lectura.
- **PFI:** Programa de formación en inglés
- **Usuarios:** incluye a todas las personas que van a interactuar con el sistema, es decir los administradores y los que van a presentar el examen.

4.2.4. MODELO DEL NEGOCIO

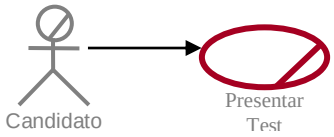
Para lograr sus procesos de clasificación en los diferentes niveles de inglés, el PFI utiliza el Placement Test en formato en papel. Su presentación se resume en el desarrollo de un formulario completo que incluye 5 áreas del idioma inglés: Escucha, Gramática, Vocabulario, Lectura, y Habla. Posteriormente, algunos de los integrantes del PFI, se encargan del cálculo de resultados del exámen para la clasificación por nivel. El objetivo de este modelo del negocio, es describir detalladamente cómo se lleva a cabo cada proceso del negocio, y los actores que hacen parte del mismo incluyendo datos, tareas, roles (o agentes) y reglas del negocio.



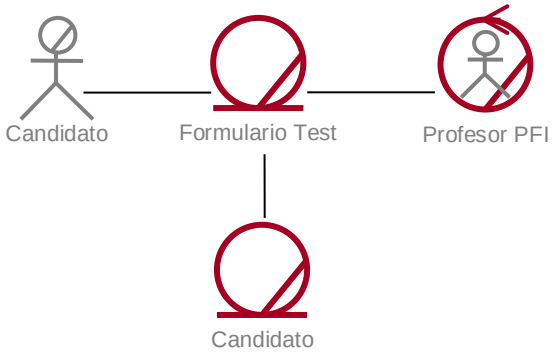
CASOS DE USO DEL NEGOCIO

Este modelo permitirá tener una visión general de las diferentes actividades de negocio de la organización. Estas actividades conforman los casos de uso del negocio. Los actores del negocio desempeñan funciones externas al sistema.

I. CASO DE USO: Presentar Test



Modelo de Objetos del Negocio



Actores del negocio

Candidato: Persona que presenta el Placement Test.

Trabajadores del negocio

Profesor del PFI: persona encargada de recibir los formularios a los candidatos una vez resuelvan la prueba.

Entidades del negocio

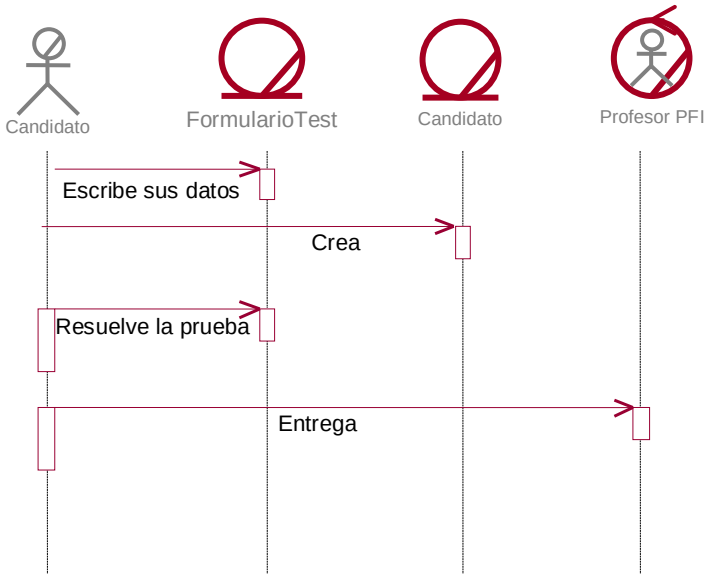
Candidato: Son los datos de todas las personas que presentan el Test (Candidatos) incluyendo los resultados de la prueba.

Formulario del Test: es el formulario que contiene la prueba que debe presentar el Candidato y los datos del mismo.

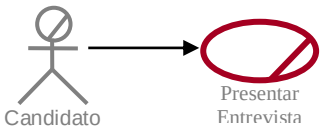
Descripción

Caso de uso	Presentar Test
Actores	Candidato (iniciador)
Descripción	El Candidato recibe el formulario.
	El Candidato escribe sus datos.
	El Candidato resuelve la prueba.
	El Candidato termina la prueba.
	El Candidato entrega el formulario.

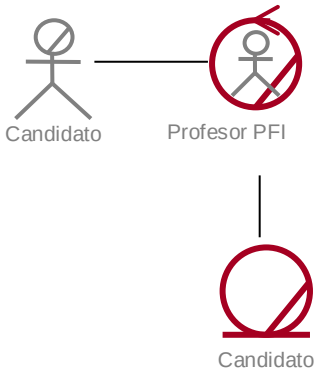
Diagrama de Secuencia



II. CASO DE USO: Presentar Entrevista



Modelo de Objetos del Negocio



ROLES

Actores del negocio

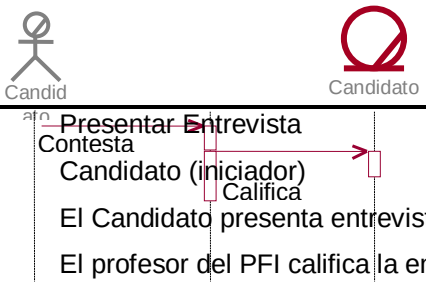
Candidato: Persona que presenta la entrevista.

Entidades del negocio

Candidato.

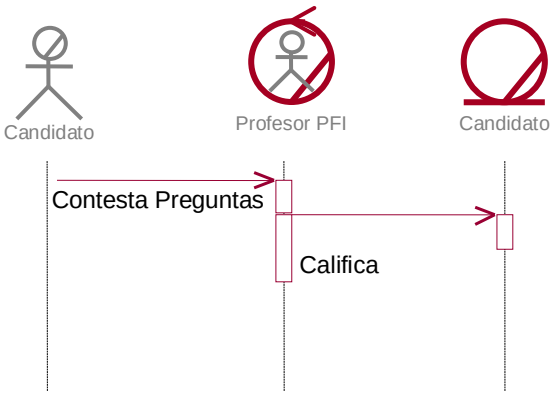
Profesor del PFI

Descripción

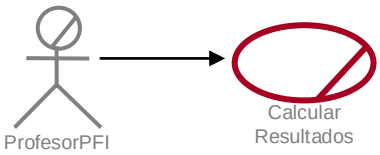


Caso de uso	Presentar Entrevista
Actores	Candidato (iniciador)
Descripción	El Candidato presenta entrevista. El profesor del PFI califica la entrevista.

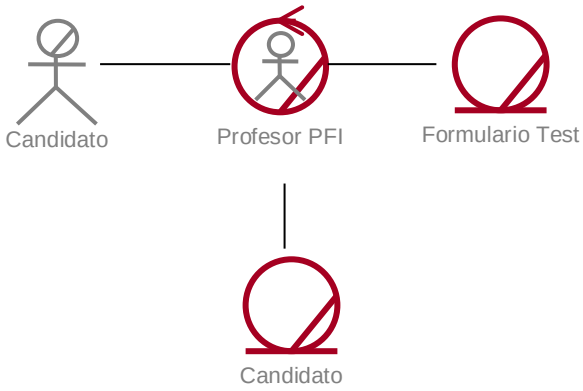
Diagrama de Secuencia



III. CASO DE USO: Calcular resultados



Modelo de Objetos del Negocio



ROLES

Trabajadores del negocio

Profesor del PFI: es la persona encargada de hacer la entrevista al Candidato.

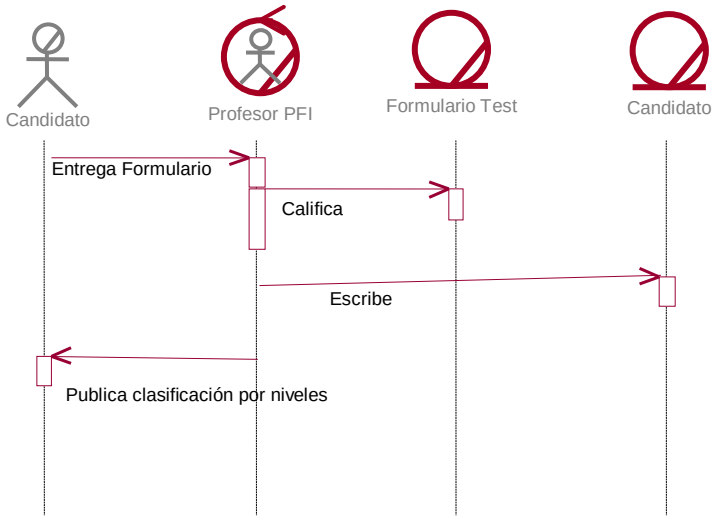
Entidades del negocio

Candidato.

Descripción

Caso de uso	Calcular resultados
Actores	Profesor del PFI (iniciador)
Trabajadores del negocio	Profesor del PFI
Descripción	El Profesor del PFI verifica las respuestas.
	El Profesor del PFI clasifica a los estudiantes por niveles de acuerdo a los resultados obtenidos.
	El Profesor del PFI publica la clasificación de los estudiantes por niveles.

Diagrama de Secuencia



4.2.5 MODELO DE DESARROLLO

ARQUITECTURA DEL SISTEMA

El sistema de automatización del Placement Test se basa en una arquitectura n-capas que hace uso del patrón de diseño Modelo vista Control (MVC), permitiendo una separación total entre lógica de negocio, presentación y acceso a datos. El paradigma MVC consiste en dividir las aplicaciones en tres partes:

- Control
- Modelo
- Vistas.

El ***control*** es el encargado de redirigir o asignar una aplicación (un modelo) a cada petición. En nuestra aplicación el control está representado por Servlets y por EJB´s de sesión.

El ***modelo*** seria la aplicación que responde a una petición, es decir, es la lógica de negocio. En esta solución esta función la cumplen los EJB´s.

Una vez realizadas las operaciones necesarias el flujo vuelve al control y este devuelve los resultados a una ***vista*** asignada (representada por JSP´s).

Otro patrón utilizado en la arquitectura del sistema es el patrón de fachada. Este patrón se implementa como un bean de sesión. Maneja las relaciones entre varios objetos de negocio y proporciona una abstracción de alto nivel para el cliente.

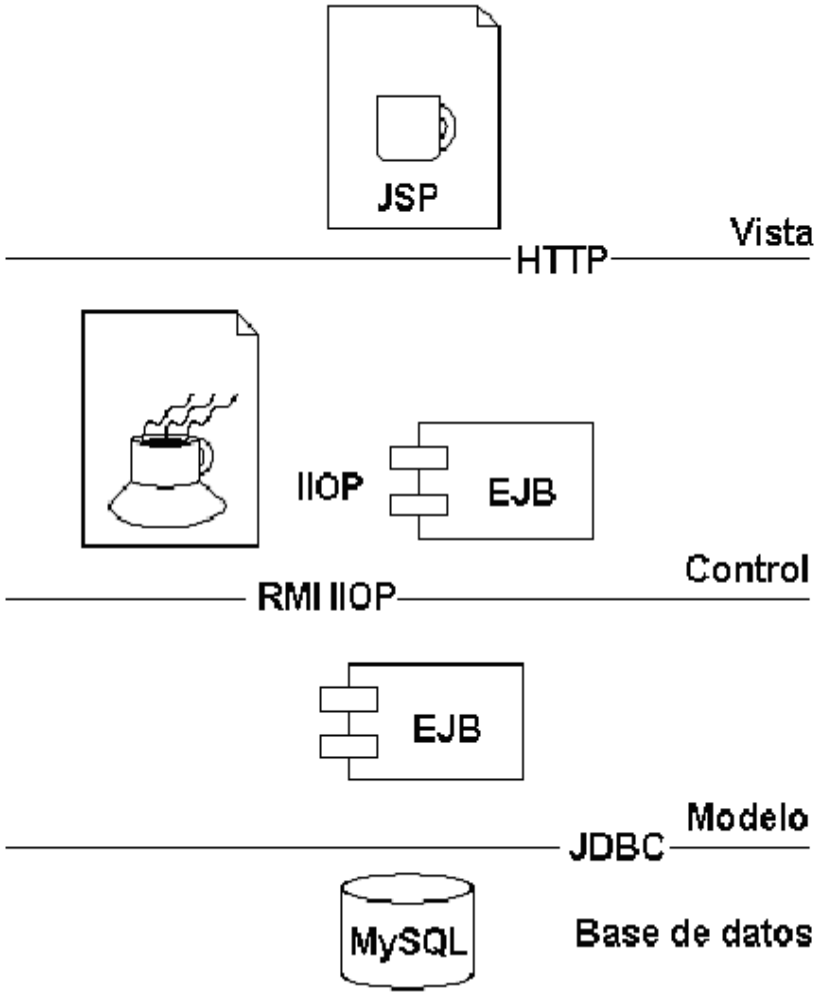


Ilustración 5. Arquitectura de la Aplicación

Las tecnologías de comunicación empleadas fueron los protocolos HTTP, IIOP, RMI-IIOP.

FUNCIONES DEL SISTEMA

CR1. El sistema gestiona los datos de los usuarios

CR2. El sistema valida los datos de los usuarios

CR3. El sistema gestiona los datos de los Formularios del Placement Test

CR4. El sistema gestiona los datos de las sesiones de los Formularios del Placement Test
CR5. El sistema gestiona las preguntas de los Formularios del Placement Test
CR6. El sistema gestiona las respuestas de los Formularios del Placement Test
CR7. El sistema gestiona los tópicos de la sesión de Escritura
CR8. El sistema soporta la presentación del Placement Test
CR9. El sistema gestiona los exámenes presentados
CR10. El sistema soporta el registro de los candidatos
CR11. El sistema arroja el nivel de clasificación de los candidatos

IDENTIFICACIÓN DE ROLES

Administrador principal: persona encargada del mantenimiento general de la información del sistema.
Administrador: persona encargada de realizar algunas tareas relacionadas a la gestión de los exámenes presentados.
Candidato: persona que presenta el Placement Test.

