



Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

**APLICACIÓN DE LA METODOLOGÍA RUP PARA EL
DESARROLLO RÁPIDO DE APLICACIONES BASADO EN EL
ESTÁNDAR J2EE**

JULIO CÉSAR RUEDA CHACÓN

Asesorado por: Ing. José Ricardo Morales Prado

Guatemala, marzo de 2006

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

**APLICACIÓN DE LA METODOLOGÍA RUP PARA EL
DESARROLLO RÁPIDO DE APLICACIONES BASADO EN EL
ESTÁNDAR J2EE**

TRABAJO DE GRADUACIÓN
PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

JULIO CÉSAR RUEDA CHACÓN

Asesorado por: Ing. José Ricardo Morales Prado

AL CONFERÍRSELE EL TÍTULO DE
INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, MARZO DE 2006

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	
VOCAL II	Lic. Amahán Sánchez Álvarez
VOCAL III	Ing. Julio David Galicia Celada
VOCAL IV	Br. Kenneth Issur Estrada Ruiz
VOCAL V	Br. Elisa Yazminda Vides Leiva
SECRETARIA	Inga. Marcia Ivonne Véliz Vargas

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Edgar Estuardo Santos Sutuj
EXAMINADOR	Inga. Floriza Felipa Avila de Medinilla
EXAMINADOR	Inga. Virginia Victoria Tala Ayerdi
SECRETARIA	Inga. Marcia Ivonne Véliz Vargas

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

APLICACIÓN DE LA METODOLOGÍA RUP PARA EL DESARROLLO RÁPIDO DE APLICACIONES BASADO EN EL ESTÁNDAR J2EE,

tema que me fuera asignado por la Coordinación de la Carrera de Ciencias y Sistemas, en febrero de 2004.

Julio César Rueda Chacón

AGRADECIMIENTOS A:

- Dios** Creador de todo lo existente y guía de mi vida, que me da la oportunidad de seguir creciendo mentalmente, y poner siempre a las personas indicadas en el transcurrir de mi vida.
- Mis padres** Rigoberto Rueda, quien me ha brindado todos sus conocimientos desde los inicios de mi vida y lo más importante, el ejemplo de llevar una vida digna de ser un hombre a admirar; padre, estaré siguiendo siempre tus pasos; Eleonora Chacón, quien me ha dado su cariño, atenciones, recuerdos y alegrías desde mi niñez y por estar siempre pendiente de mí, a ambos por el apoyo incondicional que me dieron a lo largo de la carrera y a lo largo de mi vida.
- Mi hermano
Byron** Por sus consejos y apoyo y por los buenos tiempos que hemos vivido, que siempre estarán en mis pensamientos.
- Mi familia** Porque siempre me han apoyado, aconsejado y brindado todo el cariño que ha sido fundamental en mi vida; familia, este logro es de todos.
- Mi asesor** Ingeniero Ricardo Morales, por su excelente asesoría y dirección en mi trabajo de investigación.

- Mis amigos** Que sin duda alguna, sus consejos, experiencias y sobre todo, su apoyo y paciencia, contribuyeron en todos mis éxitos. Muy especialmente a Suan por el apoyo incondicional que me ha brindado.
- La Facultad de Ingeniería** Por el soporte institucional dado para mi formación y por ende al pueblo de Guatemala.
- En general** A todas aquellas personas que de una u otra forma, colaboraron o participaron en mi formación como persona y profesional, hago extensivo mi más sincero agradecimiento.

ACTO QUE DEDICO A:

Mis padres

Rigoberto Rueda Cámara y
Eleonora Chacón Umaña de Rueda

Mis abuelos

José Luis Rueda (papá Chepe)
María Luisa Cámara de Rueda (mamá Güicha)
María Elena Umaña de Chacón (abuelita Elena) y
José María Chacón Villela (abuelito Chema), que Dios lo
tenga en su gloria.

Mi familia

A cada uno de ellos.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES	V
TABLAS	IX
GLOSARIO	XI
RESUMEN	XVII
OBJETIVOS	XIX
INTRODUCCIÓN	XXI
1. METODOLOGÍA DE DESARROLLO APLICADA	1
1.1. Introducción al RUP	1
1.1.1. Dimensiones del RUP	1
1.1.2. Fases	4
1.1.2.1. Planeando las fases	4
1.1.2.2. Esfuerzo respecto de los flujos de trabajo	7
1.1.2.3. Esfuerzo respecto de las fases	8
1.1.3. Iteraciones	9
1.1.3.1. Proceso Iterativo e Incremental	10
1.1.4. Disciplinas	12
1.1.4.1. Modelado del negocio	13
1.1.4.2. Requerimientos	13
1.1.4.3. Análisis y diseño	13
1.1.4.4. Implementación	14
1.1.4.5. Pruebas	14
1.1.4.6. Despliegue	14
1.1.4.7. Gestión y configuración de cambios	15
1.1.4.8. Gestión del proyecto	15

1.1.4.9. Entorno	16
1.1.5. Organización y elementos en RUP	16
1.1.5.1. Actores o roles	17
1.1.5.2. Artefactos	19
1.1.5.2.1. Conjuntos de artefactos	19
1.1.5.2.2. Grado de finalización de artefactos	21
1.2. Introducción al UML	22
1.2.1. Descripción del lenguaje	24
1.2.1.1. Inconvenientes en UML	25
1.2.1.2. Perspectivas de UML	25
1.2.2. Descripción de los diagramas	26
1.3. Metodología del RUP para análisis y diseño	29
1.3.1. Descripción de estereotipos	31
1.3.2. Enlace del RUP con el UML	32
1.3.3. Descripción del método	40
2. TECNOLOGÍA PARA DESARROLLO RÁPIDO DE APLICACIONES	43
2.1. Introducción al RAD	43
2.1.1. Etapas de la metodología RAD	45
2.1.2. Características de la metodología RAD	46
2.1.3. Problemas en la metodología RAD	47
2.2. Introducción a J2EE	47
2.2.1. JSR	48
2.2.2. JCP	48
2.2.3. Ventajas de J2EE	51
2.2.4. Desventajas de J2EE	52
2.2.5. Integración con otros sistemas	53
2.3. Arquitectura de múltiples capas	54
2.3.1. El modelo de desarrollo de J2EE	54
2.3.2. Servidores de aplicaciones	56

2.3.2.1. JBoss	57
2.3.2.2. JOnAS	58
2.3.2.3. <i>OpenEJB</i>	58
2.3.2.4. Ejemplo práctico de servidor de aplicaciones	58
2.4. Enlace del RUP, UML, RAD y J2EE.	59
2.4.1. Diseñador rápido de <i>rational</i> (RRD)	60
2.4.2. Modelando el sistema con RRD	62
3. HERRAMIENTAS A UTILIZAR PARA UN DESARROLLO RÁPIDO DE APLICACIONES	67
3.1. Herramienta XDE	67
3.1.1. Introducción a XDE	67
3.1.2. Características de <i>rational</i> XDE	68
3.2. Herramienta <i>WebSphere</i>	69
3.2.1. Enfoque de soluciones de la herramienta	72
3.3. Pasos para modelar en la herramienta XDE	72
3.3.1. Tipos de entornos	73
3.3.1.1. Entorno de modelado	73
3.3.1.2. Entorno de desarrollo	74
3.3.1.3. Entorno mixto	75
3.3.2. Tipos de modelos	76
3.3.3. Modelado de UML en XDE	77
3.3.3.1. Casos de uso en XDE y sus diagramas	78
3.3.3.2. Diagrama de clases con XDE	79
3.3.3.3. Diagramas de secuencia	85
3.3.3.4. Diagrama de estados con XDE	86
3.3.3.5. Diagrama de actividades con XDE	88
3.3.3.6. Diagramas de componentes con XDE	90
3.4. Generación de código	91
3.5. Publicación de código	93

4. EXPLICACIÓN ASOCIATIVA DE LAS METODOLOGÍAS ESTÁNDARES Y HERRAMIENTAS PARA EL DESARROLLO RÁPIDO DE APLICACIONES	95
4.1. Título del sistema	95
4.2. Descripción general del sistema	95
4.3. Requerimientos a satisfacer	96
4.3.1. Otros requerimientos a satisfacer	98
4.4. <i>Stakeholder</i> y descripciones de usuarios	98
4.5. Ambiente a utilizar	98
4.6. Modelo de casos de uso	99
4.7. Análisis del caso de uso Oferta en Vehículo	100
4.8. Diagrama de clases del análisis de la realización del caso de uso	105
4.9. Diagrama de secuencia del análisis de la realización del caso de uso	111
4.10. Diagrama de colaboración del análisis de la realización del caso de uso	114
4.11. Diseño de la realización del caso de uso	115
4.12. Diagrama de clases del diseño de la realización del caso de uso	118
4.13. Diagrama de secuencia del diseño de la realización del caso de uso	122
4.14. Diagrama de colaboración del diseño de la realización del caso de uso	124
4.15. Arquitectura	125
4.16. Vista de despliegue	127
CONCLUSIONES	128
RECOMENDACIONES	130
BIBLIOGRAFÍA	132

ÍNDICE DE ILUSTRACIONES

FIGURAS

1.	Disciplinas, fases, iteraciones del RUP	2
2.	Fases del RUP	4
3.	Recursos utilizados en las fases del RUP en el tiempo.	6
4.	Ciclo evolutivo en la elaboración de <i>software</i> basado en el RUP	7
5.	Esfuerzo respecto de los flujos de trabajo	8
6.	Esfuerzo respecto de las fases	9
7.	Ciclo de vida Iterativo incremental	10
8.	Enfoque cascada	11
9.	Ciclo de vida de un <i>software</i> con un enfoque iterativo incremental	12
10.	Elementos que conforman el RUP	17
11.	Grado de finalización de artefactos	22
12.	Desarrollo de UML, con sus versiones	24
13.	Relaciones de enlaces entre modelos	27
14.	Diagramas, partes de un modelo	27
15.	Ejemplo de estereotipo	31
16.	Comparación entre diagramas de casos de uso (a) RUP (b) UML	33
17.	Comparación entre diagramas de clases (a) RUP (b) UML	34
18.	Comparación entre diagramas de objetos (a) RUP (b) UML	35
19.	Comparación entre diagramas de estados (a) RUP (b) UML	35
20.	Comparación entre diagramas de actividades (a) RUP (b) UML	36
21.	Diagrama de secuencia	37
22.	Comparación entre diagramas de colaboración (a) RUP (b) UML	38
23.	Diagrama de componentes	38

24.	Comparación entre diagramas de despliegue (a) RUP (b) UML	39
25.	Esquema del JCP	49
26.	Esquema de un modelo mixto de tres y cuatro capas	55
27.	JBoss es el servidor de aplicaciones que está más de moda actualmente	57
28.	Ejemplo de servidor de aplicaciones	59
29.	Unión de las aplicaciones con los diversos niveles mediante el RRD basado en la plataforma RAD	61
30.	Modelando en RRD basado en la plataforma RAD, y en la metodología RUP	64
31.	Entorno de modelado	74
32.	Entorno de desarrollo	75
33.	Entorno mixto	75
34.	Crear modelo en XDE	77
35.	Barra de objetos para crear casos de uso (a) y caso de uso creado en XDE (b)	78
36.	Diagrama de clases en XDE	79
37.	Ventana de propiedades para el diagrama de clases en XDE	80
38.	Creación de atributos (a) y ventana de propiedades para los atributos (b) utilizados en los diagramas de clases en XDE	80
39.	Visibilidad de atributos y métodos de una clase utilizada en un diagrama de clases en XDE	81
40.	Definición de estereotipos para un diagrama de clases en XDE	81
41.	Líneas para unir artefactos utilizadas en un diagrama de clases en XDE	82
42.	Relaciones entre artefactos, utilizadas en un diagrama de clases en XDE	83

43. Restricciones entre artefactos, utilizadas en un diagrama de clases en XDE, (a) vista modelada de la restricción (b) selección de la restricción	83
44. Diagrama de una interfaz utilizada en un diagrama de clases en XDE	84
45. Cualificadores entre artefactos, utilizados en un diagrama de clases en XDE, (a) vista modelada de la cualificación (b) selección de la cualificación	84
46. Diagrama de secuencia (a) y barra de objetos para crear diagramas de secuencia (b)	85
47. Diagrama de estados en XDE	86
48. Forma de crear un estado, utilizado en un diagrama de estados en XDE	87
49. Acciones creadas a un estado, utilizado en un diagrama de estados en XDE	87
50. Diagrama de actividades en XDE	89
51. Modelado de una actividad (a) y forma de crear una actividad utilizada en un diagrama de actividades en XDE	89
52. Diagrama de componentes en XDE	90
53. Selección de <i>AutoSync</i> para generar código automáticamente en XDE	91
54. Selección del estilo de código a generar en XDE	92
55. Publicación de código en XDE	93
56. Caso de uso sistema de subastas en línea de vehículos	101
57. Realización del caso de uso sistema de subastas en línea de vehículos	102
58. Estereotipos	106
59. Diagrama de clases del análisis de la realización del caso de uso	108
60. Diagrama de secuencia del análisis de la realización del caso de uso	113

61. Diagrama de colaboración del análisis de la realización del caso de uso	114
62. Diagrama de clases del diseño de la realización del caso de uso	119
63. Diagrama de secuencia del diseño de la realización del caso de uso	123
64. Diagrama de colaboración del diseño de la realización del caso de uso	124
65. Arquitectura implementada en el sistema de subastas en línea de vehículos	127
66. Vista de despliegue del sistema de subastas en línea de vehículos	127

TABLAS

I. Esfuerzo-horario contra fases del RUP

6

GLOSARIO

AUTOSYNC	Función que realiza la generación de código de forma automática en la herramienta XDE.
B2B	<i>Business to Business</i> . Consiste en negocios electrónicos entre dos empresas.
B2C	<i>Business to Commerce</i> . Consiste en el comercio electrónico entre empresas y clientes.
BOOCH	Juntamente con los métodos <i>Objectory</i> y OMT representan las notaciones bases del surgimiento del lenguaje UML.
CU	<i>Caso de Uso</i> . Es una secuencia de pasos a seguir para la realización de un fin o propósito.
DAO	<i>Data Access Objects</i> . Objeto que permite la conexión para la transferencia de datos.
DHTML	HTML Dinámico que es una mejora de <i>Microsoft</i> de la versión 4.0 de HTML que le permite crear efectos especiales.
E-BUSINESS	Término utilizado para nombrar a los Negocios mediante la <i>Web (Internet)</i> .