DEFINICIONES

- **Formulario:** hace referencia a los tipos de formatos del examen. Se identifican con una letra mayúscula. Actualmente existen 3 formularios del Placement Test: Formularios A, B y C.
- **Tópico:** Tema del que se tiene que hacer una composición escrita en la sesión de escritura del Placement Test.
- **Sesión:** se refiera a las áreas de inglés que forman el exámen: Escucha, Gramática, Vocabulario, Lectura, Escritura y Habla.

IDENTIFICACION DE LOS CASOS DE USO DE ALTO NIVEL

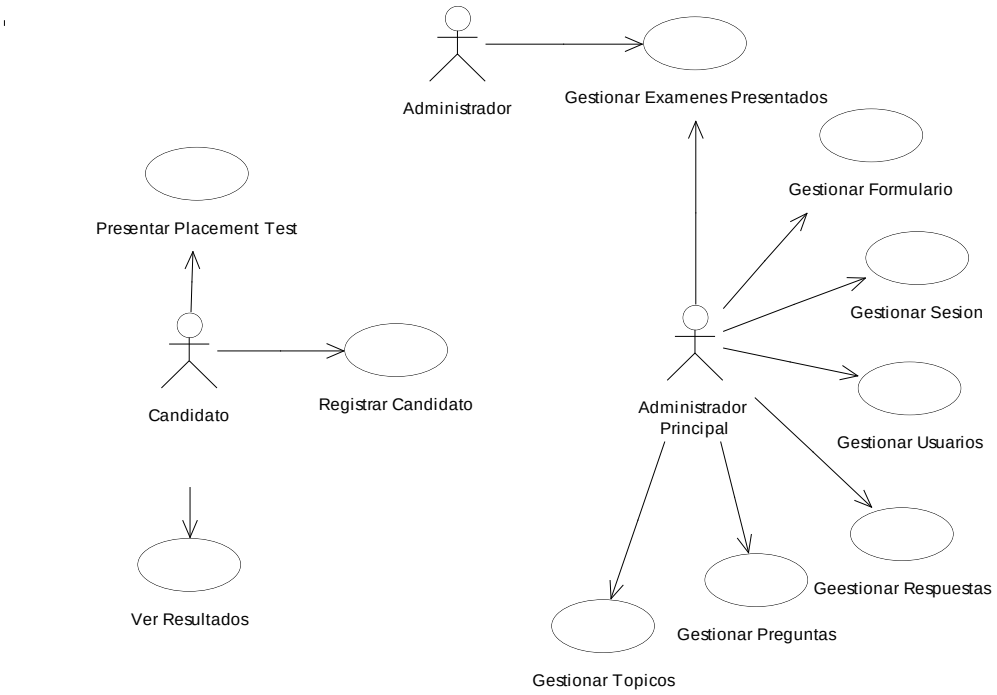


Ilustración 6. Casos de Uso de Alto Nivel

DESCRIPCIÓN DE LOS CASOS DE USO DE ALTO NIVEL

	1
Caso de Uso	**PRESENTAR PLACEMENT TEST**
Tipo	Primario
Actores	Candidato (iniciador)
Resumen	Este caso de uso inicia cuando el candidato elige nuevo examen en la interfaz que le despliega el sistema una vez se haya autenticado. El sistema despliega por cada sesión una interfaz que muestra las instrucciones respectivas y seguidas de estas, las preguntas correspondientes a cada tipo de sesión. El candidato debe elegir una respuesta y confirmarla para poder continuar con las otras preguntas. El tiempo por sesión estará controlado por el sistema, de manera que si se termina el tiempo máximo para cualquiera de las sesiones sin haber terminado de contestar las preguntas respectivas, el sistema pasa automáticamente a la siguiente sesión. Este caso de uso termina cuando el candidato decide terminar el Placement Test o cuando termina de desarrollar la sesión de escritura.
Precondiciones	*Casos de Uso:* Validar usuario, Registrar Candidato, Gestionar

	Formularios, Gestionar sesiones, Gestionar preguntas, Gestionar respuestas.
Referencias Cruzadas a Funciones	CR-2, CR-5, CR-8, CR4, CR6, CR10, CR7, CR3.

	2
Caso de Uso	**GESTIONAR USUARIOS**
Tipo	Primario
Actores	Administrador Principal (iniciador).
Resumen	Este caso de uso inicia cuando el Administrador principal elige usuarios de la interfaz que le despliega el sistema una vez se haya autenticado. El sistema le despliega una interfaz con todos los registros de los usuarios que existen en la base de datos. El administrador Principal puede crear un nuevo usuario, Buscar uno o varios usuarios, Editar los datos de los usuarios y eliminar un usuario. Cuando el Administrador Principal termina de gestionar los usuarios, escoge la opción de regresar al home del Placement Test o cuando termina de desarrollar la sesión de escritura.
Precondiciones	*Casos de Uso:* Validar Usuario
Referencias Cruzadas a Funciones	CR-2, CR1.

	3
Caso de Uso	**GESTIONAR FORMULARIOS**
Tipo	Primario
Actores	Administrador Principal (iniciador).
Resumen	Este caso de uso inicia cuando el Administrador principal elige formularios de la interfaz que le despliega el sistema una vez se haya autenticado. El sistema le despliega una interfaz con todos los registros de los formularios que existen en la base de datos. El administrador Principal puede crear un nuevo formulario, Buscar un formulario, Editar los datos de los formularios y eliminar un formulario. Cuando el Administrador Principal termina de gestionar los formularios, escoge la opción de regresar al home del Placement Test.
Precondiciones	*Casos de Uso:* Validar Usuario
Referencias Cruzadas a Funciones	CR-2, CR3.

	4
Caso de Uso	**GESTIONAR SESIONES**

Tipo	Primario
Actores	Administrador Principal (iniciador).
Resumen	Este caso de uso inicia cuando el Administrador principal elige sesiones de la interfaz que le despliega el sistema una vez se haya autenticado. El sistema le despliega una interfaz con todos los registros de las sesiones que existen en la base de datos. El administrador Principal puede crear un nueva sesión, Buscar una o varias sesiones, Editar los datos de las sesiones y eliminar una sesión. Cuando el Administrador Principal termina de gestionar las sesiones, escoge la opción de regresar al home del Placement Test.
Precondiciones	*Casos de Uso:* Validar Usuario
Referencias Cruzadas a Funciones	CR-2, CR4.

	5
Caso de Uso	**GESTIONAR PREGUNTAS**
Tipo	Primario
Actores	Administrador Principal (iniciador).
Resumen	Este caso de uso inicia cuando el Administrador principal elige no de pregunta de la interfaz de las sesiones. El sistema le despliega una interfaz con todos los registros de las preguntas que existen en la base de datos. El administrador Principal puede crear un nueva pregunta, Buscar una o varias preguntas, Editar los datos de las preguntas y eliminar una pregunta. Cuando el Administrador Principal termina de gestionar las preguntas, escoge la opción de regresar al home del Placement Test.
Precondiciones	*Casos de Uso:* Validar Usuario
Referencias Cruzadas a Funciones	CR-2, CR5.

	6
Caso de Uso	**GESTIONAR RESPUESTAS**
Tipo	Primario
Actores	Administrador Principal (iniciador).
Resumen	Este caso de uso inicia cuando el Administrador principal elige respuestas de la interfaz de preguntas. El sistema le despliega una interfaz con todos los registros de las respuestas que existen en la base de datos. El administrador Principal puede Buscar una o varias respuestas y Editar los datos de las respuestas. Cuando el Administrador Principal termina de gestionar las respuestas, escoge la opción de regresar al home del Placement Test.

Precondiciones	*Casos de Uso:* Validar Usuario
Referencias Cruzadas a Funciones	CR-2, CR6.

	7
Caso de Uso	**GESTIONAR TOPICOS**
Tipo	Primario
Actores	Administrador Principal (iniciador).
Resumen	Este caso de uso inicia cuando el Administrador principal elige tópicos de la interfaz que le despliega el sistema una vez se haya autenticado. El sistema le despliega una interfaz con todos los registros de los tópicos que existen en la base de datos. El administrador Principal puede crear un nuevo tópico, Buscar un tópico, Editar los datos de los tópicos y eliminar un tópico. Cuando el Administrador Principal termina de gestionar los tópicos, escoge la opción de regresar al home del Placement Test.
Precondiciones	*Casos de Uso:* Validar Usuario
Referencias Cruzadas a Funciones	CR-2, CR7.

	8
Caso de Uso	**GESTIONAR EXAMENES PRESENTADOS**
Tipo	Primario
Actores	Administrador Principal (iniciador). Administrador (iniciador).
Resumen	Este caso de uso inicia cuando el Administrador principal elige exámenes de la interfaz que le despliega el sistema una vez se haya autenticado. El sistema le despliega una interfaz con todos los registros de los exámenes presentados que existen en la base de datos. El administrador Principal puede Buscar uno o varios exámenes, calificar las sesiones de escritura y entrevista, ver clasificación por nivel. Cuando el Administrador Principal termina de gestionar los exámenes presentados, escoge la opción de regresar al home del Placement Test.
Precondiciones	*Casos de Uso:* Validar Usuario, Presentar Placement Test, Registrar Candidato, Gestionar Formularios, Gestionar sesiones, Gestionar preguntas, Gestionar respuestas.
Referencias Cruzadas a Funciones	CR-2, CR-5, CR-8, CR4, CR6, CR10, CR7, CR3, CR1, CR9, CR11

	9

Caso de Uso	REGISTRAR CANDIDATO
Tipo	Primario
Actores	Candidato (iniciador).
Resumen	Este caso de uso inicia cuando el Candidato se autentica en la interfaz del home del Placement Test. El sistema le despliega una interfaz con un formulario para que el usuario introduzca sus datos. El caso de uso termina cuando el candidato elige registrar de la página de Registro Candidato.
Precondiciones	*Casos de Uso:* Validar Usuario.
Referencias Cruzadas a Funciones	CR-2, CR10

10	
Caso de Uso	**VER RESULTADOS**
Tipo	Primario
Actores	Candidato (iniciador).
Resumen	Este caso de uso inicia cuando el Candidato se autentica en la interfaz del home del Placement Test. El sistema le despliega una interfaz donde puede seleccionar ver resultados. El sistema le despliega una tabla con el resultado de los exámenes presentados por ese usuarios El caso de uso termina cuando el candidato regresa a la página del home del Placement Test.
Precondiciones	*Casos de Uso:* Validar Usuario.
Referencias Cruzadas a Funciones	CR-2, CR10

4.2.6 MODELO DE ANALISIS

CASOS DE USO DEL ADMINISTRADOR PRINCIPAL

- **Acceder al sistema**
- Gestionar Formulario
- Gestionar Sesión
- Gestionar Pregunta
- Gestionar Respuesta
- Gestionar Tópicos
- **Gestionar Usuarios**

- **Gestionar Exámenes Presentados**

CASOS DE USO DEL ADMINISTRADOR

- Acceder al sistema
- **Gestionar Exámenes Presentados**

CASOS DE USO DEL CANDIDATO

- Acceder al sistema
- **Presentar Placement Test**
- Ver resultados

A continuación se realizará una descripción detallada de los siguientes casos de uso:

1. Acceder al sistema. La razón de selección de este caso de uso es porque es una función que se utiliza a lo largo del funcionamiento del sistema.
2. Presentar Placement Test. Este caso de uso es muy importante porque en él se describe de manera detallada la presentación del examen, que es uno de los objetivos principales de la solución planteada.
3. Gestionar Usuarios. Es un modelo típico de las funciones de gestión que realiza el administrador principal. Su filosofía se ve reflejada en los casos de uso Gestionar Formulario, Gestionar Sesión, Gestionar Pregunta, Gestionar Respuesta y Gestionar Tópicos.
4. Gestionar Exámenes Presentados. Es una función de gestión muy importante porque aborda las funciones de ver resultados, calificación de la composición y la entrevista y, clasificación por nivel que corresponden a objetivos de la solución.

La descripción detallada del resto de casos de uso se encuentra en el anexo A2.

DESCRIPCIÓN DE LOS CASOS DE USO EXTENDIDOS

CASO DE USO 1. ACCEDER AL SISTEMA

Actores

Administrador Principal (iniciador), Candidato (iniciador), Administrador (iniciador),.

Resumen

El candidato entra al sistema, el sistema le despliega la interfaz del home del Placement Test para que valide sus datos. El sistema le despliega las interfaces respectivas de acuerdo al rol del usuario, ya sea Administrador Principal, Administrador ó Candidato.

Tipo

Primario.

Referencias cruzadas

CR-2.

Flujo Normal de Eventos

1. El candidato abre el navegador y escribe la URL respectiva.
2. El Sistema Despliega la interfaz del home del Placement Test ***IU1***.
3. El candidato Ingresa el Login y Password y presiona el botón “enviar” ***E1***.
4. El Sistema Valida el Login y Password ***F1***.
5. El Sistema Despliega la Interfaz respectiva a cada usuario ***IU2,IU3, IU5***.

Flujos Alternos

- ***F1***: Si el Login y/ó Password son inválidos, el sistema retorna a la interfaz ***IU1*** con un mensaje de fallo en la autenticación.

Excepciones

- ***E1***: Si se omite el Login y/ó Password, el sistema muestra la interfaz ***IU4*** con un mensaje que informe la forma correcta de ingresar los datos.

Interfaces

- ***IU1***: PlacementTest.jsp.

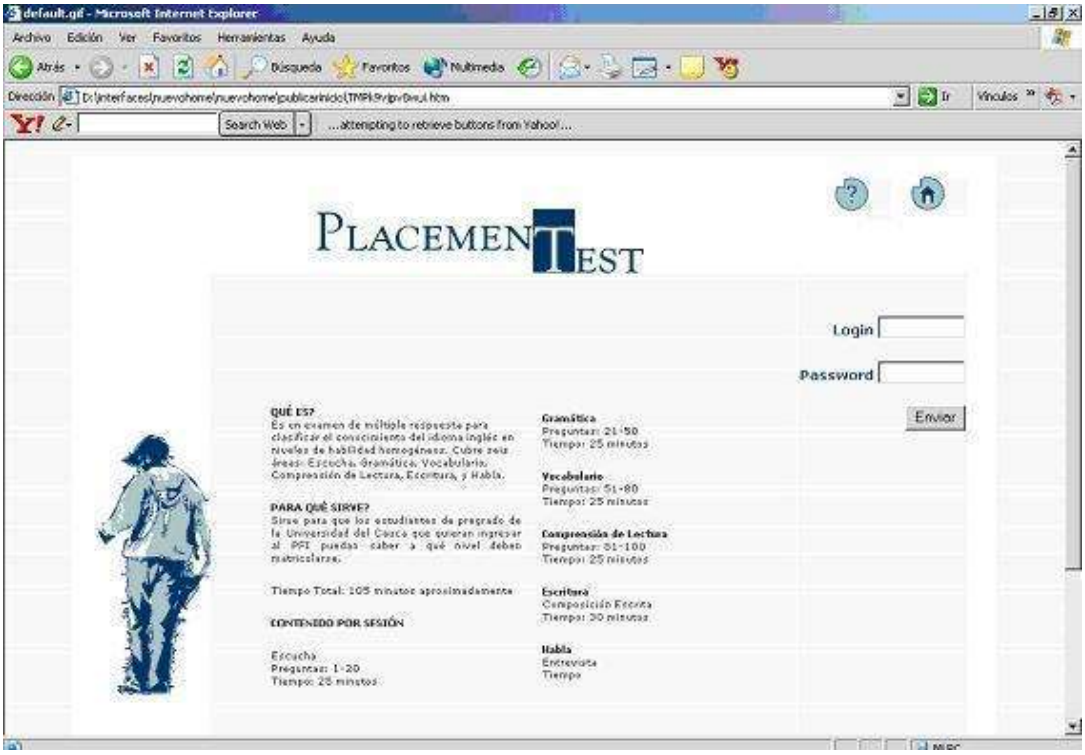


Ilustración 7. IU_Placement Test

- ***IU2:*** AdmPrincipal.jsp.

Ilustración 8. IU_AdmPrincipal

- ***IU3***: Administrador.jsp.

Ilustración 9. IU_Administrador

- ***IU5***: RegistroCandidato.jsp

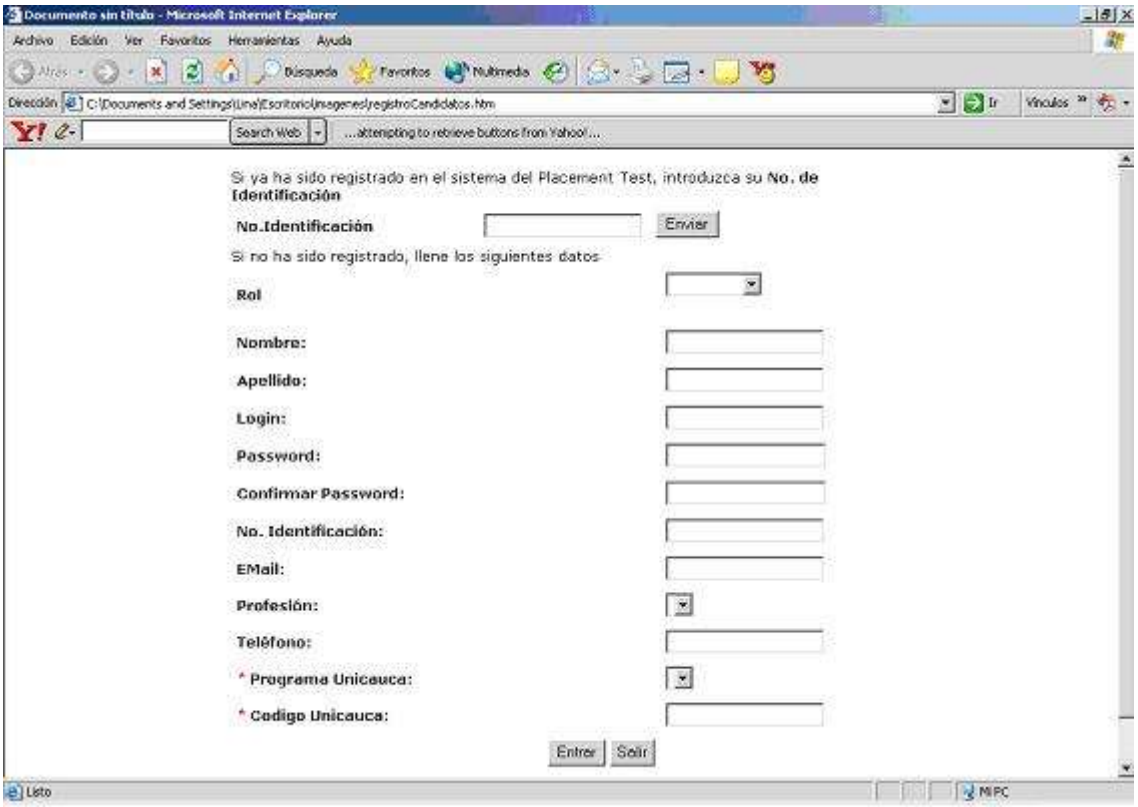


Ilustración 10. RegistrarCandidato

- ***IU4***: Mensajes de alerta

CASO DE USO 2. PRESENTAR PLACEMENT TEST

Actores

Candidato(iniciador).

Resumen

El candidato entra al sistema, el sistema le despliega la interfaz de registro para que el usuario llene el formulario si es la primera vez que va a presentar el examen, de lo contrario, ingresa los datos necesarios para que pueda entrar al sistema. El sistema valida los datos y le despliega las interfaces respectivas para comenzar la presentación de la prueba. Por cada sesión se le despliegan unas instrucciones que el candidato puede leer antes de comenzar a contestar las preguntas. El tiempo por sesión estará controlado por el sistema, de manera que si se termina el tiempo máximo para cualquiera de las sesiones sin haber terminado de contestar las preguntas respectivas el sistema pasa automáticamente a la siguiente sesión. El usuario puede terminar el examen en cualquier momento eligiendo la opción terminar de las interfaces que se le despliegan.

Tipo

Primario.

Referencias cruzadas

CR-2, CR-5, CR-8, CR4, CR6, CR10, CR7, CR3.

Precondición:

Casos de Uso: Validar usuario, Registrar Candidato, Gestionar Formularios, Gestionar sesiones, Gestionar preguntas, Gestionar respuestas.

Flujo Normal de Eventos

1. El Sistema Despliega la Interfaz de Registro de Candidatos ***IU5.***.
2. Si el candidato no ha sido registrado, debe llenar el formulario y presionar el botón “registrar” de la interfaz ***IU5, E2*.**
3. Si el candidato ha sido registrado antes, debe presionar el botón “entrar” de la interfaz ***IU5, E3.***
4. El sistema le despliega la interfaz de exámenes usuario ***IU6***.
5. El candidato presiona el botón “nuevo examen”.

6. El sistema le despliega la interfaz sesiones examen ***IU7***
7. El candidato elige la sesión escucha.
8. El sistema le despliega la página de instrucciones de escucha ***IU8***
9. El candidato presiona el botón “aceptar” de la página ***IU8***
10. El sistema le despliega la interfaz para escuchar el primer diálogo ***IU9***
11. El candidato presiona el botón “escuchar”.
12. El sistema reproduce el diálogo.
13. El candidato presiona el botón “responder”.
14. El sistema le despliega la interfaz donde se le despliegan las respuestas del diálogo escuchado ***IU10***
15. Si el candidato presiona el botón “siguiente”, puede cambiar la respuesta seleccionada.
16. Si el candidato presiona el botón “confirmar”, se le despliega la página ***IU10***
17. Si el candidato presiona el botón “terminar”, el sistema le despliega la interfaz de terminación del exámen ***IU11***
18. El candidato debe seleccionar la razón por la cual ha decidido terminar el exámen y presionar el botón “terminar” ***E4.***
19. Si el candidato presiona el botón “continuar”, el sistema continúa con el exámen desde la pregunta donde quedó y con el tiempo que llevaba en el momento en que presionó el botón “terminar”.

Excepciones

- ***E2***: Si se omite el no de Identificación en ***IU5,*** se despliega la interfaz ***IU4*** con un mensaje de error.
- ***E3***: Si omite algún dato del formulario de registro de candidatos, el sistema le despliega la interfaz ***IU4*** con un mensaje de error.
- ***E4*** Si presiona el botón “terminar” sin haber seleccionado una razón de aborto de la prueba, el sistema le despliega la interfaz ***IU4*** con un mensaje de error.

Interfaces

- ***IU1***: PlacementTest.jsp.
- ***IU4:*** Mensajes alerta
- ***IU5:*** RegistroCandidato.jsp.
- ***IU6***: ExamenesUsuario.jsp.