EJB	<i>Enterprise JavaBeans</i> . Interfaces de programación de aplicaciones que forman parte del estándar de construcción de aplicaciones empresariales J2EE, proporcionan la posibilidad de usar componentes <i>software</i> transaccionales que residen en el servidor de aplicaciones. Estos componentes de <i>software</i> pueden ser usados desde cualquier programa <i>Java</i> de forma distribuida.
ENTERPRISE JAVA BEANS	Objetos distribuidos, que contienen la lógica de negocio de nuestras aplicaciones y que hacen transparente al programador operaciones como la persistencia, la seguridad, la gestión de transacciones, etc.
FRAMEWORKS	Plantillas predefinidas, que facilitan la programación.
J2EE	<i>Java 2 Enterprise Edition</i> . Plataforma creada por la empresa SUN en el año 1997; es la que ofrece perspectivas de desarrollo para empresas que quieran basar su arquitectura en productos basados en <i>software</i> libre.
JAD	<i>Joint Application Development</i> . Pequeños grupos (hasta 10 personas) de usuarios y analistas que hacen, para en un corto espacio de tiempo, analizar y especificar entradas, procesos y salidas, a través del desarrollo conjunto de un prototipo de desarrollo de <i>software</i> .
JBOSS	Servidor de aplicaciones libres por excelencia y está implementado al 100% en <i>Java</i> .

JCP	Organismo formado por alrededor de 500 empresas, asociaciones y particulares, cuyo objetivo es asegurar la evolución de las plataformas basadas en <i>Java</i> .
JONAS	Servidor de aplicaciones y por sus características, es uno de los proyectos más ambiciosos del mundo del <i>software</i> libre en <i>Java</i> y pronto superará a JBoss en aceptación.
JSR	<i>Java Specification Request</i> . Es un documento creado por una persona o entidad que cree que es necesaria la presencia de una determinada tecnología dentro de las plataformas basadas en <i>Java</i> . Dentro de este documento se relata por qué es necesaria dicha tecnología, por qué no se pueden abordar los problemas que soluciona con las tecnologías existentes.
JSR	Cuando una persona o entidad cree que es necesaria la presencia de una determinada tecnología dentro de las plataformas basadas en <i>Java</i> , lo que hace es crear un JSR y presentarlo para su aprobación.
OBJECTORY	Juntamente con los métodos Booch y OMT representan las notaciones bases del surgimiento del lenguaje UML.
OMG	<i>Object Management Group</i> . Consorcio del cual forman parte las empresas más importantes que se dedican al desarrollo de <i>software</i> .

- OMT** Juntamente con los métodos Booch y Objectory representan las notaciones bases del surgimiento del lenguaje UML.
- OPENEJB** No es un servidor de aplicaciones completo, sino que es un contenedor de EJBs. No se obtiene todo lo que ofrece un servidor (mensajería, contenedor *Web*, etc.), sino que sólo permite utilizar ejes, esto lo hace más ligero.
- RAD** *Desarrollo Rápido de Aplicaciones*. Metodología que permite a las organizaciones desarrollar sistemas estratégicamente importantes, de manera más rápida, reduciendo a la vez los costos de desarrollo y manteniendo la calidad. Existen varias tecnologías que utilizan esta metodología y que intentan reducir el tiempo de desarrollo.
- RRD** *Rational Rapid Developer*. Diseñador Rápido de Rational, herramienta que integra el modelado visual y automatiza la construcción del sistema que permite el diseño, desarrollo y despliegue rápido, en aplicaciones de comercio electrónico y otras.
- RUP** *Rational Unified Process*. Proceso Unificado de Rational, metodología del proceso de ingeniería de *software* que proporciona un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización del desarrollo.

SERVLETS	Módulos <i>java</i> que nos sirven para extender las capacidades de los servidores <i>Web</i> , son programas para los servidores.
SISTEMAS LEGACY	Conectores que proporciona el servidor de aplicaciones, para acceder a otros ficheros o sistemas
STAKEHOLDER	Personas u organizaciones que están directamente envueltas en la elaboración o tomas de decisiones claves acerca de la funcionalidad y propiedades del Sistema.
TOOLBOX	Opciones que provee en un menú una herramienta de desarrollo.
UML	<i>Unified Modeling Language</i> . Lenguaje Unificado de Modelado, notación estándar para el modelado de sistemas <i>software</i> .
WEBSPHERE	Herramienta de desarrollo que ayuda a incrementar la eficiencia operacional, aportando agilidad y escalabilidad
WORKFLOUS	Flujos de trabajo, los cuales son una secuencia de pasos para la culminación de cada disciplina del RUP.
XDE	<i>Extended Development Environment</i> . Herramienta diseñada especialmente para desarrolladores, integrando herramientas de diseño y desarrollo de código en un único ambiente de desarrollo tanto para la plataforma .NET como J2EE.

XML

eXtensible Markup Language. Lenguaje de marcado ampliable o extensible; su objetivo principal es conseguir una página *Web* más semántica; es un estándar para el intercambio de datos entre diversas aplicaciones.

RESUMEN

Las metodologías y estándares utilizados en un desarrollo de *software* nos proporcionan las guías para poder conocer todo el camino a recorrer desde antes de empezar la implementación, con lo cual se asegura la calidad del producto final, así como también el cumplimiento en la entrega del mismo en un tiempo estipulado.

Es de suma importancia elegir la metodología adecuada, así como las herramientas de implementación adecuadas, es por ello que la metodología RUP basada en UML nos proporciona todas las bases para llevar al éxito la elaboración del *software*, para ello la utilización de la herramienta RRD es una de las elecciones más acertadas debido a que se fundamenta en el RUP para el desarrollo rápido de aplicaciones.

Este trabajo consta de cuatro capítulos, los cuales se describen a continuación.

En el capítulo uno se abarcará la explicación de la metodología RUP con sus bases en el UML, las partes que la conforman, su funcionalidad; con lo cual podremos observar la interrelación entre ambos y la importancia de su uso en el desarrollo de aplicaciones.

En el capítulo dos se abarcará lo que es el RAD, con lo cual podemos tener la información necesaria para un desarrollo rápido de aplicaciones, conjuntamente con el RRD para obtener la interrelación entre un Desarrollo Rápido de Aplicaciones utilizando la metodología RUP, por lo tanto, nos vemos en la necesidad de seguir una serie de pasos que estén definidos en forma de estándar para poder aplicar este desarrollo rápido en un lenguaje en particular, con esto nos referimos al J2EE, el cual nos provee esta información, ya que se estará describiendo la funcionalidad del mismo.

Ya teniendo toda esta información, estaremos en la capacidad de tener el conocimiento adecuado para el desarrollo de aplicaciones siguiendo modelos y estándares, por lo tanto, en el capítulo tres estaremos describiendo las herramientas XDE y *WebSphere* para el modelado y elaboración de aplicaciones respectivamente, para poder relacionar el uso que se le puede dar a estas herramientas siguiendo los modelos y estándares definidos en los primeros dos capítulos.

Para terminar se estará describiendo la interrelación entre el estándar J2EE, la metodología RUP con las herramientas para el modelado XDE y la herramienta de desarrollo *WebSphere* utilizando la herramienta RRD basada en RUP y RAD para la interrelación y así lograr un Desarrollo Rápido de Aplicaciones, con lo cual, se estará informando las conectividades que se tienen que dar para que todo esto como partes individuales se puedan mezclar, formando un todo con el cual ser capaces de conseguir elaboración de aplicaciones de forma rápida y que cumplan con la funcionalidad requerida.

OBJETIVOS

- **General**

Explicar cómo se interrelacionan el estándar J2EE y la metodología RUP basada en UML, para el Desarrollo Rápido de Aplicaciones con las herramientas de modelado XDE y desarrollo *WebSphere*, utilizando la herramienta RRD basada en RUP y RAD para la interrelación.

- **Específicos**

1. Describir el funcionamiento de la metodología RUP, el lenguaje UML y el enlace entre ellos.
2. Describir el Desarrollo Rápido de Aplicaciones y su metodología.
3. Describir las características y funcionalidad del estándar J2EE.
4. Describir las características de la herramienta de modelado XDE y la herramienta de desarrollo *WebSphere*.
5. Explicar la Herramienta RRD para utilizarla en un Desarrollo Rápido de Aplicaciones.
6. Conocer la interrelación entre la metodología RUP y la herramienta RRD.

INTRODUCCIÓN

En la actualidad, la utilización de metodologías para el desarrollo de aplicaciones es casi imposible omitirla, debido a la gran necesidad de control de variables que conlleva el mismo desarrollo, y para la ordenada elaboración de las aplicaciones, por lo tanto, seguir metodologías y estándares nos llevan a estar en competitividad en todo momento.

Es de suma importancia conocer el modo como se interrelacionan metodologías con estándares y herramientas siguiendo un único propósito, el cual consiste en la elaboración de aplicaciones de manera eficiente, ordenada y con el menor número de defectos.

La metodología RUP nos proporciona disciplinas en las cuales se encuentran artefactos con lo cual se podrá contar con guías para poder documentar e implementar de una manera fácil y eficiente, todas las guías para un buen desarrollo, todo esto dentro de las respectivas fases con las cuales cuenta.

Al contar con las guías en las cuales nos podremos basar durante todo el desarrollo, se podrá utilizar la herramienta RRD basada en el RUP para poder implementar todo lo prescrito en nuestras guías de una manera segura y sobre todo rápida.

Además, contando con el estándar J2EE se podrá entrelazar la metodología RUP con éste, ya que se ofrece una gran interoperabilidad entre ambos, con lo cual la implementación del *software* utilizando RRD se realizará de una manera mucho más sencilla, ordenada y eficiente.

No es posible realizar un desarrollo de *software* de una manera lenta, ya que las exigencias de los clientes actuales conllevan a verse en la necesidad de implementar soluciones rápidas y que cumplan con los requerimientos planteados, por lo que el Desarrollo Rápido de Aplicaciones es una de las características que más impacto tiene en la actualidad, para solventar esto se deben utilizar herramientas basadas en este nuevo enfoque.

1. METODOLOGÍA DE DESARROLLO APLICADA

1.1. Introducción al RUP

Las siglas RUP en inglés significa *Rational Unified Process* (Proceso Unificado de Rational) es un producto del proceso de ingeniería de *software* que proporciona un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización del desarrollo. Su meta es asegurar la producción del *software* de alta calidad que resuelve las necesidades de los usuarios dentro de un presupuesto y tiempo establecidos.

1.1.1. Dimensiones del RUP

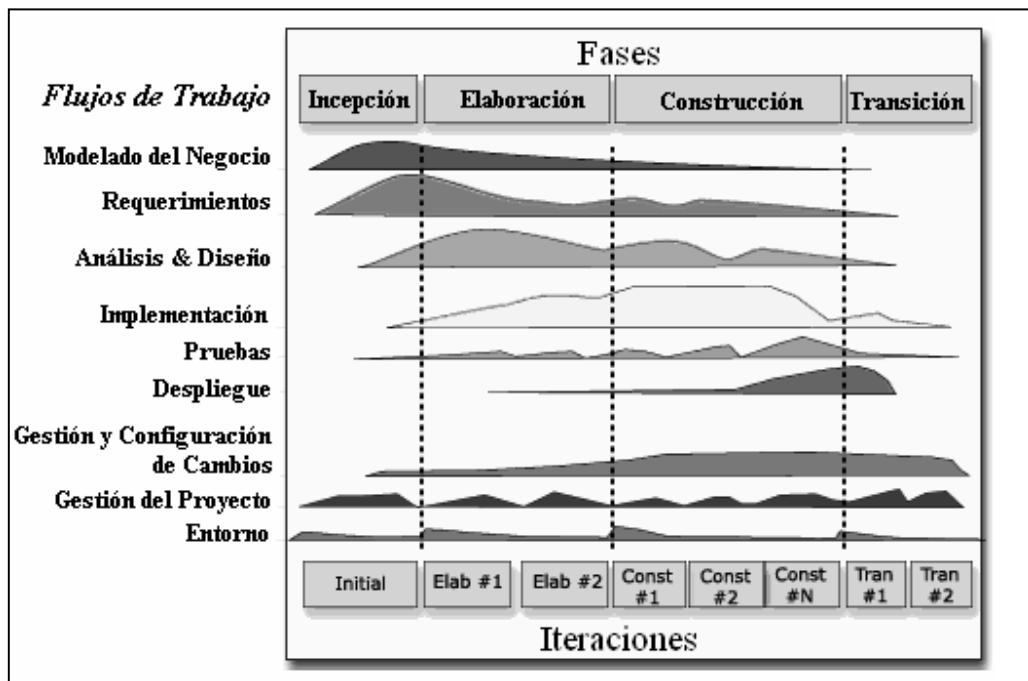
El RUP tiene dos dimensiones:

- El eje horizontal representa tiempo y demuestra los aspectos del ciclo de vida del proceso.
- El eje vertical representa las disciplinas, que agrupan actividades definidas lógicamente por la naturaleza.