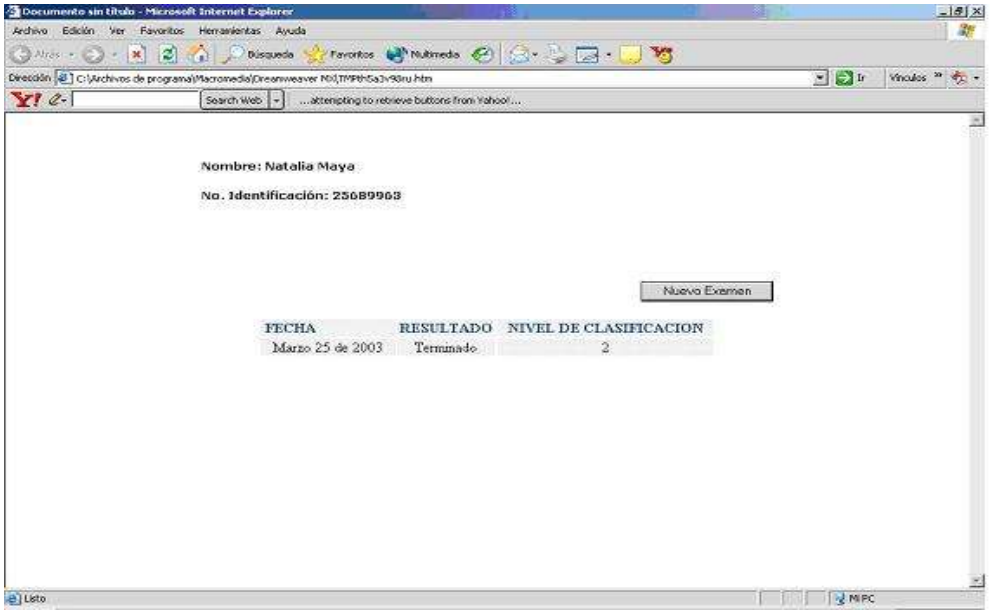


Ilustración 11. IU_ExamenesUsuario

- ***IU7***: SesionesExamen.jsp

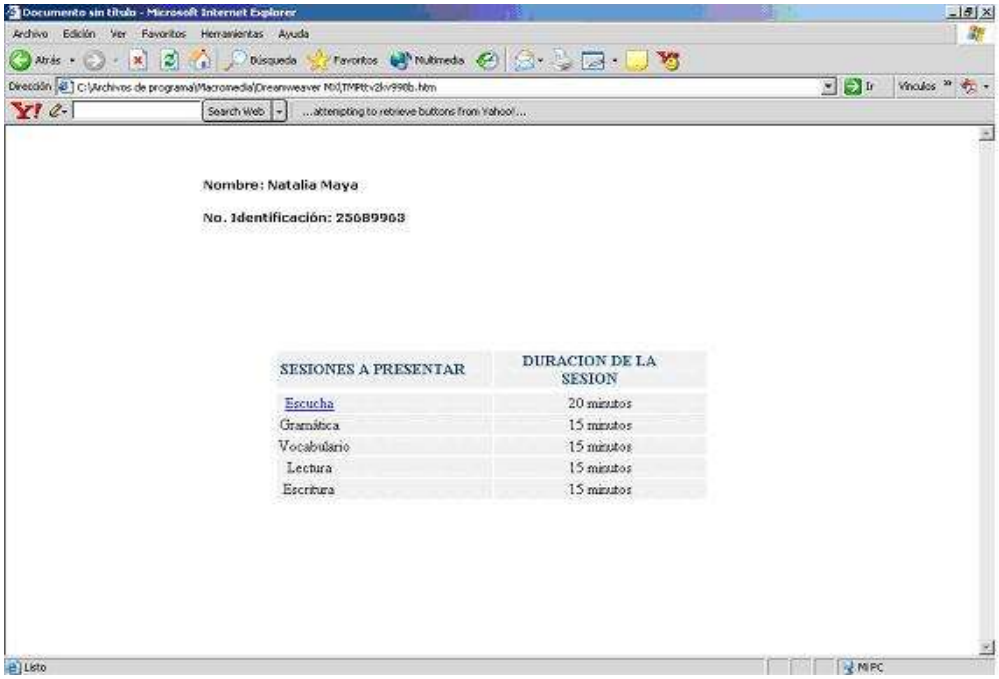


Ilustración 12. SesionesExamen

- ***IU8***: InstruccionesEscucha.jsp

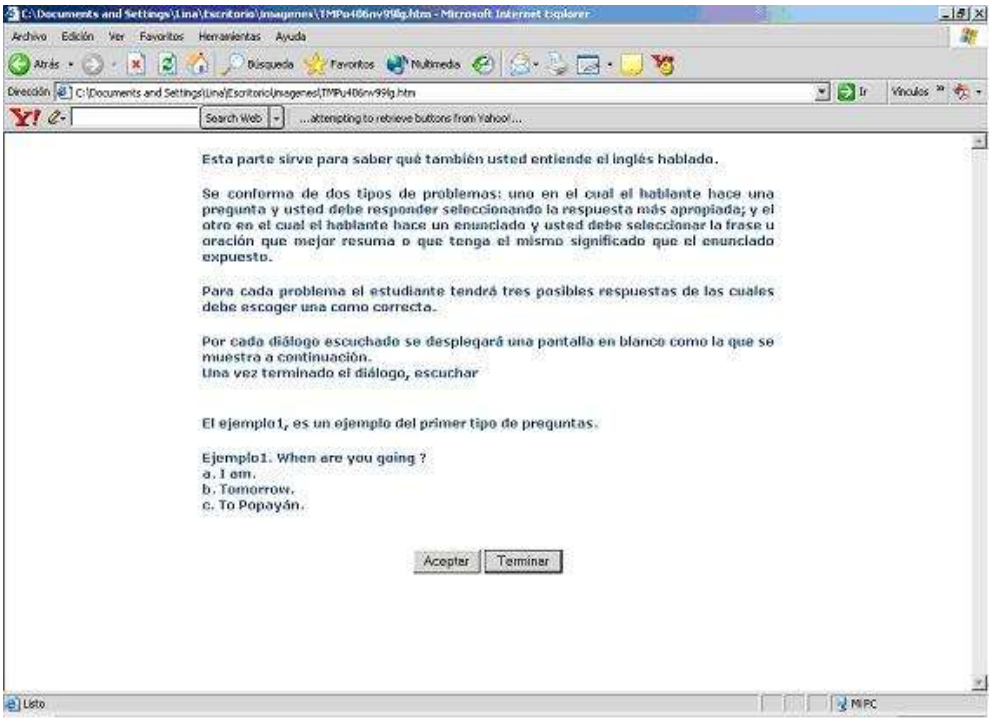


Ilustración 13. IU_Instrucciones

- ***IU9***: Dialogo.jsp

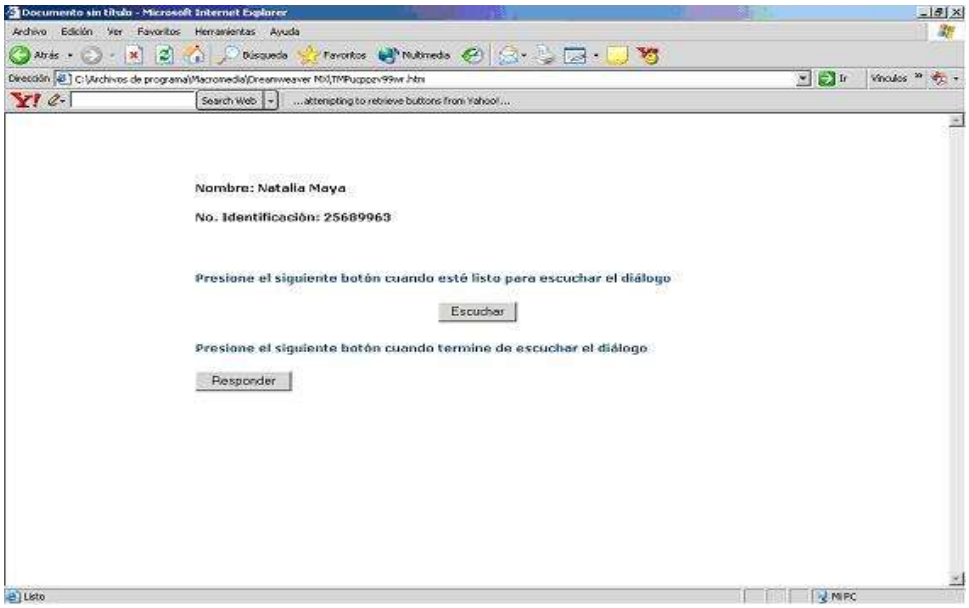


Ilustración 14. IU_Dialogo

- ***IU10***: PreguntasExamen.jsp

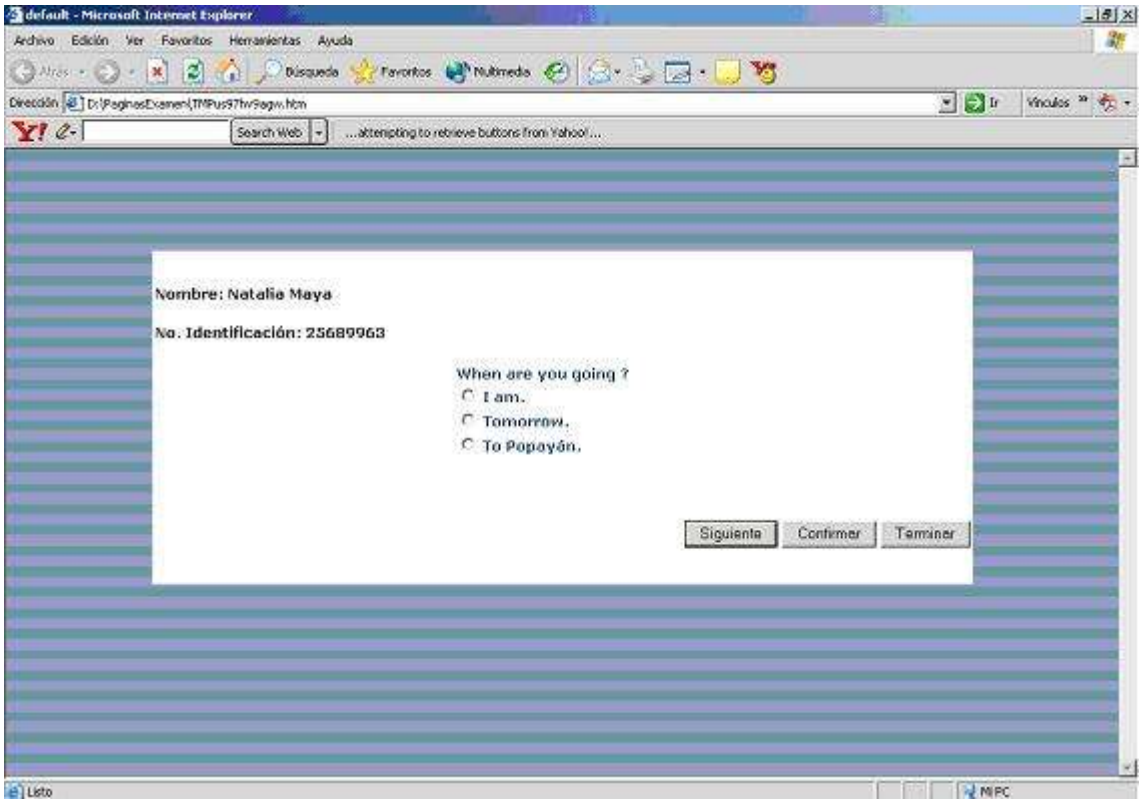


Ilustración 15. IU_PreguntasExamen

- ***IU11***: TerminarExamen.jsp

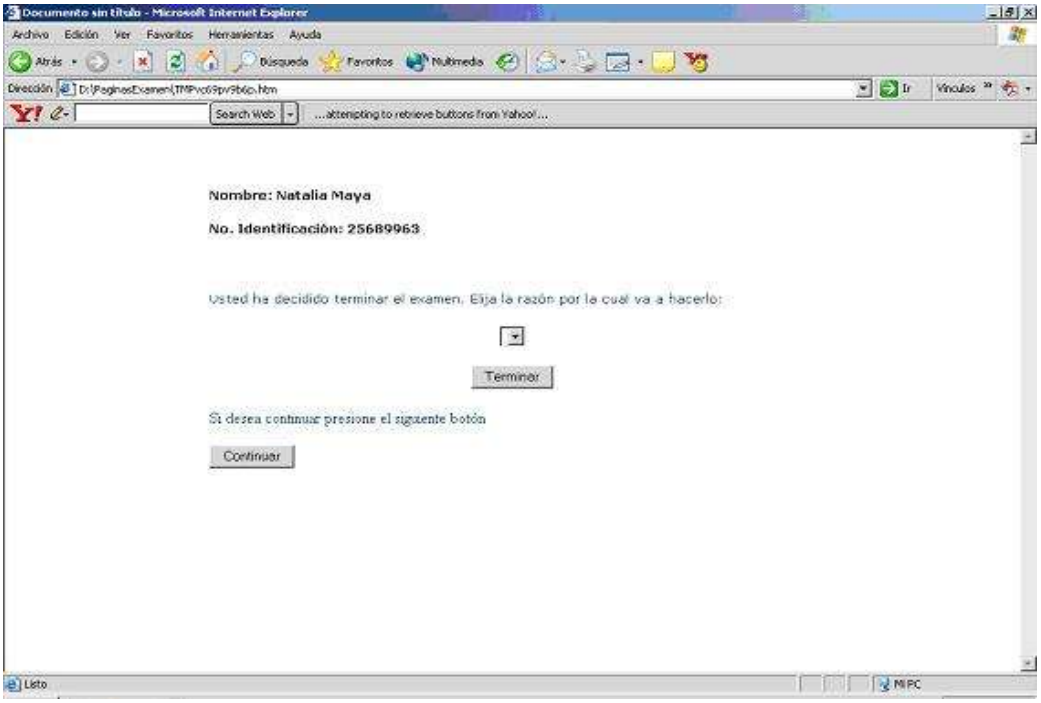


Ilustración 16. IU_TerminarExamen

y

CASO DE USO 3. GESTIONAR USUARIO

Actores

Administrador Principal (iniciador)

Resumen

Este caso de uso inicia cuando el Administrador principal elige usuarios de la interfaz que le despliega el sistema una vez se haya autenticado. El sistema le despliega una interfaz con todos los registros de los usuarios que existen en la base de datos. El administrador Principal puede crear un nuevo usuario, Buscar uno o varios usuarios, Editar los datos de los usuarios y eliminar un usuario.

Si elige crear un nuevo usuario, el sistema le despliega un formulario que se debe llenar con los datos del nuevo usuario. Si elige editar, el sistema le despliega una interfaz con los datos del usuario que pueden ser modificados. Si elige eliminar un usuario, el sistema le despliega una interfaz de confirmación. Cuando el Administrador Principal termina de gestionar los usuarios, escoge la opción Administrador Principal, para regresar a la página principal de gestión del Administrador Principal.

Tipo

Primario

Referencias Cruzadas a Funciones

CR-2, CR1.

Precondiciones

Casos de Uso: Validar Usuario

Flujo Normal de Eventos

1. El sistema le despliega la interfaz de usuarios ***IU12.***
2. El Administrador Principal presiona el botón “buscar”.
3. El Sistema despliega en la tabla los registros que coincidan con el parámetro de búsqueda introducido ***F3***.
4. El Administrador Principal presiona el botón “Nuevo Usuario”.
5. El Sistema le despliega la interfaz crear usuario ***IU13***.
6. El Administrador Principal llena el formulario con todos los datos del usuario ***E5***.
7. El Administrador Principal presiona el botón “Editar Usuario”.

8. El Sistema le despliega la interfaz actualizar usuario ***IU14*** con los datos del usuario que pueden ser modificados**.**
9. El Administrador Principal edita los datos necesarios ***E6.***
10. El sistema le despliega la interfaz de actualización exitosa ***IU16***.
11. El Administrador Principal presiona el botón “Eliminar Usuario”.
12. El Sistema le despliega una interfaz de confirmación de eliminación de usuarios ***IU15.***
13. El sistema le despliega la interfaz de eliminación exitosa ***IU16***.
14. El Administrador Principal presiona el botón “Administrador Principal”.
15. El sistema regresa a la interfaz de gestión del Administrador Principal ***IU2***.

Flujos Alternos

- ***F1***: Si no existe ningún registro en la base de datos con el parámetro de búsqueda introducido, el sistema retorna a la interfaz ***IU12*** con un mensaje de error.

Excepciones

- ***E6***: Si los datos ingresados en el formulario de creación de usuarios no son del tipo correcto o están incompletos, el sistema muestra la interfaz ***IU4*** con un mensaje que informe la forma correcta de ingresar los datos.

Interfaces

- ***IU16***: Mensajes.jsp.
- ***IU4***: mensajes de alerta.

- ***IU12***: Usuarios.jsp.

Ilustración 17. IU_Usuarios

- ***IU13:*** CrearUsuarios.jsp.