La primera dimensión representa el aspecto dinámico del proceso y se expresa en términos de fases, de iteraciones, y la finalización de las fases. La segunda dimensión representa el aspecto estático del proceso: cómo se describe en términos de componentes de proceso, las disciplinas, las actividades, los flujos de trabajo, los artefactos, y los roles.

En la figura 1 se puede observar como varía el énfasis de cada disciplina en un cierto plazo en el tiempo, y durante cada una de las fases. Por ejemplo, en iteraciones tempranas, pasamos más tiempo en requerimientos, y en las últimas iteraciones pasamos más tiempo en poner en práctica la realización del proyecto en si.

Figura 1. Disciplinas, fases, iteraciones del RUP

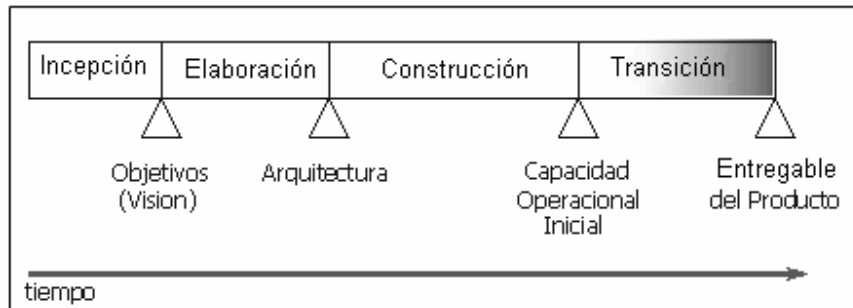


Se puede hacer mención de las tres características esenciales que definen al RUP:

- **Proceso Dirigido por los Casos de Uso:** Con esto se refiere a la utilización de los Casos de Uso para el desenvolvimiento y desarrollo de las disciplinas con los artefactos, roles y actividades necesarias. Los Casos de Uso son la base para la implementación de las fases y disciplinas del RUP. Un Caso de Uso es una secuencia de pasos a seguir para la realización de un fin o propósito, y se relaciona directamente con los requerimientos, ya que un Caso de Uso es la secuencia de pasos que conlleva la realización e implementación de un Requerimiento planteado por el Cliente.
- **Proceso Iterativo e Incremental:** Es el modelo utilizado por RUP para el desarrollo de un proyecto de *software*. Este modelo plantea la implementación del proyecto a realizar en Iteraciones, con lo cual se pueden definir objetivos por cumplir en cada iteración y así poder ir completando todo el proyecto iteración por iteración, con lo cual se tienen varias ventajas, entre ellas se puede mencionar la de tener pequeños avances del proyectos que son entregables al cliente el cual puede probar mientras se esta desarrollando otra iteración del proyecto, con lo cual el proyecto va creciendo hasta completarlo en su totalidad. Este proceso se explica mas adelante a detalle.
- **Proceso Centrado en la Arquitectura:** Define la Arquitectura de un sistema, y una arquitectura ejecutable construida como un prototipo evolutivo. Arquitectura de un sistema es la organización o estructura de sus partes más relevantes. Una arquitectura ejecutable es una implementación parcial del sistema, construida para demostrar algunas funciones y propiedades. RUP establece refinamientos sucesivos de una arquitectura ejecutable, construida como un prototipo evolutivo.

1.1.2. Fases

Figura 2. Fases del RUP



El ciclo de vida del *software* del RUP se descompone en cuatro fases secuenciales (figura 2). En cada extremo de una fase se realiza una evaluación (actividad: Revisión del ciclo de vida de la finalización de fase) para determinar si los objetivos de la fase se han cumplido. Una evaluación satisfactoria permite que el proyecto se mueva a la próxima fase.

1.1.2.1. Planeando las fases

El ciclo de vida consiste en una serie de ciclos, cada uno de los cuales produce una nueva versión del producto, cada ciclo está compuesto por fases y cada una de estas fases está compuesta por un número de iteraciones, estas fases son:

1. Concepción, Inicio o Estudio de oportunidad
 - Define el ámbito y objetivos del proyecto
 - Se define la funcionalidad y capacidades del producto

2. Elaboración

- Tanto la funcionalidad como el dominio del problema se estudian en profundidad
- Se define una arquitectura básica
- Se planifica el proyecto considerando recursos disponibles

3. Construcción

- El producto se desarrolla a través de iteraciones donde cada iteración involucra tareas de análisis, diseño e implementación
- Las fases de estudio y análisis sólo dieron una arquitectura básica que es aquí refinada de manera incremental conforme se construye (se permiten cambios en la estructura)
- Gran parte del trabajo es programación y pruebas
- Se documenta tanto el sistema construido como el manejo del mismo
- Esta fase proporciona un producto construido junto con la documentación

4. Transición

- Se libera el producto y se entrega al usuario para un uso real
- Se incluyen tareas de *marketing*, empaquetado atractivo, instalación, configuración, entrenamiento, soporte, mantenimiento, etc.
- Los manuales de usuario se completan y refinan con la información anterior
- Estas tareas se realizan también en iteraciones

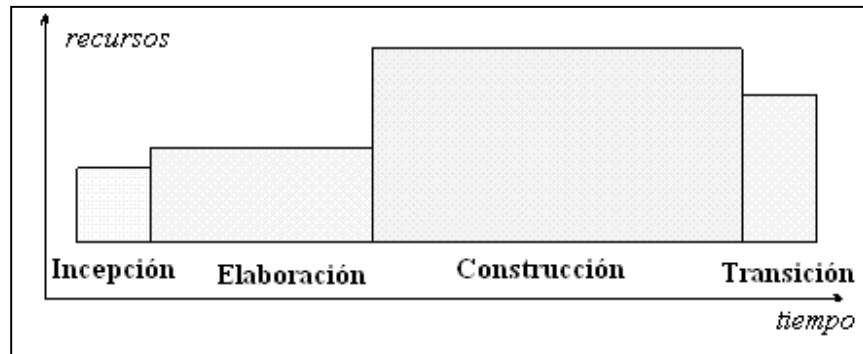
Todas las fases no son idénticas en términos de tiempo y esfuerzo. Aunque esto varía considerablemente dependiendo del proyecto, un ciclo de desarrollo inicial típico para un proyecto de tamaño mediano debe anticipar la distribución siguiente el esfuerzo y horario:

Tabla I. Esfuerzo-horario contra fases del RUP

	<u>Concepción</u>	<u>Elaboración</u>	<u>Construcción</u>	<u>Transición</u>
Esfuerzo	~5 %	20 %	65 %	10%
Horario	10 %	30 %	50 %	10%

Lo cuál se puede representar gráficamente como se muestra en la figura 3:

Figura 3. Recursos utilizados en las fases del RUP en el tiempo.

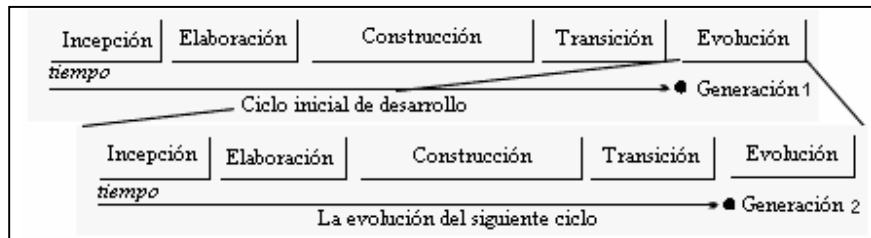


En un ciclo evolutivo, las fases de concepción y elaboración serían considerablemente más pequeñas. Algunas herramientas que pueden automatizar una cierta porción del esfuerzo de la fase de Construcción pueden atenuar esto, haciendo que la fase de construcción sea mucho más pequeña que las fases de concepción y elaboración juntas. Este es precisamente el objetivo del trabajo.

Cada paso con las cuatro fases produce una generación del *software*. A menos que el producto "muera", se desarrollará nuevamente repitiendo la misma secuencia las fases de concepción, elaboración, construcción y transición, pero con diversos énfasis cada fase.

Estos ciclos subsecuentes se llaman los ciclos de la evolución. Mientras que el producto pasa durante varios ciclos, se producen las nuevas generaciones. En la figura 4 se muestra este ciclo evolutivo.

Figura 4. Ciclo evolutivo en la elaboración de *software* basado en el RUP



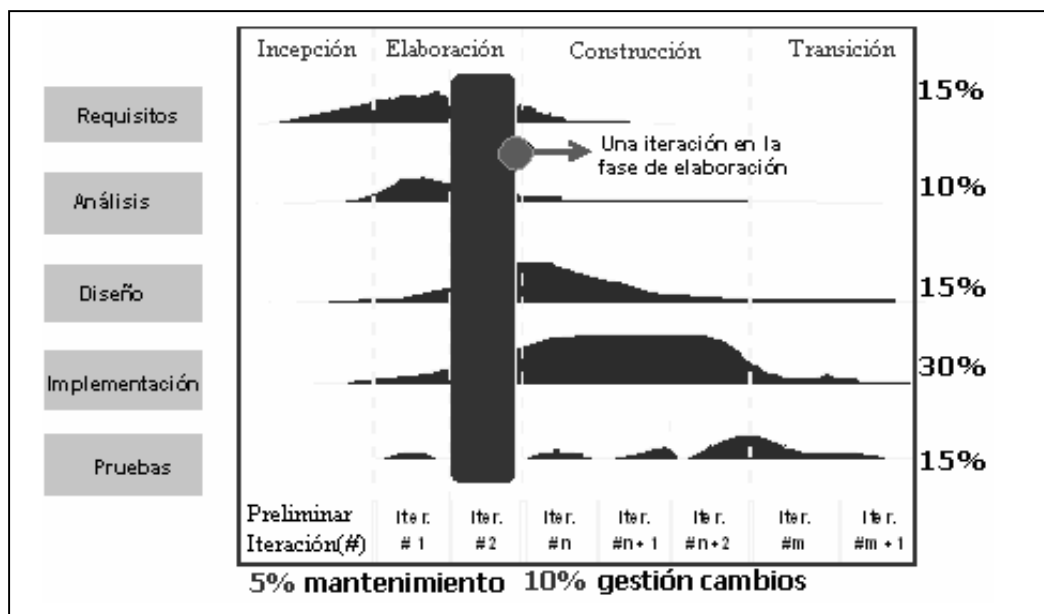
Los ciclos evolutivos pueden ser iniciados por las mejoras sugeridas por el usuario, cambios en el contexto del usuario, cambios en la tecnología subyacente, reacción a la competencia, etcétera. Los ciclos evolutivos tienen típicamente fases de concepción y elaboración mucho más cortas, puesto que la definición y la arquitectura básicas del producto son determinadas por los ciclos de desarrollo anteriores. Las excepciones a esta regla son los ciclos evolutivos en los cuales ocurre o surge un producto significativo o una redefinición arquitectónica.

1.1.2.2. Esfuerzo respecto de los flujos de trabajo

En la figura 5 se muestran ciertos porcentajes, de forma vertical se muestra el esfuerzo que se tiene que realizar por cada una de las disciplinas o flujos de trabajo, y los dos porcentajes que se muestran de forma horizontal son para todo el proyecto.

Explicando mas puntualmente la figura 5 se puede observar que para la obtención de requerimientos o requisitos en la fase de concepción se empiezan a obtener, en la fase de elaboración tiene su auge y va declinando en la fase de construcción, realizar todo esto requiere aproximadamente un 15% de esfuerzo, y así sucesivamente con las demás disciplinas. En esta sección y la siguiente, los porcentajes pueden variar de un proyecto a otro

Figura 5. Esfuerzo respecto de los flujos de trabajo

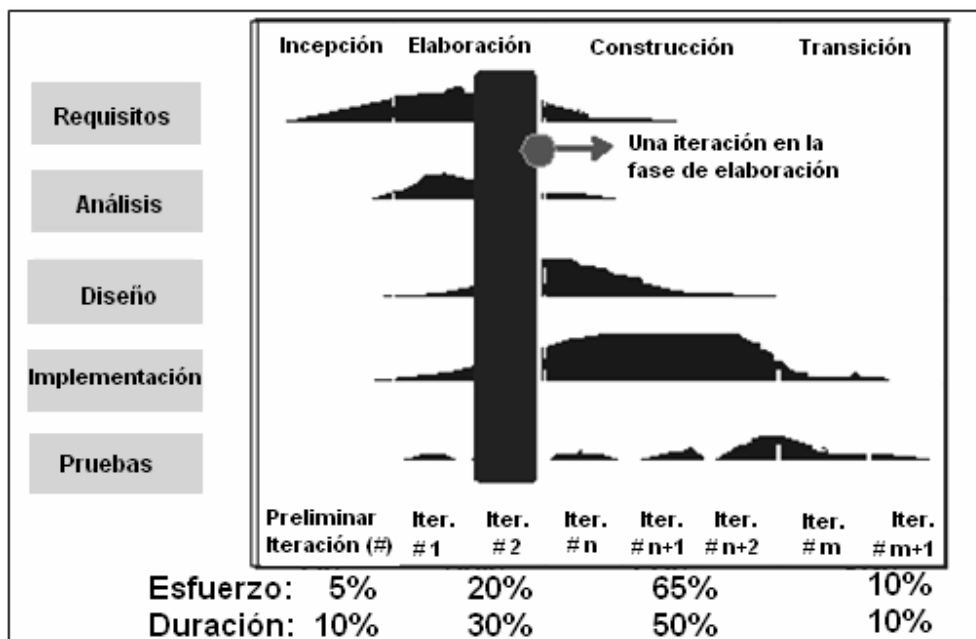


1.1.2.3. Esfuerzo respecto de las fases

En la figura 6 se muestran dos filas de porcentajes, el primero que es el esfuerzo realizado por cada fase en forma general e incluyendo las iteraciones dentro de cada fase; y en la segunda fila, la duración que tiene aproximadamente en porcentajes del tiempo total del proyecto para cada una de las fases incluyendo todas las iteraciones que conlleven realizar cada fase.

Explicando más puntualmente una pequeña parte de la figura 6 se puede observar que para la fase de construcción se tiene que dedicar más esfuerzo y mayor duración, siempre y cuando dependiendo de que disciplina estemos ejecutando, por ejemplo en la disciplina de implementación se tiene mucho auge en la fase de construcción.

Figura 6. Esfuerzo respecto de las fases



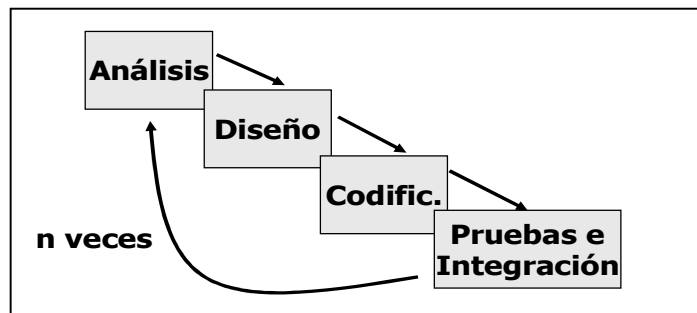
1.1.3. Iteraciones

El RUP maneja el proceso Iterativo Incremental para el desarrollo de las aplicaciones o proyectos, por tal motivo es de suma importancia explicar brevemente en que consiste este proceso.

1.1.3.1. Proceso Iterativo e Incremental

Este proceso se refiere a la realización de un ciclo de vida de un proyecto y se basa en la evolución de prototipos ejecutables que se muestran a los usuarios y clientes. En este ciclo de vida iterativo a cada iteración se reproduce el ciclo de vida en cascada a menor escala, estableciendo los objetivos de una iteración en función de la evaluación de las iteraciones precedentes y las actividades se encadenan en una mini-cascada con un alcance limitado por los objetivos de la iteración. En la figura 7 se muestran los pasos a realizar para seguir el ciclo de vida iterativo incremental, hasta la realización de una fase.

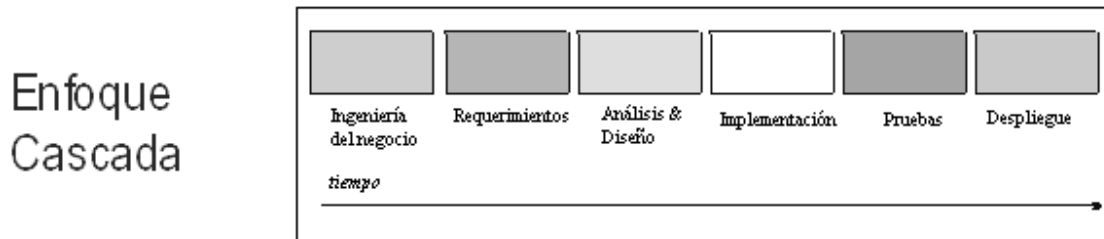
Figura 7. Ciclo de vida Iterativo incremental



Para la realización de cada iteración se tiene que tomar en cuenta la planificación de la iteración, estudiando los riesgos que conlleva su realización, también incluye el análisis de los casos de uso y escenarios, el diseño de opciones arquitectónicas, la codificación y pruebas, la integración gradual durante la construcción del nuevo código con el existente de iteraciones anteriores, la evaluación de la entrega ejecutable (evaluación del prototipo en función de las pruebas y de los criterios definidos) y la preparación de la entrega (documentación e instalación del prototipo). Algunos de estos elementos no se realizan en todas las fases.

A continuación se presenta una comparación entre 2 enfoques de un ciclo de vida del desarrollo de *software*, el primero consiste en el ciclo común, el de Cascada (figura 8), en el cual cada disciplina se realiza al finalizar su predecesora y solo al finalizar la nueva se empieza la sucesora y así hasta terminar con las disciplinas necesarias.

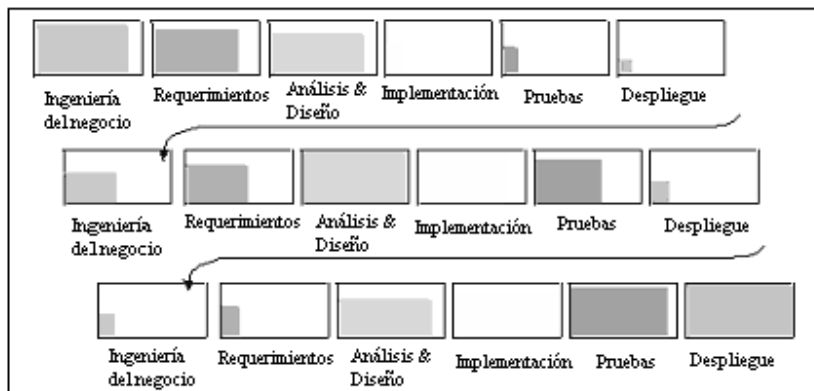
Figura 8. Enfoque cascada



En la figura 9 se muestra el ciclo de vida de un *software* siguiendo el enfoque Iterativo Incremental (utilizado por el RUP), en el cual se puede observar que en cada iteración se realiza una pequeña parte de cada disciplina en paralelo, aumentando así poco a poco hasta concluir con la realización de todas las disciplinas con un número de iteraciones prudente. En la gráfica siguiente se habla de ingeniería del negocio y en la siguiente sección de modelado del negocio, es necesario conservar la consistencia de esto en todo el trabajo, una u otra.

Figura 9. Ciclo de vida de un *software* con un enfoque iterativo incremental

Enfoque Iterativo e Incremental



1.1.4. Disciplinas

Las disciplinas conllevan los flujos de trabajo, los cuales son una secuencia de pasos para la culminación de cada disciplina, estas disciplinas se dividen en dos grupos: las primarias y las de apoyo. Las primarias son las necesarias para la realización de un proyecto de *software*, aunque para proyectos no muy grandes se pueden omitir algunas; entre ellas se tienen: Modelado del Negocio, Requerimientos, Análisis y Diseño, Implementación, Pruebas, Despliegue. Las de apoyo son las que como su nombre lo indica sirven de apoyo a las primarias y especifican otras características en la realización de un proyecto de *software*; entre estas se tienen: Entorno, Gestión del Proyecto, Gestión de Configuración y Cambios. A continuación se describe rápidamente cada una de estas disciplinas.

1.1.4.1. Modelado del negocio

Esta disciplina tiene como objetivos comprender la estructura y la dinámica de la organización, comprender problemas actuales e identificar posibles mejoras, comprender los procesos de negocio. Utiliza el Modelo de CU del Negocio para describir los procesos del negocio y los clientes, el Modelo de Objetos del Negocio para describir cada CU del Negocio con los Trabajadores, además utilizan los Diagramas de Actividad y de Clases.

1.1.4.2. Requerimientos

Esta disciplina tiene como objetivos establecer lo que el sistema debe hacer (Especificar Requisitos), definir los límites del sistema, y una interfaz de usuario, realizar una estimación del costo y tiempo de desarrollo. Utiliza el Modelo de CU para modelar el Sistema que comprenden los CU, Actores y Relaciones, además utiliza los diagramas de Estados de cada CU y las especificaciones suplementarias.

1.1.4.3. Análisis y diseño

Esta disciplina define la arquitectura del sistema y tiene como objetivos trasladar requisitos en especificaciones de implementación, al decir análisis se refiere a transformar CU en clases, y al decir diseño se refiere a refinar el análisis para poder implementar los diagramas de clases de análisis de cada CU, los diagramas de colaboración de de cada CU, el de clases de diseño de cada CU, el de secuencia de diseño de CU, el de estados de las clases, el modelo de despliegue de la arquitectura.

1.1.4.4. Implementación

Esta disciplina tiene como objetivos implementar las clases de diseño como componentes (ej. fichero fuente), asignar los componentes a los nodos, probar los componentes individualmente, integrar los componentes en un sistema ejecutable (enfoque incremental). Utiliza el Modelo de Implementación, conjuntamente los Diagramas de Componentes para comprender cómo se organizan los Componentes y dependen unos de otros.

1.1.4.5. Pruebas

Esta disciplina tiene como objetivos verificar la integración de los componentes (prueba de integración), verificar que todos los requisitos han sido implementados (pruebas del sistema), asegurar que los defectos detectados han sido resueltos antes de la distribución

1.1.4.6. Despliegue

Esta disciplina tiene como objetivos asegurar que el producto está preparado para el cliente, proceder a su entrega y recepción por el cliente. En esta disciplina se realizan las actividades de probar el *software* en su entorno final (Prueba Beta), empaquetarlo, distribuirlo e instalarlo, así como la tarea de enseñar al usuario.

1.1.4.7. Gestión y configuración de cambios

Es esencial para controlar el número de artefactos producidos por la cantidad de personal que trabajan en un proyecto conjuntamente. Los controles sobre los cambios son de mucha ayuda ya que evitan confusiones costosas como la compostura de algo que ya se había arreglado etc., y aseguran que los resultados de los artefactos no entren en conflicto con algunos de los siguientes tipos de problemas:

- Actualización simultánea: Es la actualización de algo elaborado con anterioridad, sin saber que alguien más lo está actualizando.
- Notificación limitada: Al realizar alguna modificación, no se deja información sobre lo que se hizo, por lo tanto no se sabe quien, como, y cuando se hizo.
- Versiones múltiples: No saber con exactitud, cual es la última versión, y al final no se tiene un orden sobre que modificaciones se han realizado a las diversas versiones.

1.1.4.8. Gestión del proyecto

Su objetivo es equilibrar los objetivos competitivos, administrar el riesgo, y superar restricciones para entregar un producto que satisface las necesidades de ambos clientes con éxito (los que pagan el dinero) y los usuarios. Con la Gestión del Proyecto se logra una mejoría en el manejo de una entrega exitoso de *software*. En resumen su propósito consiste en proveer pautas para:

- Administrar proyectos de *software* intensivos.
- Planear, dirigir personal, ejecutar acciones y supervisar proyectos.
- Administrar el riesgo.

Sin embargo, esta disciplina no intenta cubrir todos los aspectos de dirección del proyecto. Por ejemplo, no cubre problemas como:

- Administración de personal: contratando, entrenando, enseñando.
- Administración del presupuesto: definiendo, asignando.
- Administración de los contratos con proveedores y clientes.

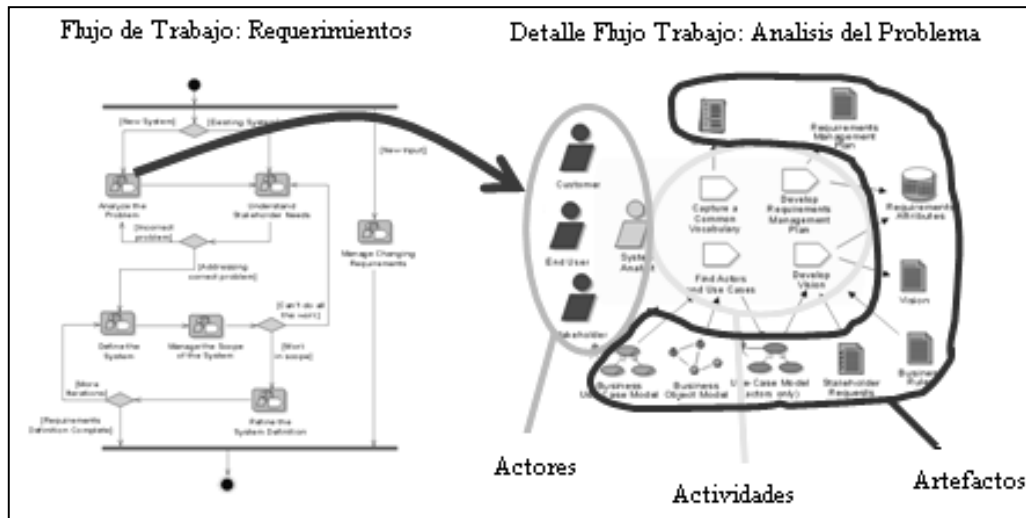
1.1.4.9. Entorno

Esta disciplina se enfoca sobre las actividades necesarias para configurar el proceso que engloba el desarrollo de un proyecto y describe las actividades requeridas para el desarrollo de las pautas que apoyan un proyecto. Su propósito es proveer a la organización que desarrollará el *software*, un ambiente en el cual basarse, el cual provee procesos y herramientas para poder desarrollar el *software*.