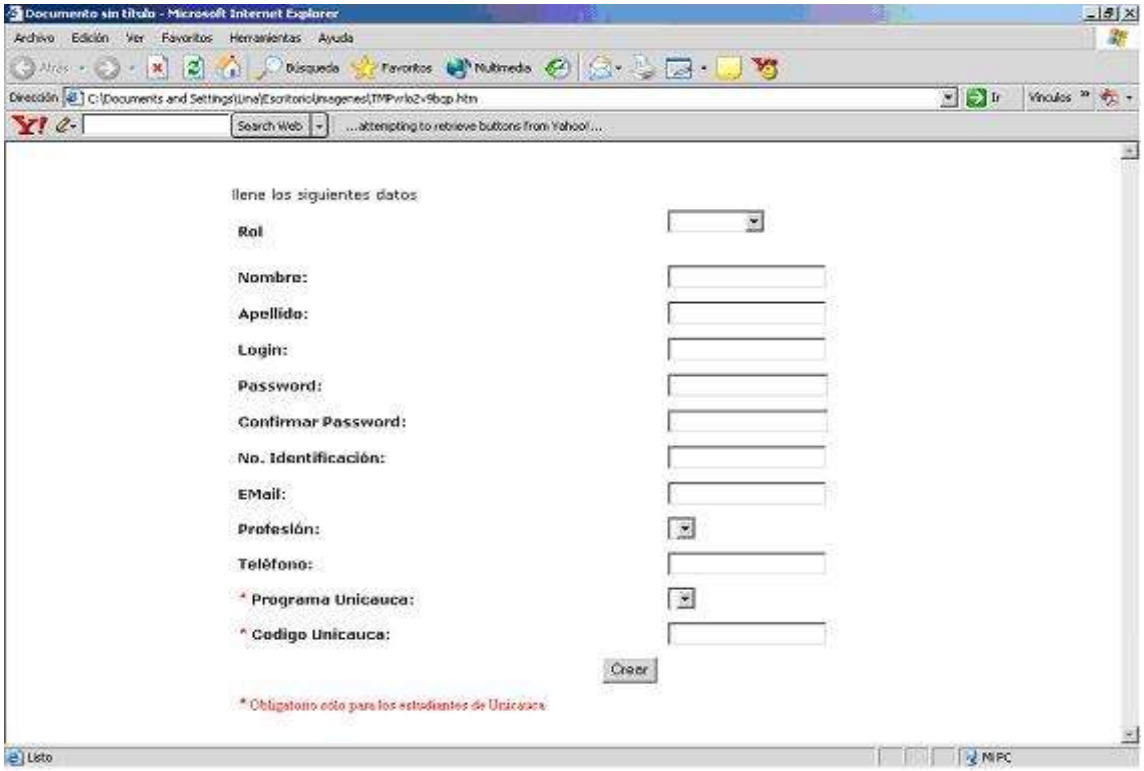


Ilustración 18. IU_CrearUsuarios

- ***IU14***: ActualizarUsuarios.jsp.

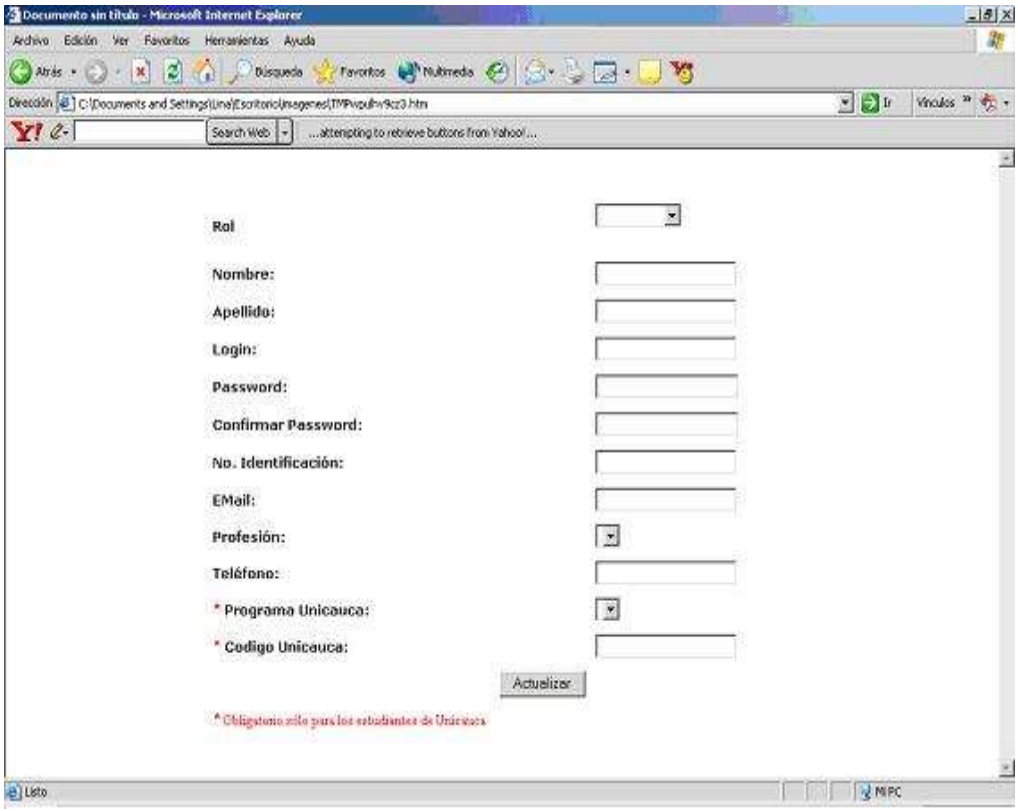


Ilustración 19. IU_ActualizarUsuarios

- ***IU15***: EliminarUsuarios.jsp.

Ilustración 20. IU_EliminarUsuarios

CASO DE USO 4. GESTIONAR EXAMENES PRESENTADOS

Actores

Administrador Principal (iniciador), Administrador.

Resumen

Este caso de uso inicia cuando el Administrador principal elige exámenes de la interfaz que le despliega el sistema una vez se haya autenticado. El sistema le despliega una interfaz con todos los registros de los exámenes presentados que existen en la base de datos. El administrador Principal puede Buscar uno o varios exámenes, calificar las sesiones de escritura y entrevista, ver clasificación por nivel. Cuando el Administrador Principal termina de gestionar los exámenes presentados, escoge la opción de regresar al home del Placement Test.

Tipo

Primario.

Referencias cruzadas

CR-2, CR-5, CR-8, CR4, CR6, CR10, CR7, CR3, CR1, CR9, CR11

Precondiciones

Casos de Uso: Validar Usuario, Presentar Placement Test, Registrar Candidato, Gestionar Formularios, Gestionar sesiones, Gestionar preguntas, Gestionar respuestas.

Flujo Normal de Eventos

1. El sistema le despliega la interfaz de exámenes presentados ***IU17.***
2. El Administrador Principal presiona el botón “buscar”.
3. El Sistema despliega en la tabla los registros que coincidan con el parámetro de búsqueda introducido ***F4***.
4. El Administrador Principal presiona el enlace resultados.
5. El Sistema le despliega la interfaz Calificar Sesión ***IU18***.
6. El Administrador Principal presiona el enlace entrevista.
7. El Sistema le despliega la interfaz calificar entrevista ***IU19.***
8. El Administrador Principal llena la matríz de calificación ***E7.***
9. El sistema le despliega la interfaz de calificación exitosa ***IU16***.
10. El sistema le despliega la interfaz ***IU18***.

11. El Administrador Principal presiona el enlace escritura.
12. El Sistema le despliega la interfaz calificar escritura ***IU20.***
13. El Administrador Principal llena la matríz de calificación ***E7.***
14. El sistema le despliega la interfaz de calificación exitosa ***IU16***.
15. El sistema le despliega la interfaz ***IU18***.
16. El Administrador Principal presiona el botón “Nivel de Clasificación” de la interfaz ***IU18***.
17. El sistema le despliega la interfaz de clasificaciexitosa ***IU16***.
18. El sistema le despliega la interfaz ***IU18***.

Flujos Alternos

- ***F4***: Si no existe ningún registro en la base de datos con el parámetro de búsqueda introducido, el sistema retorna a la interfaz ***IU17*** con un mensaje de error.

Excepciones

- ***E7***: Si se omite la calificación de alguno de los ítems de las matrices de calificación el sistema muestra la interfaz ***IU4*** con un mensaje que informe la forma correcta de ingresar los datos.

Interfaces

- ***IU17***: Examenes.jsp

Ilustración 21. IU_Exámenes

- ***IU18:*** CalificarSesion.jsp

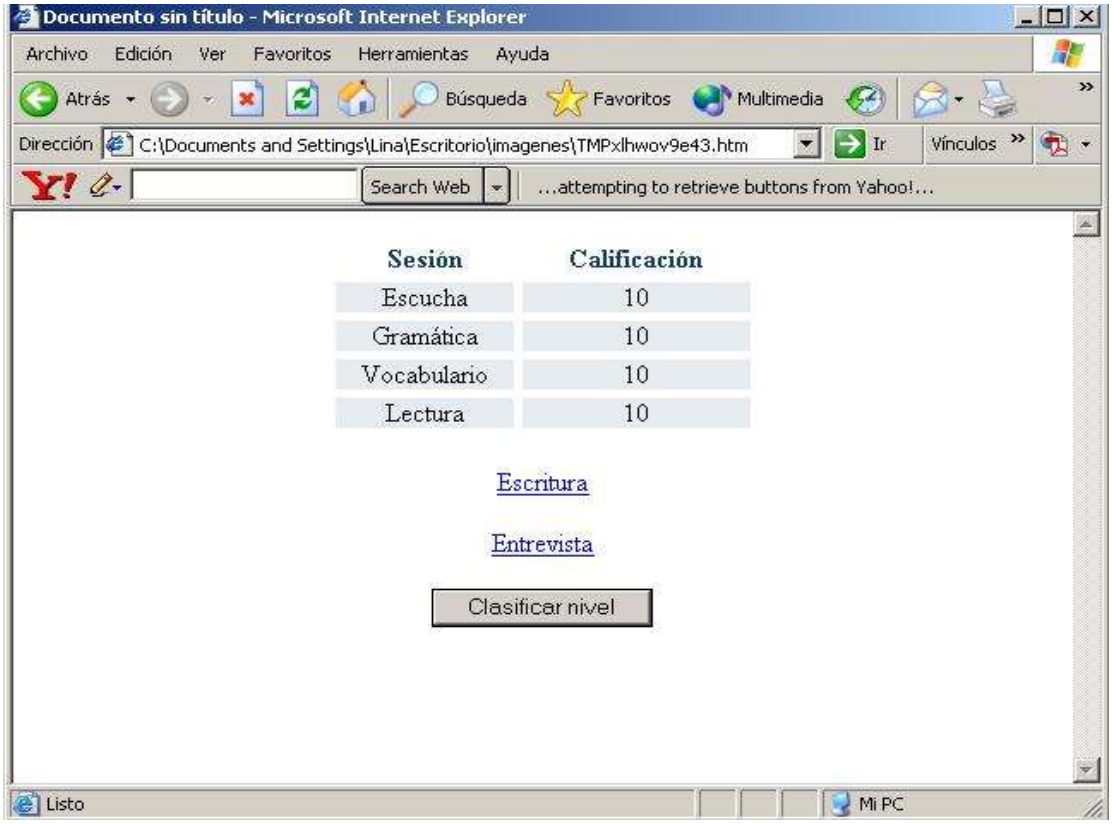


Ilustración 22. IU_CalificaciónSesion

- ***IU19:*** CalificarEntrevista.jsp

Ilustración 23. IU_CalificarEntrevista

- ***IU20:*** CalificarEscritura.jsp

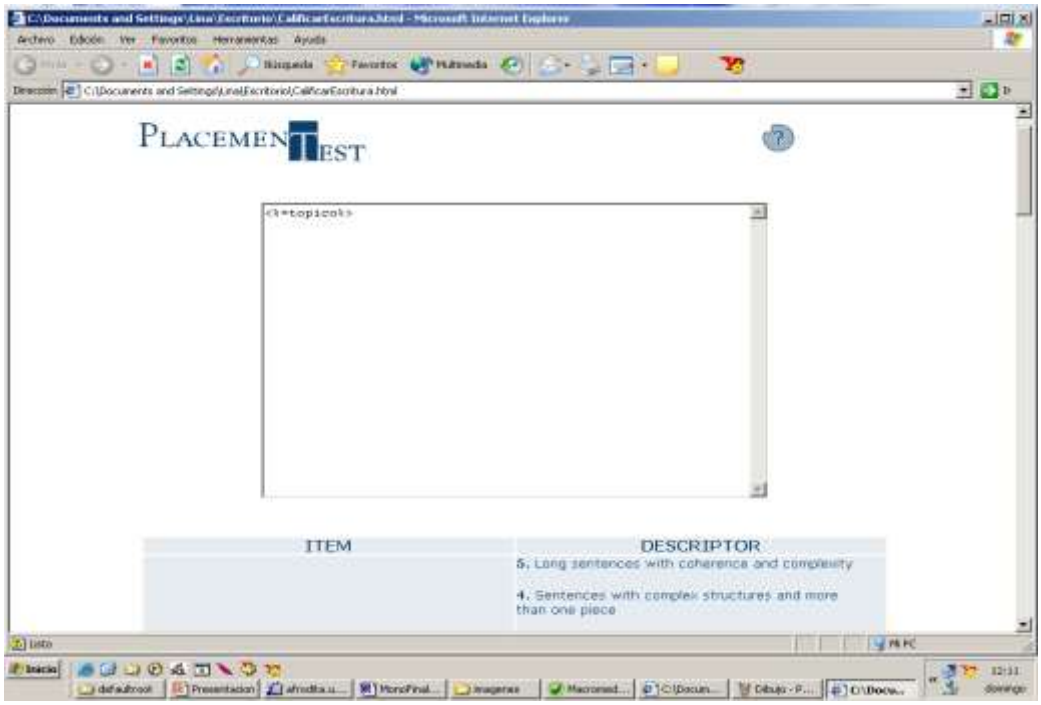
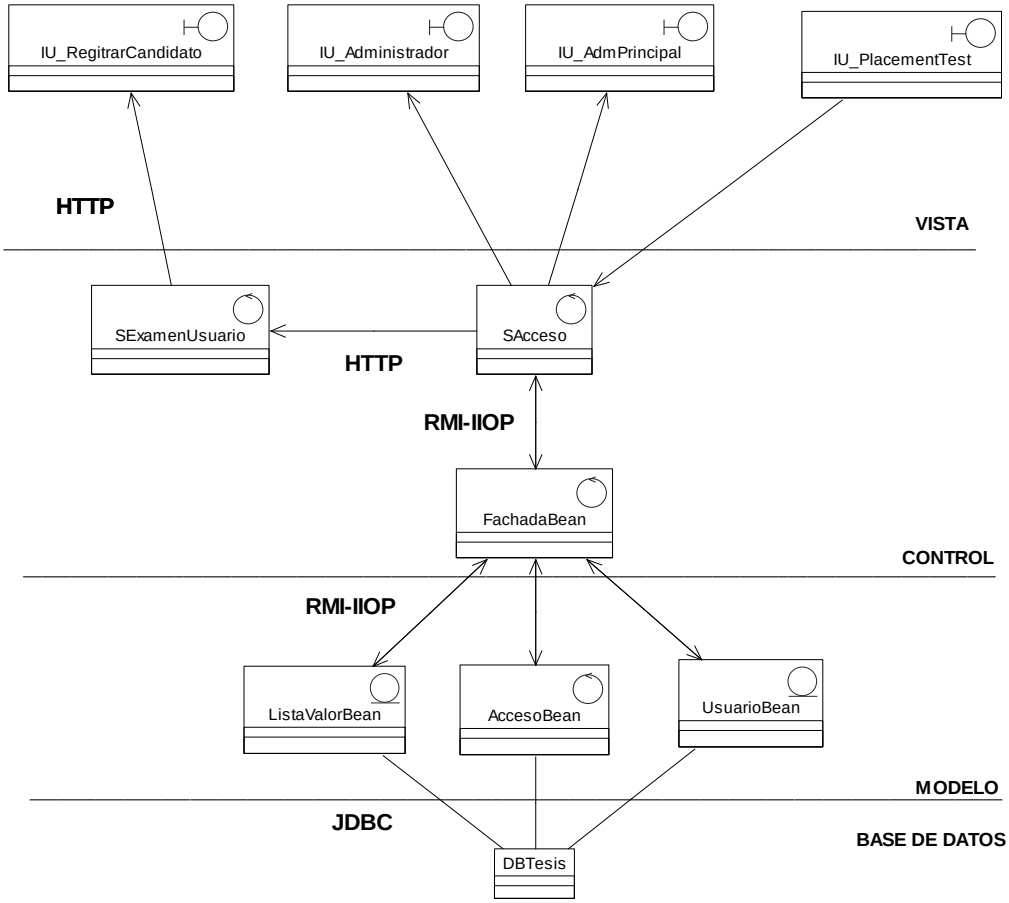