1.1.5. Organización y elementos en RUP

Ya conociendo varias partes del RUP nos concentraremos ahora en los elementos que lo componen, entre estos se tienen: Flujos de Trabajo, Detalle de los Flujos de Trabajo, Actores, Actividades y Artefactos. En la figura 10 se muestran mas claramente como se representan gráficamente cada uno de estos elementos y la interrelación entre ellos. Se puede observar que el Flujo de Trabajo de Requerimientos conlleva varios pasos, cada uno de estos pasos tiene asociado uno o varios actores, los cuales a su vez son los encargados de la ejecución de varias actividades, las cuales a la vez están definidas en artefactos o guías para su realización.

Figura 10. Elementos que conforman el RUP



1.1.5.1. Actores o roles

Son los personajes encargados de la realización de las actividades definidas dentro de los flujos de trabajo de cada una de las disciplinas del RUP, estos actores se dividen en varias categorías: Analistas, Desarrolladores, Probadores, Encargados, Otros actores. A continuación se presenta una lista de actores de acorde a las categorías mencionadas con anterioridad:

Analistas

- Analista del Proceso del Negocio
- Diseñador del Negocio
- Revisor del Modelo del Negocio
- Revisor de Requerimientos
- Analista del Sistema
- Especificador de Casos de Uso
- Diseñador de Interfaz del Usuario

Desarrolladores

- Arquitecto
- Revisor de la Arquitectura
- Diseñador de Cápsulas
- Revisor del Código y Revisor del Diseño
- Diseñador de la Base de Datos
- Diseñador
- Implementador y un Integrador

Probadores Profesionales

- Diseñador de Pruebas
- Probador

Encargados

- Encargado de Control del Cambio
- Encargado de la Configuración
- Encargado del Despliegue
- Ingeniero de Procesos
- Encargado de Proyecto
- Revisor de Proyecto

Otros

- Cualquier trabajador
- Artista Gráfico
- *Stakeholder*
- Administrador del Sistema
- Escritor técnico
- Especialista de Herramientas

1.1.5.2. Artefactos

Los artefactos son el resultado parcial o final que es producido y usado por los actores durante el proyecto. Son las entradas y salidas de las actividades, realizadas por los actores, los cuales utilizan y van produciendo estos artefactos para tener guías. Un artefacto puede ser un documento, un modelo o un elemento de modelo.

1.1.5.2.1. Conjuntos de artefactos

Se tiene un conjunto de artefactos definidos en cada una de las disciplinas y utilizadas dentro de ellas por lo actores para la realización de las mismas, a continuación se definen cada una de estas categorías o grupos de artefactos dentro de las disciplinas del RUP:

a) Modelado del negocio

Los artefactos del modelado del negocio capturan y presentan el contexto del negocio del sistema. Los artefactos del modelado del negocio sirven como entrada y como referencia para los requisitos del sistema.

b) Requerimientos

Los artefactos de requerimientos del sistema capturan y presentan la información usada en definir las capacidades requeridas del sistema.

c) Análisis y diseño del sistema

Los artefactos para el análisis y del diseño capturan y presenta la información relacionada con la solución a los problemas se presentaron en los requisitos fijados.

d) Implementación

Los artefactos para la implementación capturan y presentan la realización de la solución presentada en el análisis y diseño del sistema.

e) Pruebas

Los artefactos desarrollados como productos de las actividades de prueba y de la evaluación son agrupadas por el actor responsable, con el cual se lleva un conjunto de documentos de información sobre las pruebas realizadas y las metodologías de pruebas aplicadas.

f) Despliegue

Los artefactos del despliegue capturan y presentan la información relacionada con la transitividad del sistema, presentada en la implementación en el ambiente de la producción.

g) Administración del proyecto

El conjunto de artefactos de la administración del proyecto capturan los artefactos asociados con el proyecto, el planeamiento y a la ejecución del proceso.

h) Administración de cambios y configuración

Los artefactos de la configuración y administración del cambio capturan y presentan la información relacionada con la disciplina de configuración y administración del cambio.

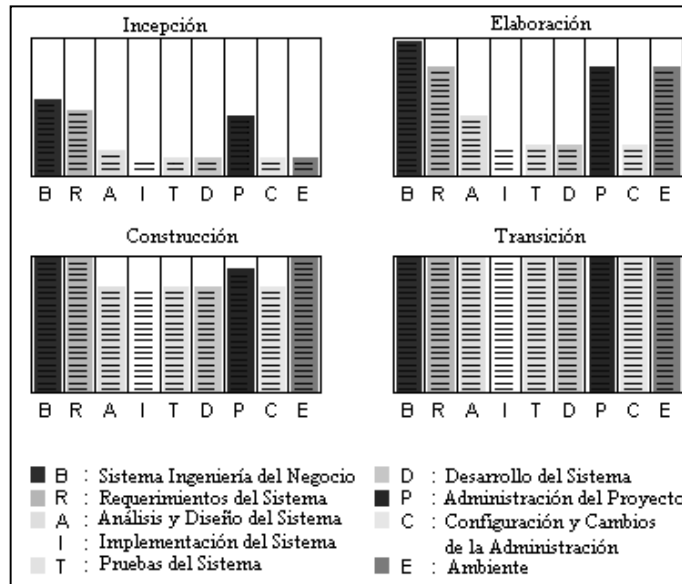
i) Entorno o ambiente

El conjunto de artefactos del ambiente presentan los artefactos que se utilizan como dirección a través del desarrollo del sistema para asegurar la consistencia de todos los artefactos producidos.

1.1.5.2.2. Grado de finalización de artefactos

Consiste en cuanto hemos finalizado del artefacto propuesto, con esto nos referimos por ejemplo, si definimos que vamos a utilizar un artefacto, este nos dice los lineamientos que necesita para ser completado, por lo tanto con grado de finalización nos referimos a cuantos de esos lineamientos del artefacto hemos completado o llenado, esto en cada una de las disciplinas, de acorde a la fase en que se encuentre, para entender de mejor manera lo antes dicho se muestra la figura 11, en la cual se puede observar que en la fase de Concepción, en la disciplina de Implementación del Sistema los artefactos tienen una poca finalización y van aumentando progresivamente en cada fase hasta llegar a su culminación en la fase de Transición, en la disciplina de Ingeniería del Negocio los artefactos tienen una media finalización y sucede lo mismo que con los artefactos de la disciplina anterior los cuales finalizan también en la fase de Transición.

Figura 11. Grado de finalización de artefactos



Con esto se puede mostrar el aumento progresivo de los artefactos en cada disciplina en la fase dada, teniendo una idea un poco más amplia sobre el desenvolvimiento del proyecto hablando en términos de sus artefactos.

1.2. Introducción al UML

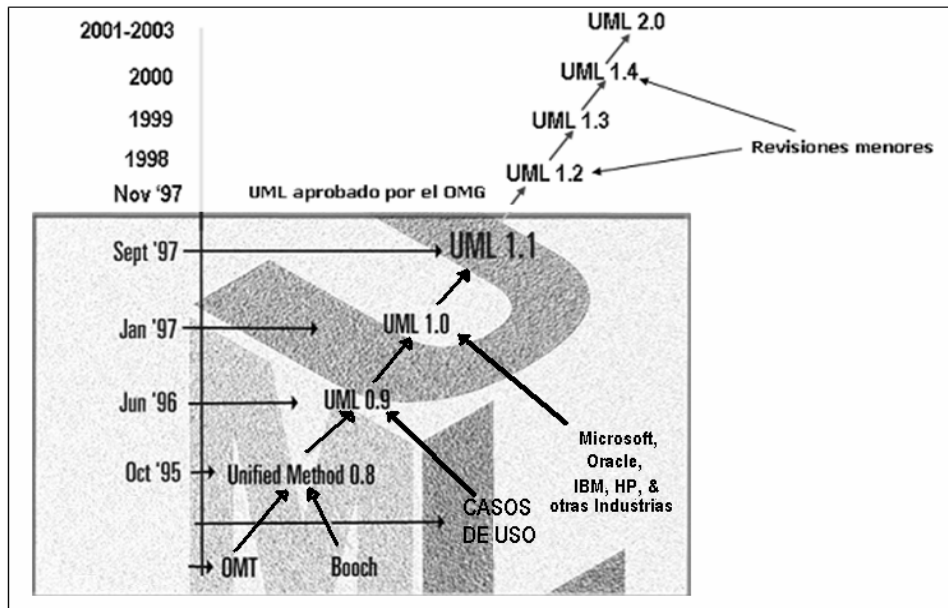
UML surge como respuesta al problema de contar con un lenguaje estándar para escribir planos de *software*. Muchas personas han creído ver UML como solución para todos los problemas sin saber en muchos casos de lo que se trataba en realidad.

El Lenguaje Unificado de Modelado, UML es una notación estándar para el modelado de sistemas *software*, resultado de una propuesta de estandarización promovida por el consorcio OMG (*Object Management Group*), del cual forman parte las empresas más importantes que se dedican al desarrollo de *software*, en 1996.

UML representa la unificación de las notaciones de los métodos Booch, *Objectory* (Ivar Jacobson) y OMT (James Rumbaugh) siendo su sucesor directo y compatible. Igualmente, UML incorpora ideas de otros metodólogos entre los que se pueden incluir a Peter Coad, Derek Coleman, Ward Cunningham, David Harel, Richard Helm, Ralph Johnson, Stephen Mellor, Bertrand Meyer, Jim Odell, Kenny Rubin, Sally Shlaer, John Vlissides, Paul Ward, Rebecca Wirfs-Brock y Ed Yourdon.

En Septiembre de 2001 se ha publicada la especificación de la versión 1.4. Es importante recalcar que sólo se trata de una notación, es decir, de una serie de reglas y recomendaciones para representar modelos. UML no es un proceso de desarrollo, es decir, no describe los pasos sistemáticos a seguir para desarrollar *software*. UML sólo permite documentar y especificar los elementos creados mediante un lenguaje común describiendo modelos. En la figura 12 se puede observar el desarrollo de UML y sus versiones en los años dados, sufriendo revisiones menores, y ciertos participantes en las diversas versiones.

Figura 12. Desarrollo de UML, con sus versiones



1.2.1. Descripción del lenguaje

UML es un lenguaje de propósito general para el modelado orientado a objetos, que combina notaciones provenientes desde: Modelado Orientado a Objetos, Modelado de Datos, Modelado de Componentes, Modelado de Flujos de Trabajo (*Workflows*).

En todos los ámbitos de la ingeniería se construyen modelos, en realidad, simplificaciones de la realidad, para comprender mejor el sistema que vamos a desarrollar: los arquitectos utilizan y construyen planos (modelos) de los edificios, los grandes diseñadores de coches preparan modelos en sistemas existentes con todos los detalles y los ingenieros de *software* deberían igualmente construir modelos de los sistemas *software*.

Un enfoque sistemático permite construir estos modelos de una forma consistente demostrando su utilidad en sistemas de cierto tamaño. Cuando se trata de un programa de cincuenta, cien líneas, la utilidad del modelado parece discutible pero cuando involucramos a cientos de desarrolladores trabajando y compartiendo información, el uso de modelos y el proporcionar información sobre las decisiones tomadas, es vital no sólo durante el desarrollo del proyecto, sino una vez finalizado éste, cuando se requiere algún cambio en el sistema. En realidad, incluso en el proyecto más simple los desarrolladores hacen algo de modelado, si bien informalmente.

Para la construcción de modelos, hay que centrarse en los detalles relevantes mientras se ignoran los demás, por lo cual con un único modelo no tenemos bastante.

1.2.1.1. Inconvenientes en UML

Como todo en el desarrollo de software UML presenta ciertos inconvenientes, entre los cuales se pueden mencionar: Falta integración con respecto de otras técnicas tales como patrones de diseño, interfaces de usuario, documentación, etc., los ejemplos aislados, el monopolio de conceptos, técnicas y métodos en torno a UML.

1.2.1.2. Perspectivas de UML

También se prevé varias perspectivas de UML ya que por ser un lenguaje de propósito general será un lenguaje de modelado orientado a objetos estándar predominante los próximos años, esto se basa en las siguientes razones:

- Participación de metodólogos influyentes
- Participación de importantes empresas
- Aceptación del OMG como notación estándar

También se muestran las siguientes evidencias que apoyan lo antes dicho:

- Herramientas que proveen la notación UML
- “Edición” de libros
- Congresos, cursos, “camisetas”, etc.

1.2.2. Descripción de los diagramas

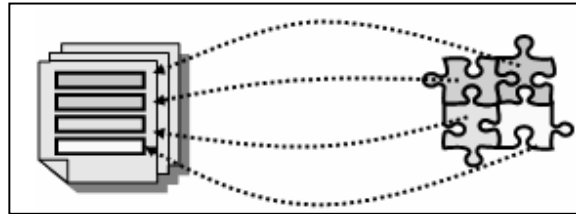
Un modelo captura una vista de un sistema del mundo real. Es una abstracción de dicho sistema, considerando un cierto propósito. Así, el modelo describe completamente aquellos aspectos del sistema que son relevantes al propósito del modelo, y a un apropiado nivel de detalle.

Un diagrama es una representación gráfica de una colección de elementos de modelado, a menudo dibujada como un grafo con vértices conectados por arcos

Un proceso de desarrollo de *software* debe ofrecer un conjunto de modelos que permitan expresar el producto desde cada una de las perspectivas de interés. Es aquí donde se hace evidente la importancia de UML en el contexto de un proceso de desarrollo de *software*.

El código fuente del sistema es el modelo más detallado del sistema (y además es ejecutable). Sin embargo, se requieren otros modelos.

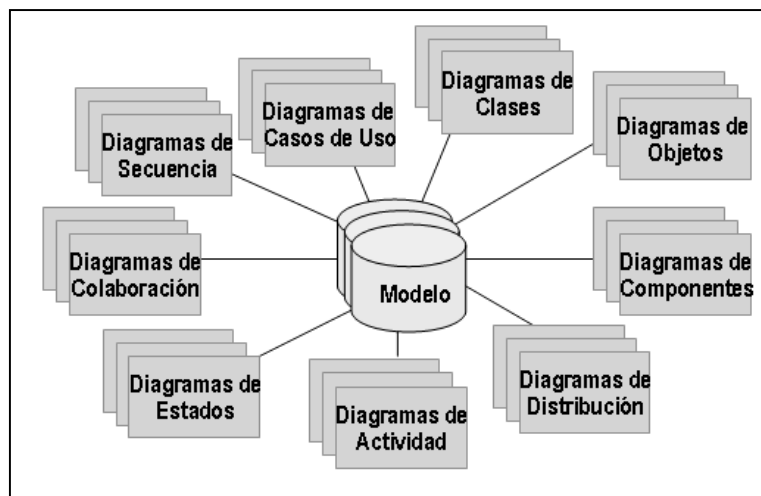
Figura 13. Relaciones de enlaces entre modelos



Cada modelo es completo desde su punto de vista del sistema, sin embargo, existen relaciones de enlaces entre los diferentes modelos (figura 13).

Varios modelos aportan diferentes vistas de un sistema los cuales nos ayudan a comprenderlo desde varios frentes. Así, UML recomienda la utilización de nueve diagramas que, para representar las distintas vistas de un sistema. Estos diagramas de UML se presentan en la figura 14 y se describen a continuación:

Figura 14. Diagramas, partes de un modelo



- a) **Diagrama de Casos de Uso:** modela la funcionalidad del sistema agrupándola en descripciones de acciones ejecutadas por un sistema para obtener un resultado.
- b) **Diagrama de Clases:** muestra las clases (descripciones de objetos que comparten características comunes) que componen el sistema y cómo se relacionan entre sí.
- c) **Diagrama de Objetos:** muestra una serie de objetos (instancias de las clases) y sus relaciones.
- d) **Diagramas de Comportamiento:** dentro de estos diagramas se encuentran:
- **Diagrama de Estados:** modela el comportamiento del sistema de acuerdo con eventos.
 - **Diagrama de Actividades:** simplifica el Diagrama de Estados modelando el comportamiento mediante flujos de actividades. También se pueden utilizar caminos verticales para mostrar los responsables de cada actividad.
 - **Diagramas de Interacción:** Estos diagramas a su vez se dividen en 2 tipos de diagramas, según la interacción que enfatizan:
 - **Diagrama de Secuencia:** enfatiza la interacción entre los objetos y los mensajes que intercambian entre sí junto con el orden temporal de los mismos.

- **Diagrama de Colaboración:** igualmente, muestra la interacción entre los objetos resaltando la organización estructural de los objetos en lugar del orden de los mensajes intercambiados.

e) **Diagramas de implementación**

- **Diagrama de Componentes:** muestra la organización y las dependencias entre un conjunto de componentes.
- **Diagrama de Despliegue:** muestra los dispositivos que se encuentran en un sistema y su distribución en el mismo.

1.3. Metodología del RUP para análisis y diseño

El RUP propone la utilización de los modelos para la implementación completa de todas sus fases respectivamente con sus disciplinas:

- *Modelo de Casos de Uso del Negocio:* Describe la realización del Caso de Uso, es realizado en la disciplina de Modelado del Negocio.
- *Modelo de Objetos del Negocio:* Se utiliza para identificar roles dentro de la organización, es realizado en la disciplina de Modelado del Negocio.
- *Modelo de Casos de Uso:* Muestra las interrelaciones entre el sistema y su ambiente, además sirve como un contrato entre el cliente y los diseñadores. Es considerado esencial al iniciar las actividades de análisis, diseño y prueba; este modelo es realizado en la disciplina de Requerimientos.

- *Modelo de Análisis*: Contiene los resultados del análisis del Caso de Uso, y contienen instancias del artefacto de Análisis de Clases; es realizado en la disciplina de Análisis y Diseño.
- *Modelo de Diseño*: Es un modelo de objetos que describe la realización del Caso de Uso, y sirve como una abstracción del modelo de implementación y su código fuente, es utilizado como entrada en las actividades de implementación y prueba; este modelo se realizado en la disciplina de Análisis y Diseño.
- *Modelo de Despliegue*: Muestra la configuración de los nodos del proceso en tiempo de ejecución, muestra los lazos de comunicación entre estos nodos, así como las de los objetos y componentes que en el se encuentran; se realizado en la disciplina de Análisis y Diseño.
- *Modelo de Datos*: Es un subconjunto del modelo de implementación que describe la representación lógica y física de datos persistentes en el sistema. También incluye cualquier conducta definida en la base de datos como disparadores, restricciones, etc. Es elaborado en la disciplina de Análisis y Diseño.
- *Modelo de Implementación*: Es una colección de componentes, y de subsistemas de aplicación que contienen estos componentes, entre estos están los entregables, ejecutables, archivos de código fuente. Es realizado en la disciplina de Implementación.
- *Modelo de Pruebas*: Es utilizado para la elaboración de las pruebas, y se realiza en la disciplina de Pruebas.

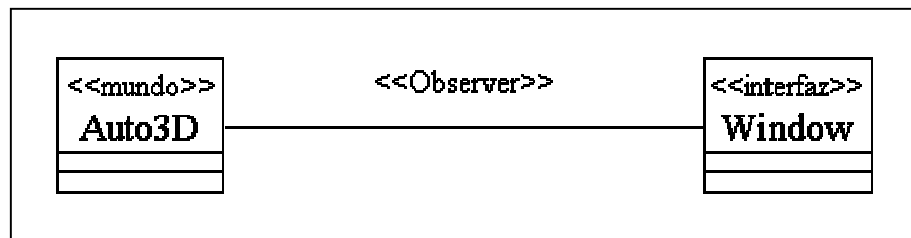
Estos modelos representan los diagramas que propone el UML para el desarrollo de modelado de un proyecto de *software*, con los cuales se puede representar los propuesto por UML mediante la metodología RUP utilizando las herramientas que esta provee para la implementación fácil, clara y estructurada de los diagramas utilizados.

1.3.1. Descripción de estereotipos

Para poder entender la interrelación que tiene UML con RUP se tiene que saber que el inicio de todo está con los casos de uso, para poder tener una base sobre lo cual se quiere trabajar, los casos de uso son la base para esta técnica; luego se procede a la obtención de los diagramas expuestos anteriormente, dependiendo de cuales son los necesarios de implementar, luego se procede a la identificación de los estereotipos, ya que cada uno de estos representan las funciones que se van a definir dentro de los diagramas, estos diagramas nos ayudan a entender la lógica del caso de uso expuesto.

Las clases, al igual que los demás elementos notacionales del UML, pueden estar clasificadas de acuerdo a varios criterios, como por ejemplo su objetivo dentro de un programa. Esta clasificación adicional se puede expresar mediante la utilización de estereotipos.

Figura 15. Ejemplo de estereotipo



En la figura 15, *Auto3D* está clasificado con el estereotipo Mundo (denotado por <<>>), y la clase *Window* con el de interfaz. Nótese que también las relaciones pueden tener esta clasificación, en este caso la relación se identifica como *Observer*.

Los estereotipos más comunes utilizados para clasificar las clases son: Entidad (*entity*), Frontera (*boundary*), Control (*control*). Se proponen varias pautas a seguir a la hora de encontrar las clases de nuestro sistema durante la fase de análisis. Dichas pautas se centran en la búsqueda de los estereotipos entidad, control y frontera:

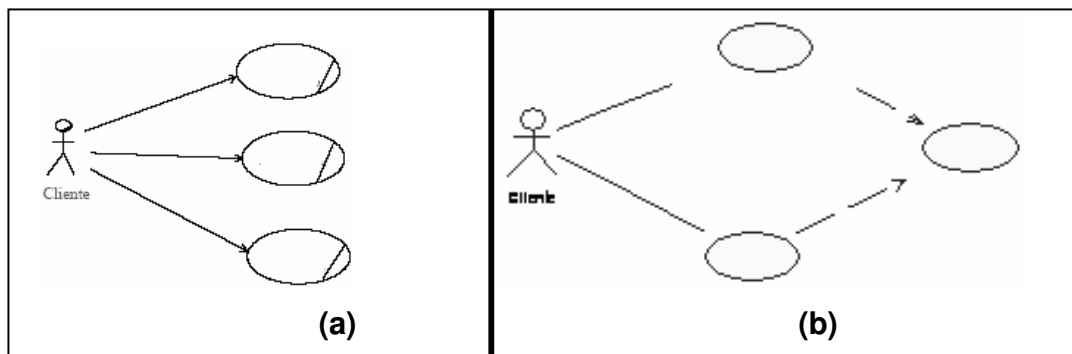
- Una clase entidad modela la información de larga vida y su comportamiento asociado. Este tipo de clase suele reflejar entidades del mundo real o elementos necesarios para realizar tareas internas al sistema. También se denominan clase dominio, ya que suelen tratar con abstracciones de entidades del mundo real.
- Una clase frontera maneja comunicaciones entre el entorno del sistema y el sistema, suelen proporcionar la interfaz del sistema con un usuario o con otro sistema, en general, por tanto, modelan las interfaces del sistema. Cuando se trata de clases que definen la interfaz con otro sistema se refinarán durante la fase de diseño, para tener en cuenta los protocolos de comunicación elegidos.
- Una clase control modela el comportamiento secuenciado específico de uno o varios casos de uso. Se trata de clases que coordinan los eventos necesarios para llevar a cabo el comportamiento que se especifica en el caso de uso, representan su dinámica.

1.3.2. Enlace del RUP con el UML

Conociendo los estereotipos utilizados para la representación de las clases (Entidad, Control y Frontera), ahora podemos explicar la interrelación que existe entre el UML y RUP describiendo los diagramas que describe UML y como se convierten en diagramas del RUP que utilizan estos estereotipos.

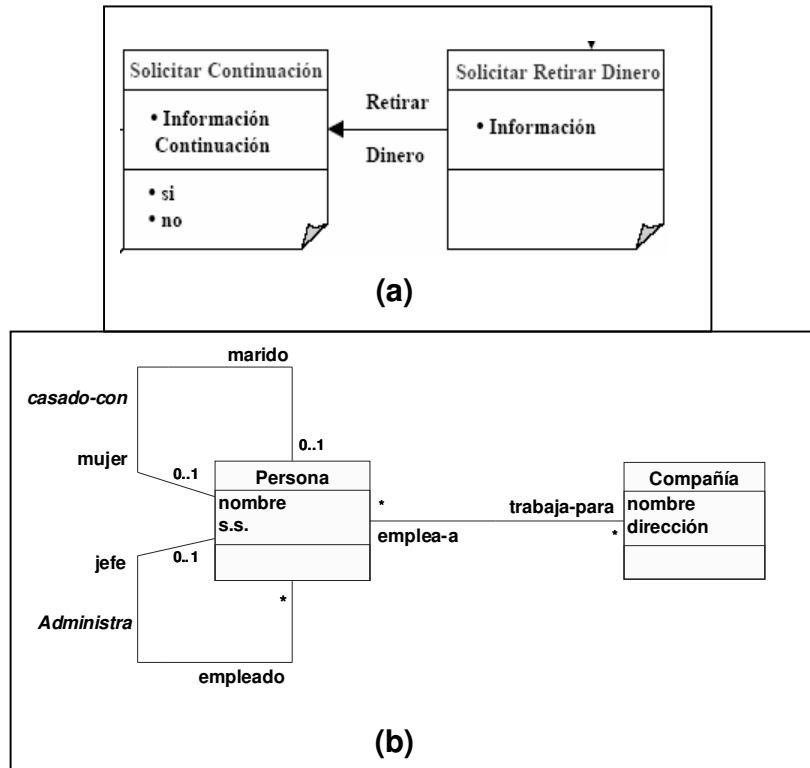
El UML proporciona los diagramas de Caso de Uso, al mismo tiempo que el RUP, la única diferencia es la forma de dibujar los estereotipos, ya que en el RUP son una elipse con una diagonal al lado derecho (pero esto es para casos de uso de negocio, los de sistema no tienen la diagonal), y en UML solamente una elipse, pero en el RUP significa lo mismo a lo que se refiere en UML. En la figura 16 se muestra lo descrito anteriormente, aunque no nos importa en este caso el motivo por el cual se hicieron los diagramas, o la utilización que se les dio, ya que únicamente nos interesa conocer la forma de dibujar ambos diagramas para conocer sus diferencias:

Figura 16. Comparación entre diagramas de casos de uso (a) RUP (b) UML



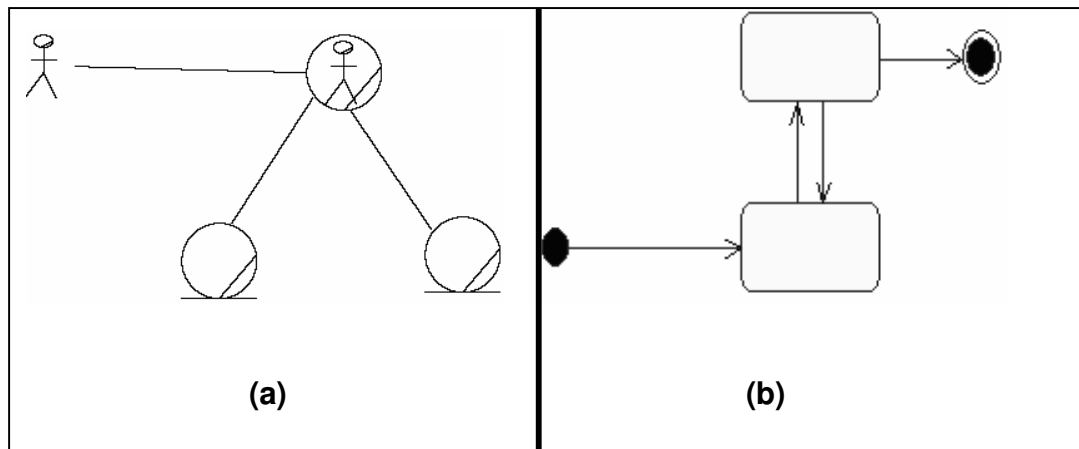
Los diagramas de Clases tienen la misma lógica para lo cual fueron creados en ambos lenguajes, solamente con las diferencias en la forma de dibujar los cuadros que indican las clases, por ejemplo se pueden observar en las siguientes figuras 17(a) y 17(b), que en el RUP se dibujan los cuadros con una pestaña inferior derecha doblada, y en UML solamente rectángulos con ángulos rectos; otra característica que se puede observar es que a la hora de ejemplificar las relaciones uno a uno, uno a muchos etc., se realizan de diferente manera. Pero en ambos lenguajes se puede observar que los diagramas de clases son lo más cercano a la elaboración de un modelo Entidad Relación para la puesta en práctica de la finalización del proyecto de *software*.