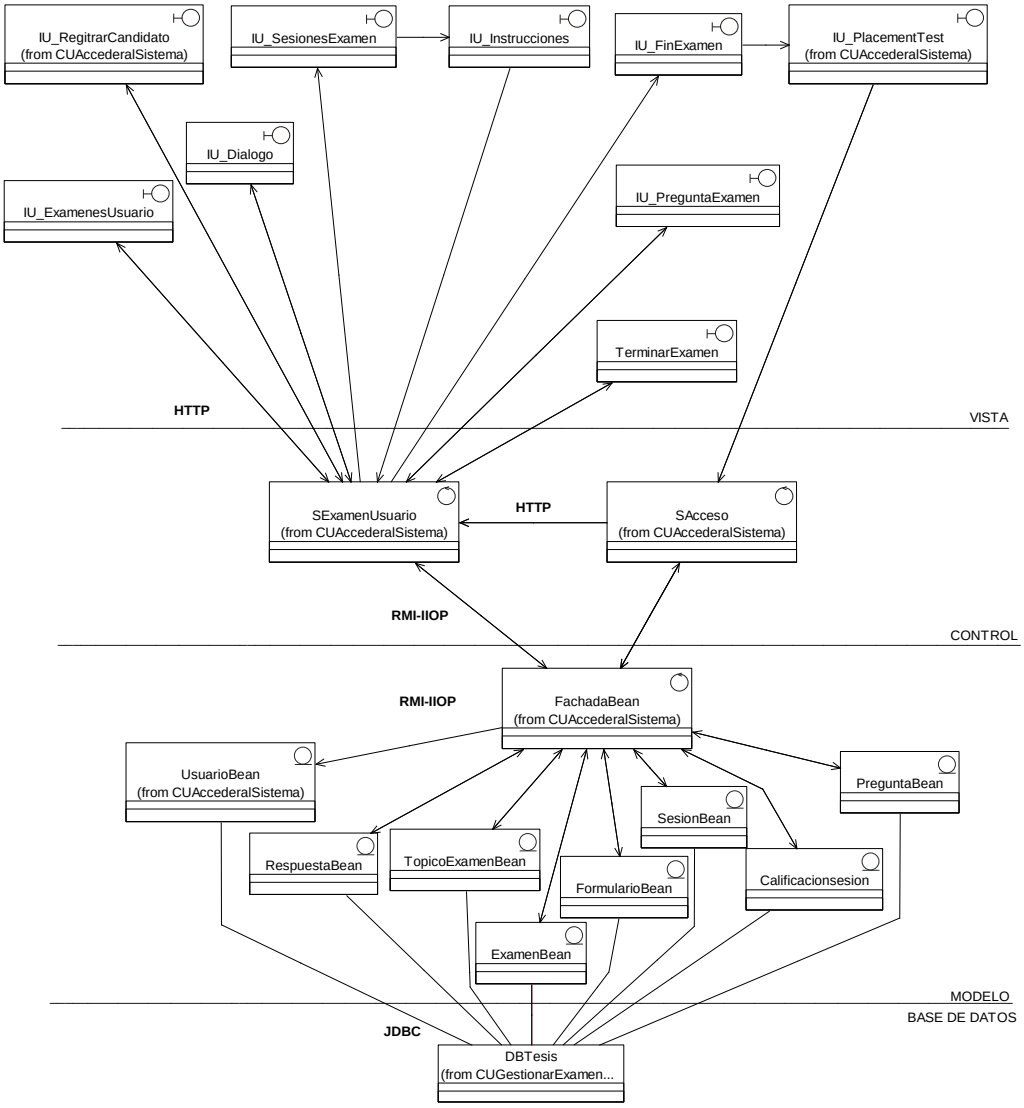
Ilustración 24. IU_CalificarEscritura

Diagramas de Clases-Arquitectura en 4 Niveles

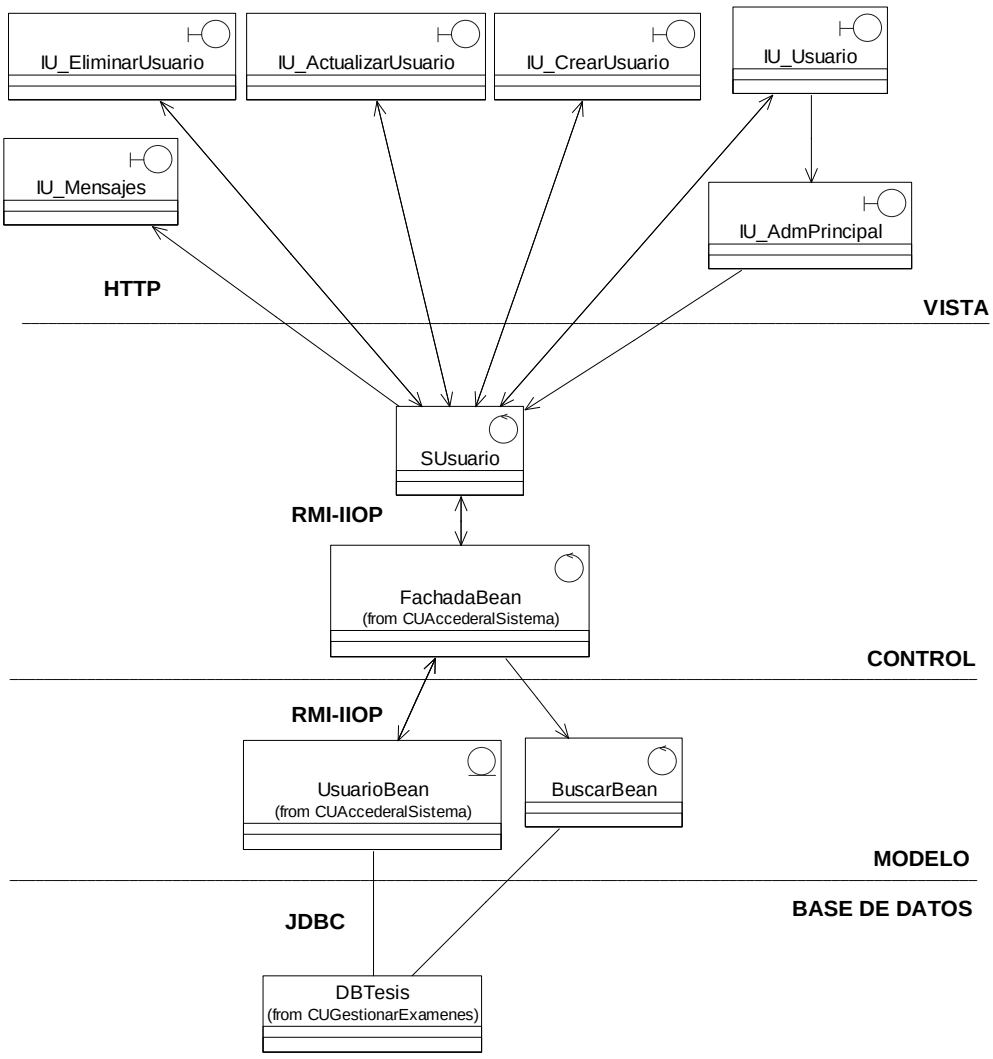
Caso de Uso Acceder al sistema



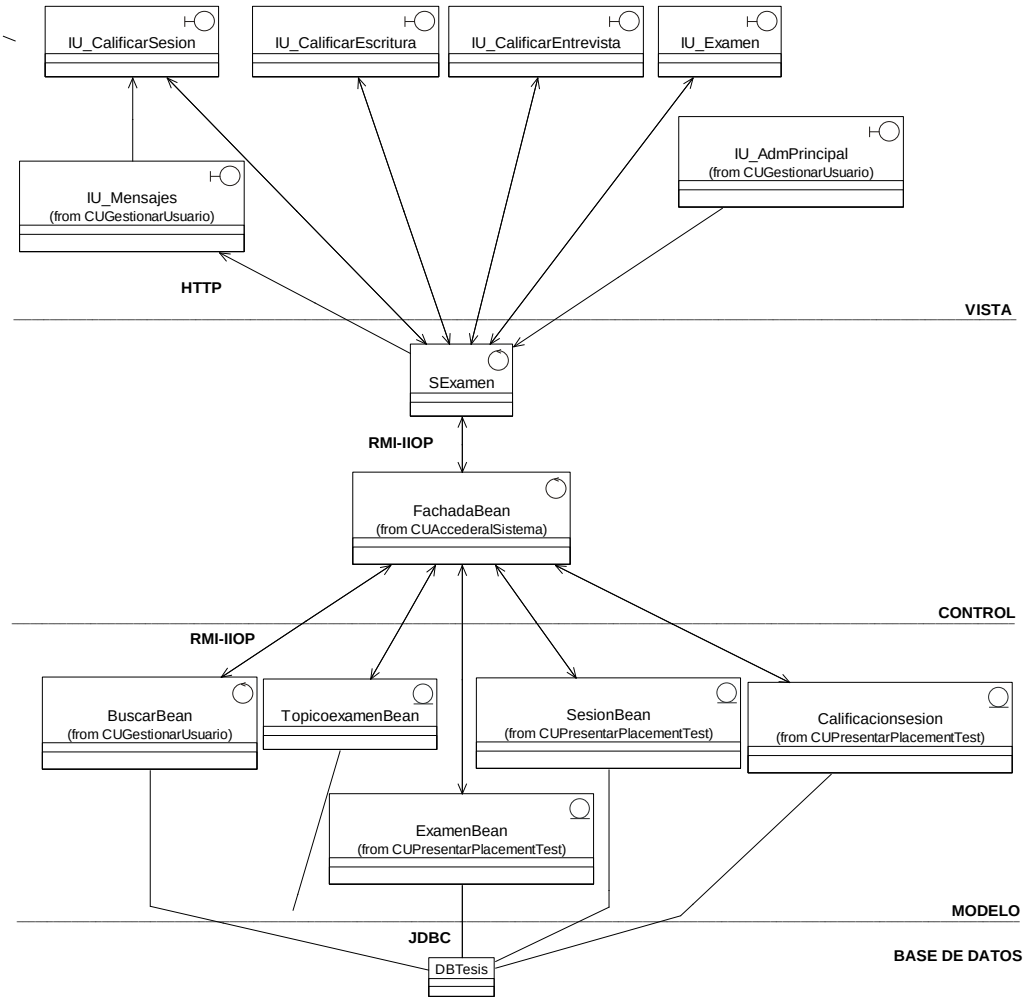
Caso de Uso Presentar Placement Test



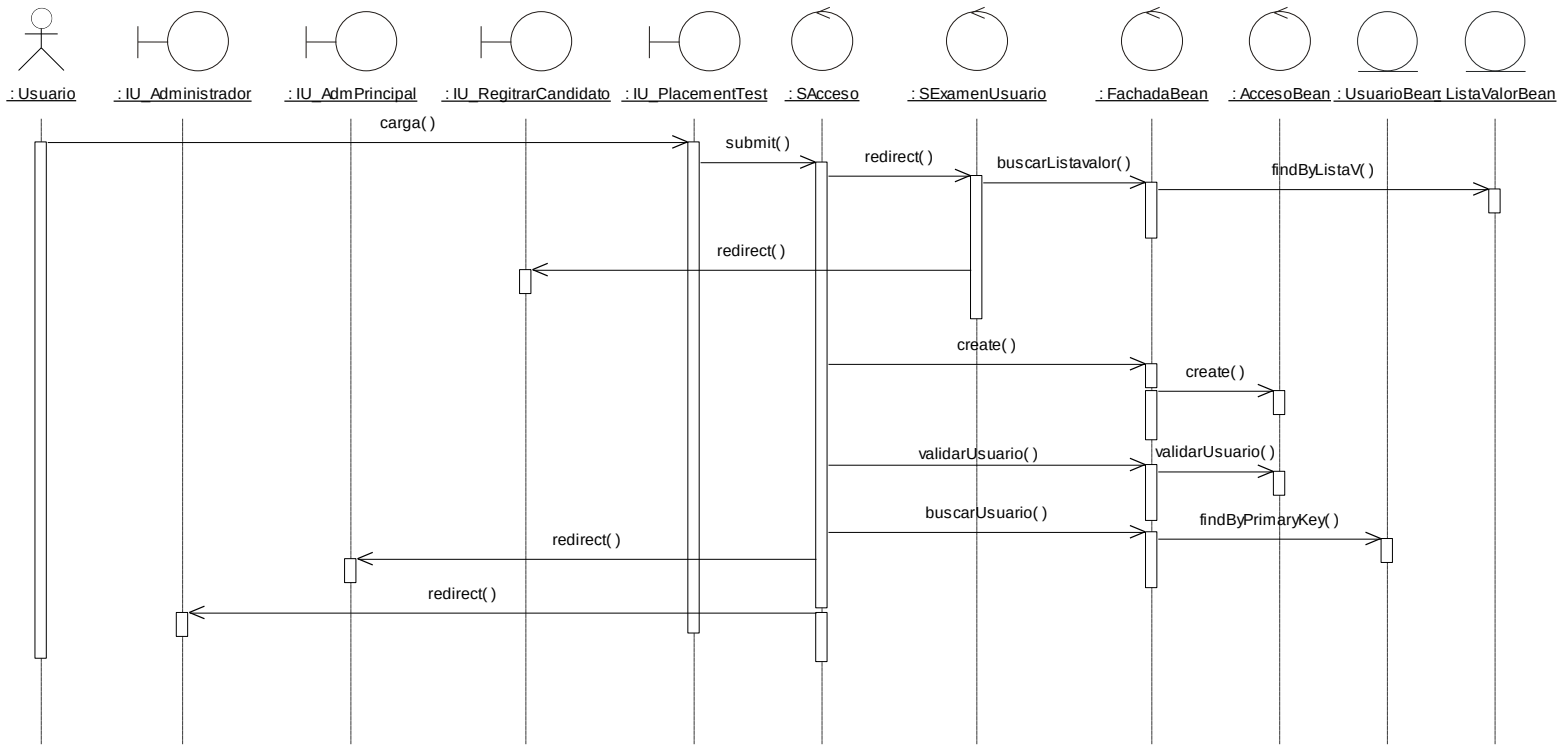
Caso de Uso Gestionar Usuario



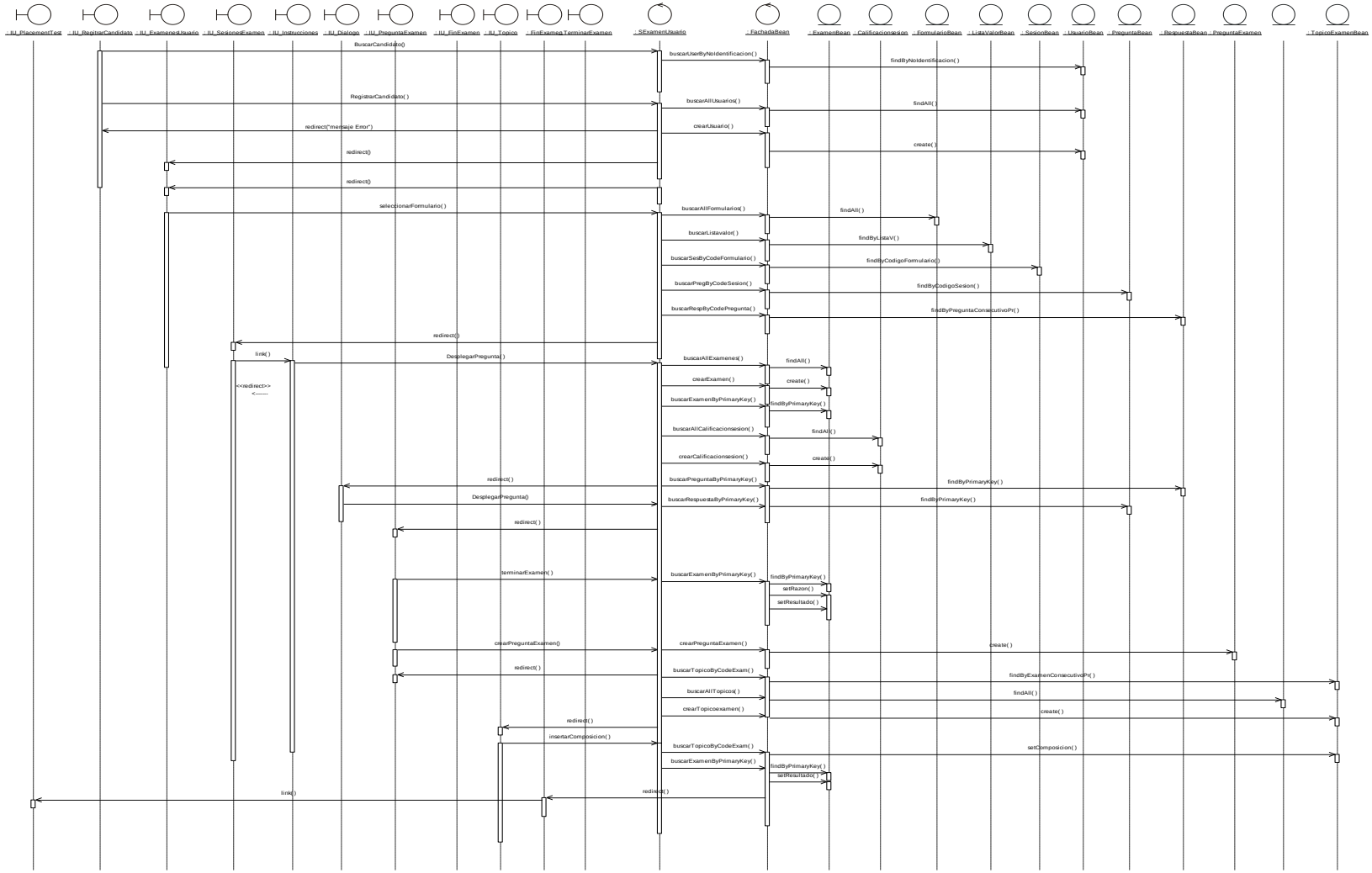
Caso de Uso Gestionar Exámenes



Diagramas de Secuencia_Caso de Uso Acceder al Sistema



Diagramas de Secuencia_Caso de Uso Presentar Placement Test



Diagramas de Secuencia_Caso de Uso Gestionar Usuario

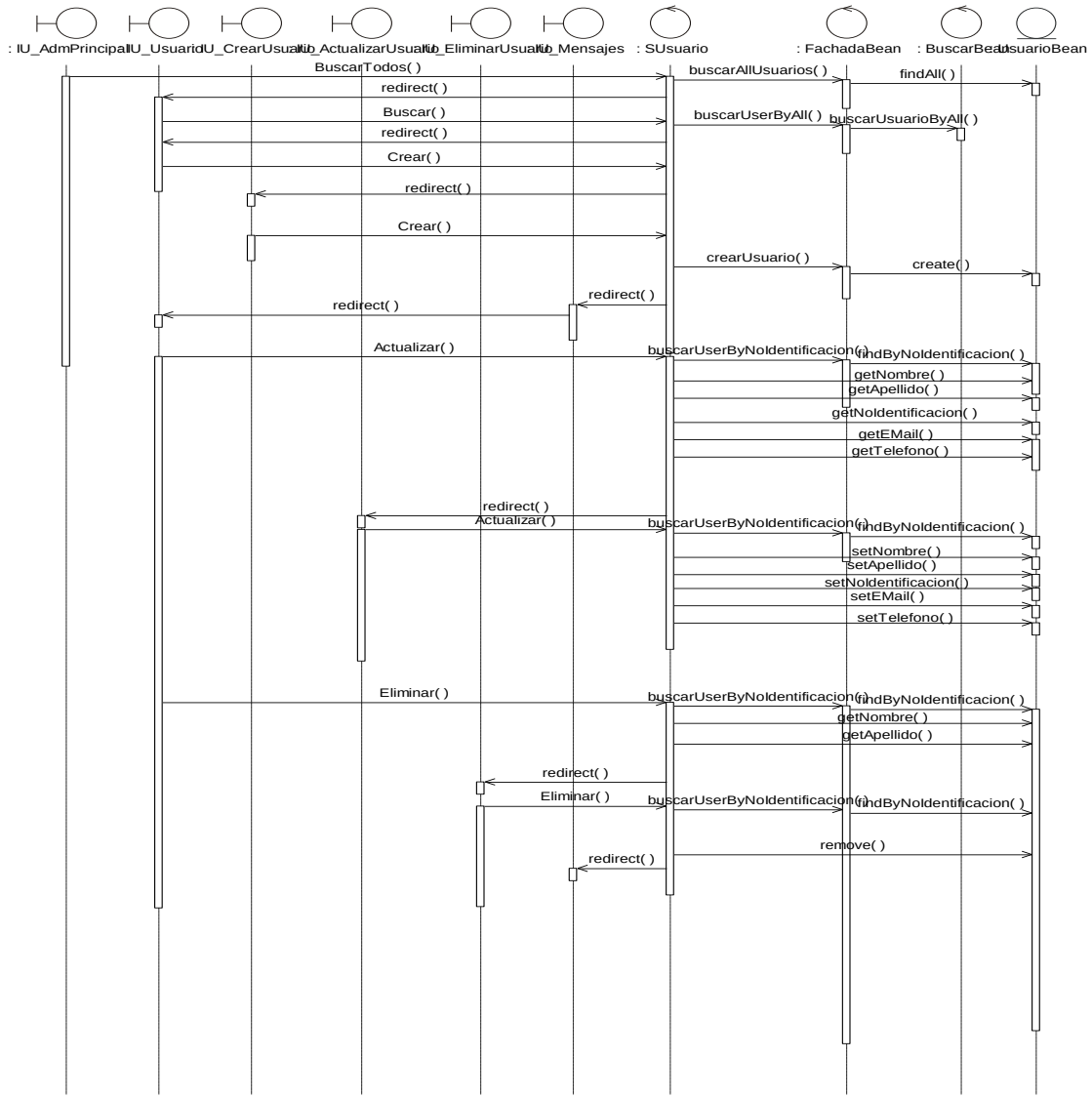


Diagrama de Secuencia_Caso de Uso Gestionar Exámenes Presentados

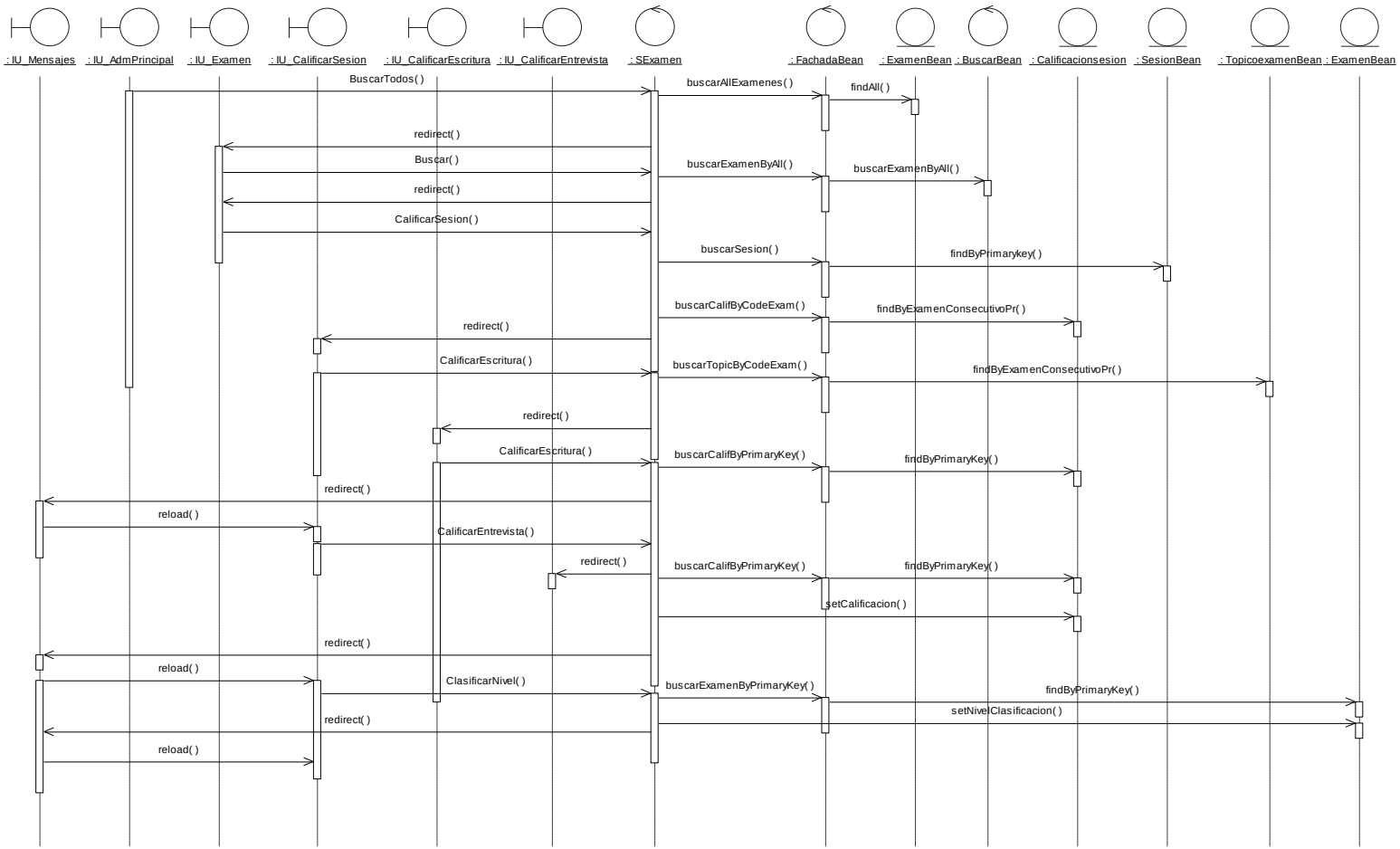


Diagrama de Implantación

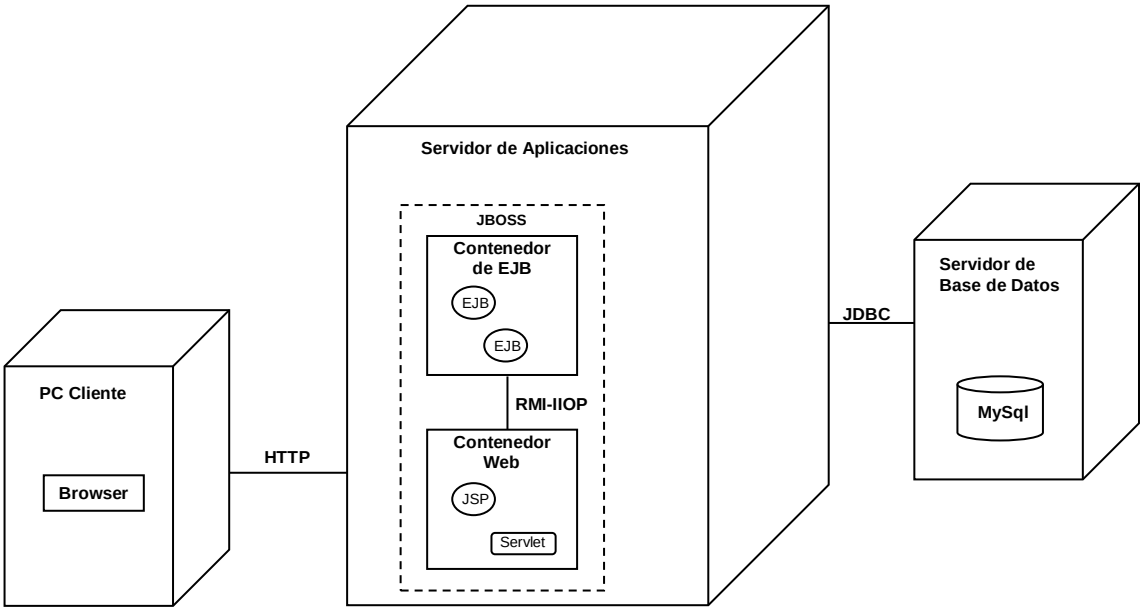


Ilustración 25. Diagrama de Implantación

CONCLUSIONES Y RECOMENDACIONES

1. La necesidad de desarrollar aplicaciones software complejas en periodos de tiempo cada vez más cortos está produciendo importantes cambios en la forma de construirlas, siendo el desarrollo de software basado en componentes (DSBC) uno de los mecanismos más efectivos para conseguir este objetivo. Esta nueva disciplina se basa en componentes software comercial, construido y probado previamente, que se conocen como componentes COTS (*Commercial-Off-The-Shelf*). Esto permite construir una aplicación buscando y ensamblando componentes, desarrollados por terceros y cuyo código no se puede modificar, los cuales al ser combinados adecuadamente, satisfacen los requisitos del sistema, y representan a su vez una gran oportunidad de negocio.

2. Uno de los grandes inconvenientes que se presenta al momento de desarrollar una aplicación con J2EE es el poco conocimiento de esta plataforma en el entorno, por eso se recomienda un orden lógico de desarrollo que incluya los siguientes pasos:
    a. Conocer y entender las necesidades de los clientes.
    b. Conocer los recursos HW y SW con los que se puede contar para el desarrollo de la solución
    c. Conocer el equipo de trabajo.
    d. Evaluar las tecnologías existentes que pueden dar una solución óptima al problema contando con los recursos antes mencionados.
    e. Priorizar las características del sistema para una solución adaptada a la necesidad del cliente (1. flexible, 2. seguro, 3. transaccional, 4. distribuido…etc).
    f. Adoptar una metodología de desarrollo con los roles del equipo de trabajo claramente diferenciados.
    g. Elaborar un plan de trabajo.

h. De las tecnologías evaluadas, probar y comparar las tecnologías, para elegir las que den la mejor solución a los requerimientos del cliente.

3. El aprovechamiento de las facilidades para construir software modificable con un lenguaje orientado a objetos (como Java) encaja dentro de un modelo de desarrollo que asuma gestión de cambios, como es el Proceso Unificado. Esto se debe a su naturaleza iterativa-incremental, que permite la incorporación de los cambios y los nuevos requisitos en cada iteración. Se pueden conseguir diseños suficientemente abiertos y flexibles como para que los cambios introducidos a lo largo del tiempo puedan obtenerse con un esfuerzo razonable.

4. Cuando se desarrolla un sistema que se ajusta dentro de cierto tipo, es muy útil consultar patrones que traten el dominio en el que estamos. Un lenguaje de patrones sirve como referencia conceptual del dominio del problema, ya que éstos parten como solución a un conjunto de casos de uso, e interacciones con actores específicos. Además constituyen también un marco conceptual en el planteamiento de la arquitectura de nuestros sistemas, ya que como la función define a la forma, sintetizan por lo general soluciones arquitectónicas y estructurales bien probadas y muy útiles dentro del tipo de problemas que modelan. Hay patrones que describen soluciones para todo, desde el análisis hasta el diseño y desde la arquitectura hasta la implementación. Además, los patrones existen en diversas áreas de interés y tecnologías.

5. El desarrollo y la implantación fiable de aplicaciones J2EE de gran calidad y alto rendimiento puede ser un reto inalcanzable para los desarrolladores, dada la creciente complejidad tecnológica y el reducido tiempo de que se dispone para ponerlas en producción. Para cumplir

este objetivo es indispensable contar con un buen equipo de trabajo y con los conocimientos necesarios en Tecnologías de la Información.

6. La creciente popularidad de las aplicaciones Web se debe a sus múltiples ventajas entre las cuales podemos citar:
   - Multiplataforma: las aplicaciones pueden ser utilizadas independiente de la plataforma Hardware/Software con que se cuente.
   - Aplicación instantánea: Debido a que todos los usuarios de la aplicación hacen uso de un solo programa que radica en el servidor, no existe la necesidad de ejecutar software cliente y se garantiza el uso de la versión en producción más reciente del sistema.
   - Fácil de integrar con otros sistemas; Debido a que se basa en protocolos estándares, la información manejada por el sistema puede ser accedida son mayor facilidad por otros sistemas.
   - Acceso móvil: El usuario puede acceder a la aplicación con la única restricción de que cuente con un acceso a la red privada de la organización o a Internet, dependiendo de las políticas de dicha organización; puede hacerlo desde una computadora de escritorio, una laptop o desde un PDA; desde su oficina, hogar u otra parte del mundo.

7. El mercado de desarrollo de software plantea la necesidad de contar con medios y herramientas que promuevan y permitan la construcción de aplicaciones empresariales. J2EE fue concebida y diseñada para tal fin, como una plataforma que además de proporcionar especificaciones técnicas descritas por un lenguaje, dispone de las herramientas requeridas para la implementación de productos de software basados en dichas especificaciones. Esquemas como el utilizado en Placement Test se ajustan en este plano, implementando una serie de características y funcionalidades ofrecidas por esta plataforma, logrando la apropiación de tecnologías básicas que dieron solución a un problema en particular.

8. La primera versión desarrollada de la aplicación para la automatización del Placement Test con las funcionalidades que cuenta se puede desarrollar bajo otro esquema menos exigente que el ofrecido por la plataforma J2EE.

9. J2EE sobrepasa los requerimientos iniciales del Placement Test, la plataforma es demasiado robusta y esta pensada para construir sobre ella soluciones mucho más complejas.

10. Dadas las características del DSBC, todo el código que se escribe no se queda solo para una aplicación, puede ser reutilizado en otros contextos.

11. La integridad de las tecnologías que ofrece J2EE vienen de un mismo lugar, no son proyectos completamente independientes como los que se ven en soluciones tipo php , mysql, postgresql, etc.

12. Las construcciones de software en J2EE tienen muy claros los esquemas y formas de trabajo. lo que garantiza éxito en la creación de software en equipo.

REFERENCIAS

1. Alexander C. The Timeless Way of Building. Oxford University Press, 1979.

2. Allamaraju S, Beust C, Davies J, Jewell T, Johnson R, Longshaw A, Nagappan R, O'Connor D, Sarang PG, Toussaint A, Tyagi S, Watson G, Wilcox M, Williamson A. Professional Java Server Programming J2EE. Wrox Press Inc., 2001.

3. Appleton, B. Patterns and Software: Essential Concepts and Terminology. Available Online at http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html. February 14, 2000.

4. Bent K. Patterns Generate Architectures. In Proceedings of the 8th European Conference for Object-Oriented Programming, Bologna, Italy, July 4-8, 1994.