Figura 17. Comparación entre diagramas de clases (a) RUP (b) UML



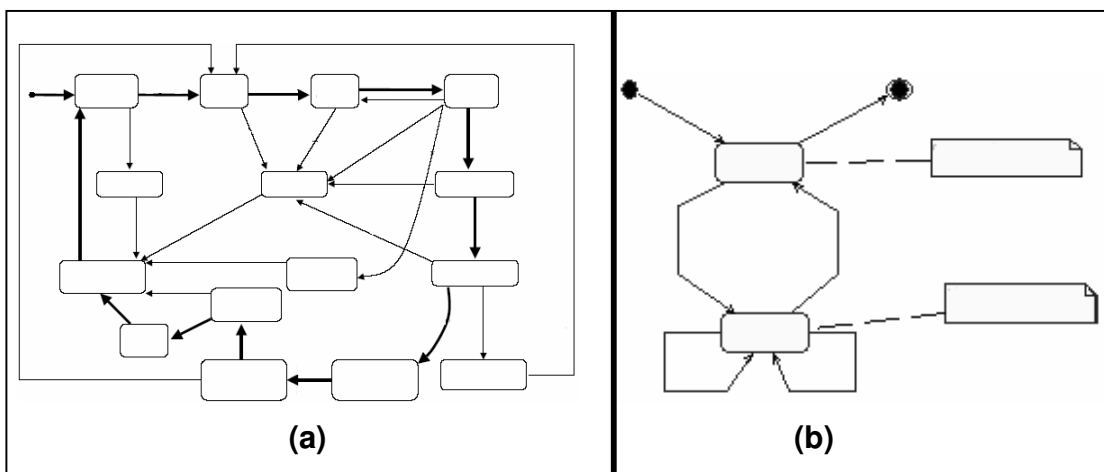
Los diagramas de objetos, difieren únicamente en la forma como se dibujan los objetos o instancias de las clases, ya que en el RUP se dibujan círculos con una diagonal en la parte inferior derecha, y en UML como rectángulos con las esquinas redondeadas, también en RUP no se colocan flechas indicativas, y en UML si. En las siguientes figuras se presentan las diferencias planteadas anteriormente, es importante mencionar que la lógica de cada figura no nos importa en este momento, solamente deseamos representar la forma en que se dibujan los objetos, esto ya que cada una de las figuras 18(a) y 18(b) se refieren a distintos ejemplos.

Figura 18. Comparación entre diagramas de objetos (a) RUP (b) UML



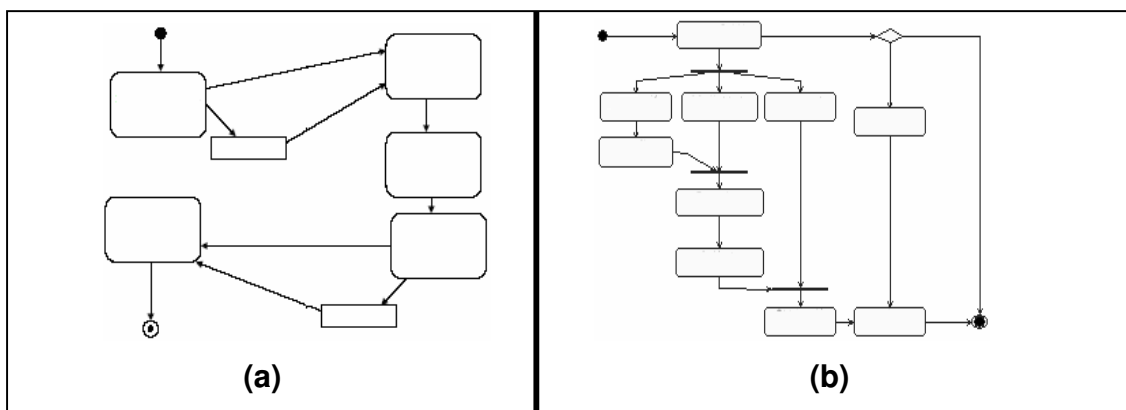
Los siguientes dos diagramas (figura 19 (a) y (b)) presentan la forma como se elabora un diagrama de estados en RUP y UML, se puede observar que de igual manera se elaboran ambos, únicamente que en el diagrama de UML se muestran unos rectángulos con la esquina superior derecha doblada, que indican notas sobre este estado.

Figura 19. Comparación entre diagramas de estados (a) RUP (b) UML



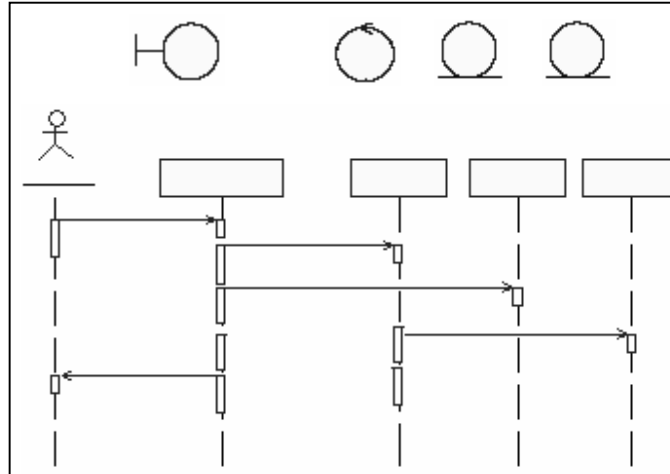
En los diagramas de actividades se utilizan todos los bloques utilizados en la elaboración de diagramas de flujo, por lo tanto en ambos lenguajes se utilizan los mismos, a continuación podemos ver la figura 20 que muestra ejemplos de ambos lenguajes, aunque el enfoque de cada diagrama presentado es distinto, solamente se tomaron de ejemplos para ejemplificar los bloques utilizados.

Figura 20. Comparación entre diagramas de actividades (a) RUP (b) UML



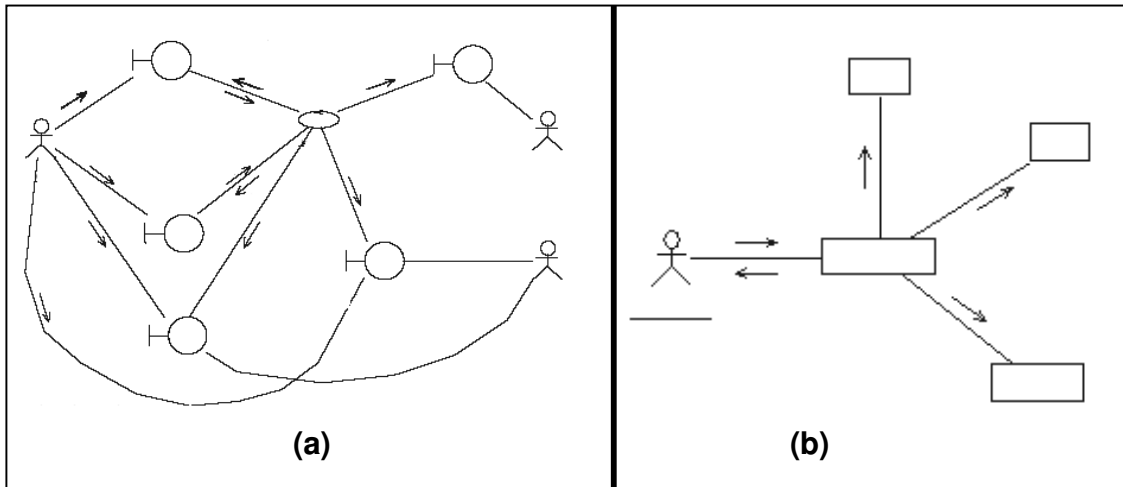
En los diagramas de secuencia se pueden encontrar diferencias leves, como se puede mostrar en la figura 21 los diagramas de secuencia de UML no llevan los símbolos que identifican los estereotipos interfase (círculo con una raya horizontal del lado izquierdo y junto a esta otra vertical), control (círculo con una flecha sobre su borde apuntando al lado izquierdo) y entidad (círculo con una raya horizontal en la parte inferior del mismo), representados por círculos con características independientes

Figura 21. Diagrama de secuencia



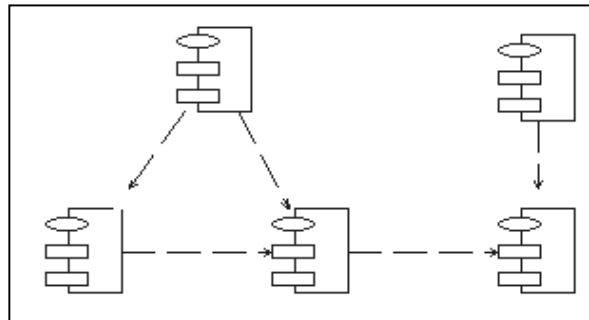
Los diagramas de colaboración se representan similares, con la única diferencia de los bloques que representan las clases, ya que en el RUP se representan por medio de los círculos con sus características individuales de acorde a la función que desempeñan (interfaz, control, entidad), y en UML solamente como rectángulos. En ambos se colocan las actividades que conllevan realizar para llegar a una clase determinada, esto se coloca directamente en la flecha dibujada en la línea que va hacia la clase. Esto se puede observar en la figura 22.

Figura 22. Comparación entre diagramas de colaboración (a) RUP (b) UML



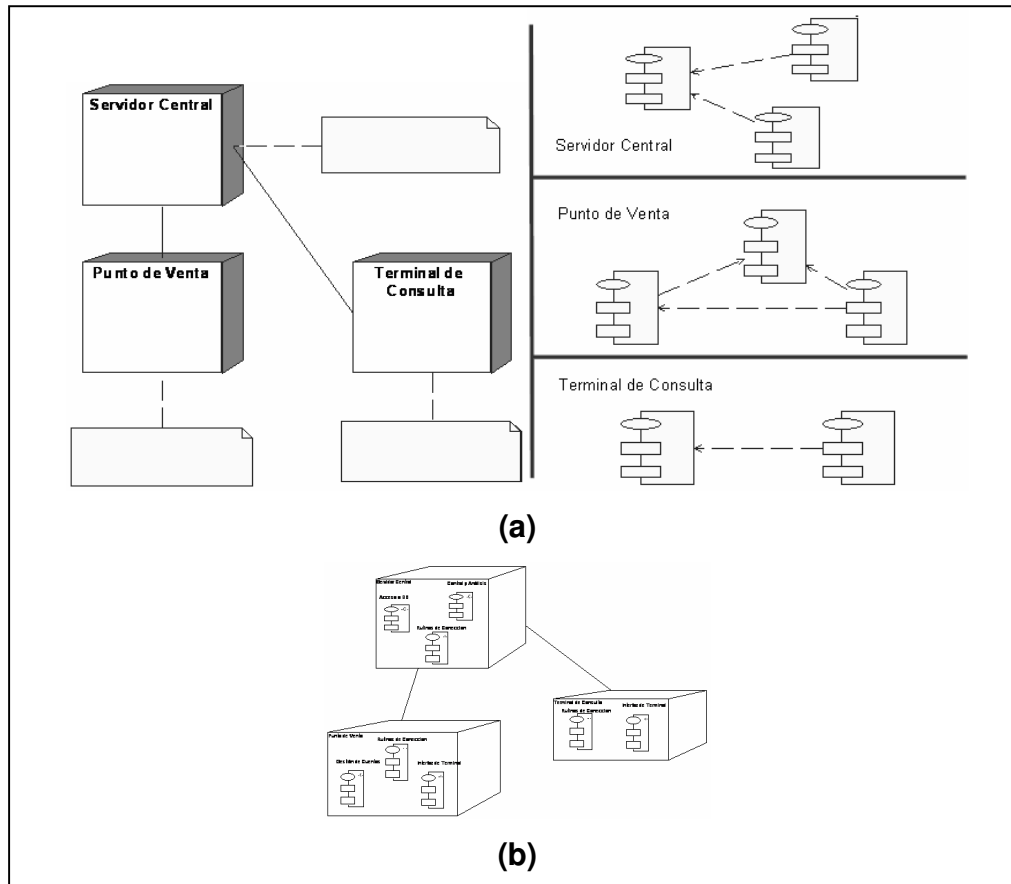
Dentro de los diagramas de implementación se encuentran los de componentes (figura 23), los cuales se representan de manera similar en ambos lenguajes, como se muestra en la figura siguiente.