5. Brown, A. (1999). Principles and Practice of Component-based Software. Macmillan Technical Publishing, 1999.

5. Brown, A. W. y Wallnau, K. C. (1998). The Current State of CBSE. IEEE Software, 15(5):37–46, 1998.

5. Ciupke O, Schmidt R. Components as Context-Independent Units of Software. In Proceedings of the 10th European Conference for Object-Oriented Programming, Linz, Austria, July 8-12, 1996.

6. Dean J, Vigder M. System Implementation Using Commercial Off-The-Shelf (COTS) Software. En 9th Annual Software Technology Conference (STC'97), Salt Lake City, Utah, USA. http://seg.iit.nrc.ca/English/abstracts/ NRC40173.html, 1997.

6. D´Souza DF, Wills AC. Objects, Components and Frameworks with UML: The Catalysis Approach. Addison-Wesley, 1998.

7. Fayad ME, Schmidt DC. Object-Oriented Application Frameworks. Communications of the ACM 1997; 40:32-38.

8. Fernández JF. Prueba de Software Basado en Componentes: Estado Actual. Informe Técnico No. 28 Serie Verde, Centro de Investigación en Computación, Instituto Politécnico Nacional, México, 1999.

9. Gamma E, Vlissides J, Johnson R, Helm R. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

10.IBM-WebSphere. Large Grained Components. http://www.ibm.com/software/components, 2001.

10. Kruchten P. Modeling Component Systems with the Unified Modeling Language. In Proceedings of International Workshop on Component Based Engineering, Kyoto, Japan, April 25-26, 1998.

11. Liskov B. Program Development in Java: Abstraction, Specification, and Object-Oriented Design. Pearson Education, 2000.

12. Meyer B. Object-Oriented Software Construction. Prentice Hall, Inc., 1997.

13. Monday P, Carey J, Dangler M. SanFrancisco™ Component Framework: An Introduction. Addison Wesley Professional, 2000.

14. Oberndorf, P. (1997). COTS and Open Systems – An Overview. http://www.sei.cmu.edu/activities/str/descriptions/cots_body.html.

15. OMG. A UML Profile for Enterprise Distributed Object Computing V1.0. Object Management Group. OMG document ad/2001-08-19.

15. Roman E, Ambler SW, Marinescu F, Ambler S, Tyler J. Mastering Enterprise JavaBeans. Wiley, John & Sons, Inc., 2001.

16. Shaw M. Software Architecture: Perspectives on an Emerging Discipline. New Jersey: Prentice Hall, 1996.

17. Siegel J. CORBA 3 Fundamentals and Programming. Jonh Wiley & Sons, 2000.

18. Szyperski C. Components vs. Objects vs. Component Objects. In Proceedings of Object Oriented Programming, Munich, Germany, 1999.

19. Szyperski C, Gruntz D, Murer S. Component Software: Beyond Object-Oriented Programming. Pearson Education, 2002.

20. Szyperski C, Pfister C. Why Objects Are Not Enough. In Proceeding of the First International Component Users Conference, Munich, Germany, July 15-19, 1996.

21. Szyperski, C. Component Software. Beyond Object-Oriented Programming.Addison-Wesley. ISBN: 0-201-17888-5, 1998.

19. Tary Z, Bukhres O. Fundamentals of Distributed Object Systems: The CORBA Perspective. John Wiley & Sons, 2001.

20. Vanhelsuwe L. Mastering Java Beans. Sybex, Inc., 1998.

21. Vinoski, S. CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments. IEEE Communications Magazine No. 2, págs. 46-55.1997.

ANEXOS

A.1. Documentación

Documentación de Servlets y JSP´s

A.2. Documento de análisis completo

Descripción de Casos de Uso alto nivel

Diagrama de Casos de Uso alto nivel

Descripción de Casos de Uso extendidos

Diagrama de Casos de Uso extendidos

Modelo Entidad – Relación

A.3. Documento de Diseño

Diagramas de paquetes de diseño

Diagrama Gral de Clases de Diseño

A.4. Pruebas y Puesta en Servicio de la solución

Posibles extensiones del sistema

Manual del Usuario del Sistema

Manual del Administrador del Sistema

Manual de Mantenimiento del Sistema

Manual de Instalación del Sistema