Figura 23. Diagrama de componentes



La diferencia básica en los diagramas de despliegue (figura 24) es que en UML se dibujan dentro de las cajas los componentes utilizados, en cambio en el RUP se diagraman de forma separada como se muestran en las figuras siguientes.

Figura 24. Comparación entre diagramas de despliegue (a) RUP (b) UML



Se puede observar en todos los diagramas presentados anteriormente que la similitud entre ambos lenguajes es demasiado grande, y es de esperar esto, ya que RUP utiliza los del UML y por lo tanto recopila todo lo que este lenguaje necesita para la implementación, y agrega mejoras, siendo una herramienta de modelado muy eficiente, ya que proporciona todas las herramientas necesarias para tal función, por lo tanto la funcionalidad completa de UML esta descrita e implementada por el RUP, solamente mejorando las características como el cambio de ciertos diagramas de una manera sutil, para diferenciar mas claramente que es lo que se esta haciendo y no perder el enfoque de lo que se desea.

1.3.3. Descripción del método

Primero que todo debemos recordar que existe un Caso de Uso, el cual nos dice la secuencia de pasos a seguir para completar un objetivo, además este Caso de Uso tiene interacciones con Usuarios o Actores, así como con otros Casos de Uso. También recordemos que existen 3 estereotipos principales: Frontera o Presentación, Control y Entidad.

Ahora bien, es muy sencilla la implementación de este método y consiste en pasar lo que se conoce como Caso de Uso (tal y como lo entiende el Cliente), a la Realización del Caso de Uso (tal y como lo entienden los desarrolladores del sistema).

Únicamente se debe de desglosar el Caso de Uso y comprender con quienes o que va a interactuar, al conocer esto se pueden comprender cuantos y cuales estereotipos Frontera se tienen que colocar, por ejemplo: si el sistema es una Caja de un Banco, se debe tener clase Frontera (*boundary*) para que el Usuario (Cajero) pueda acceder a las opciones del sistema.

En el mismo ejemplo se puede entender que el cajero necesita comunicarse con el Banco mediante la clase Frontera, para enviar y recibir información, por tal motivo se necesita un estereotipo Control para el manejo de dichas funciones.

Así mismo se puede observar en este ejemplo que es necesaria la información del Usuario y de la Cuenta (como su ID, Numero Cuenta, Monto Disponible), todos estos están en el Sistema del Banco y por lo tanto cada repositorio de datos que contenga el Banco es una clase Entidad.

Con este ejemplo sencillo podemos observar que de un Caso de Uso se puede mapear a una Realización del mismo en el Análisis únicamente identificando los estereotipos que interactúan en todo el proceso del Caso de Uso. Es de suma importancia mencionar que pueden existir n estereotipos Frontera, Control (generalmente es solo uno) y Entidad en un Caso de Uso particular.

Al encontrar todos los estereotipos del Caso de Uso, se pueden realizar los diagramas de Clases, Secuencia y Colaboración etc. necesarios para ejemplificar los distintos escenarios que se descubran en el Caso de Uso.

Es sumamente sencillo identificar los estereotipos necesarios, para tener bases de cómo hacerlo ver la sección Descripción de Estereotipos incluida en este trabajo; en la cual se mencionan características a tomar en cuenta para poder identificar de una manera concisa estos estereotipos.

En el Capítulo 4 se explica un ejemplo más concisamente de un Caso de Uso y su Realización, en el cual se podrá observar la manera de Identificar estos estereotipos.

2. TECNOLOGÍA PARA DESARROLLO RÁPIDO DE APLICACIONES

2.1. Introducción al RAD

James Martin creó el término “Desarrollo Rápido de Aplicaciones” apuntando hacia una metodología y conjunto de herramientas específicos. Mientras tanto, hoy día se utiliza el término RAD para señalar una serie de tecnologías que utilizan esta metodología y que intentan reducir el tiempo de desarrollo. Esta es una metodología que permite a las organizaciones desarrollar sistemas estratégicamente importantes, de manera más rápida reduciendo a la vez los costos de desarrollo y manteniendo la calidad. Esto se hace por medio de la automatización de porciones grandes del ciclo de vida del desarrollo de sistemas, imponiendo límites entre los plazos de desarrollo y volviendo a usar los componentes existentes y se logra mediante el uso de una serie de técnicas de utilidad comprobada de desarrollo de aplicaciones, dentro de una metodología bien definida. Algunas de estas tecnologías son:

- *JAD (Joint Application Development)*: pequeños grupos (hasta 10 personas) de usuarios y analistas hacen reuniones, para en un corto espacio de tiempo analizar y especificar entradas, procesos y salidas, a través del desarrollo conjunto de un prototipo.
- *Generadores de Aplicación*: estas herramientas posibilitan generar código ejecutable a partir de definiciones generales o prototipos. Son utilizadas como parte de un proceso mayor de *JAD* o prototipación. El mayor problema es la calidad (desempeño) del código generado, principalmente en un ambiente multiusuario.

- Prototipación rápida: el objetivo de esta técnica es obtener en el menor tiempo posible el análisis, diseño e implementación de un sistema, completo o parcial, a través de la utilización de técnicas y tecnologías complementarias (*JAD*, generadores de aplicación, etc.).

Estas técnicas incluyen el uso de:

- Equipos pequeños de desarrollo y bien capacitados.
- Prototipos evolutivos.
- Herramientas poderosas integradas que apoyan el modelo, el prototipo y la reutilización de componentes.
- Un depósito central de la información para tenerla a la mano en el momento que se le necesita.
- Requisitos interactivos y talleres de diseño.
- Límites rígidos en los plazos de desarrollo.

Las herramientas gráficas orientadas a objetos tienen, casi todas, interiorizadas el concepto general de *RAD*. Además, con la creación bien planificada de objetos, la programación de nuevos módulos se vuelve cada vez más simplificada, reutilizando los objetos creados anteriormente.

Uno de los conceptos de *RAD* más interesantes, y que provee mejores resultados prácticos, es el de “entrega incremental de productos”. La idea es detectar durante el análisis módulos del sistema tributario que puedan ser desarrollados e implantados aisladamente, y trabajar en este sentido utilizando las técnicas descritas anteriormente.

Por ejemplo, en el subsistema de Registro de Contribuyentes, el módulo de captura de datos del contribuyente puede ser rápidamente desarrollado e implantado por un pequeño equipo de personas, mientras se desarrollan otros módulos: actualización, emisión de tarjetas, estadísticas, etc.

Para el éxito de un desarrollo tipo *RAD* el personal técnico elegido debe poseer fuertes habilidades de relaciones interpersonales, aliadas a un dominio excelente de las herramientas utilizadas y también conocer el negocio. Además, es esencial la disponibilidad y el fácil acceso a los usuarios para la realización de las muchas reuniones requeridas. Con esto se puede decir que una de las limitantes de implementar el RAD es el costo elevado, debido a las exigencias que requiere para su implementación, tanto de personal como de tecnología.

El RAD apoya el análisis, el diseño, el desarrollo y la implementación de los sistemas de aplicación individual. Sin embargo, el RAD no apoya la planificación o el análisis necesario para definir las necesidades de información de la empresa en su totalidad o de un área empresarial principal de la empresa.

2.1.1. Etapas de la metodología RAD

La metodología del RAD tiene cuatro etapas principales:

1. La etapa de Definición Conceptual que define las funciones del negocio y las áreas sujeto de datos que el sistema apoyará y determina el alcance del sistema.

2. La etapa de Diseño Funcional que usa los talleres para modelar los datos y los procesos del sistema y para construir un prototipo de trabajo de los componentes críticos del sistema.
3. La etapa de Desarrollo que completa la construcción física de la base de datos y del sistema de aplicación, construye el sistema de conversión y elabora ayudas de usuarios y planes de trabajo a desarrollar o de despliegue.
4. La etapa de Despliegue que incluye la puesta a prueba y la capacitación del usuario final, la conversión de datos y la implementación del sistema de aplicación.

2.1.2. Características de la metodología RAD

- **Modelo Central:** Se pueden crear modelos o redefinir modelos existentes, y se pueden integrar estos modelos con la funcionalidad de aplicaciones existentes (componentes, paquetes, etc.)
- **Desarrollo Visual:** Proporciona un nivel alto de abstracción, y da facilidad de crear nuevas aplicaciones y mantener las existentes.
- **Código Construido:** Diseñado para alto rendimiento, escalabilidad y ahorro de tiempo.
- **Finalización de la Integración del Desarrollo del Ciclo de Vida:** Proporciona un desarrollo de artefactos y semántica del negocio capturados y organizados en modelos visuales. Universalmente aplicados durante el desarrollo del proyecto.
- **Dar esfuerzo a la Orientación a Objetos:** Implica que el proceso de desarrollo esta manejado por el modelo del negocio (clases).

- **Extensible:** La integración que tiene abarca: XML, Servicios *Web*, *Java* / componentes EJB, DHTML.

2.1.3. Problemas en la metodología RAD

Los problemas que se han encontrado a esta metodología son:

1. Se requiere que el problema sea fácilmente modularizable.
2. Se requiere de recursos Humanos para cada equipo
3. Cada equipo debe estar altamente comprometido y con la capacidad de manejar las herramientas muy bien.

RAD no es recomendable cuando los riesgos técnicos del proyecto son altos. Por ejemplo cuando se introducen nuevas herramientas, nueva tecnología no probada, o cuando se requiere de complicadas interfaces con *software* ya existente.

Hay voces en favor y en contra de la efectividad de la técnica *RAD*. Algunas veces, el tiempo reducido de puesta en marcha de un sistema es obtenido al costo de baja calidad y/o difícil mantenimiento y/o un pobre desempeño.

2.2. Introducción a J2EE

J2EE, la plataforma creada por SUN en el año 1997 es la que ofrece perspectivas de desarrollo para empresas que quieran basar su arquitectura en productos basados en *software* libre.

2.2.1. JSR

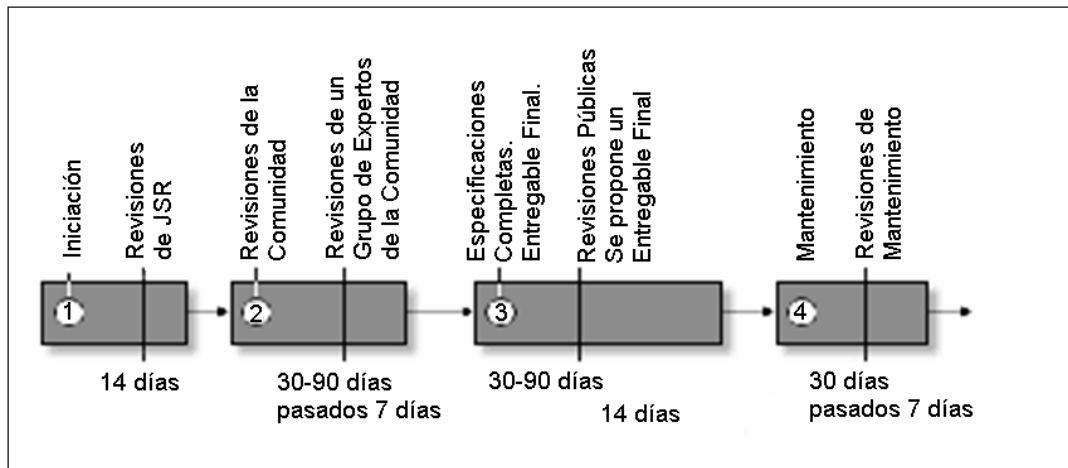
JSR es el acrónimo de *Java Specification Request*. Cuando una persona o entidad cree que es necesaria la presencia de una determinada tecnología dentro de las plataformas basadas en *Java*, lo que hace es crear un JSR y presentarlo para su aprobación. Dentro de este documento se relata por que es necesaria dicha tecnología, por que no se pueden abordar los problemas que soluciona con las tecnologías existentes, etc.

2.2.2. JCP

Java, siempre fue criticado por ser única y exclusivamente de SUN. A raíz de todas esas críticas, SUN, el 8 de diciembre de 1998, decidió dar la posibilidad a todo el mundo de participar en la evolución de *Java* y de todas las plataformas que se basan en *Java*. Con esa intención se creó el JCP, que es un organismo formado por alrededor de 500 empresas, asociaciones y particulares cuyo objetivo es asegurar la evolución de las plataformas basadas en *Java*.

Para entrar a formar parte del JCP las empresas e individuales han de pagar una cuota anual a SUN. Cualquiera puede participar en el desarrollo de cualquier parte de la plataforma, previo pago de esa cuota, y por supuesto, si el comité de la especificación en la que quiere participar considera que esa persona o entidad tiene los conocimientos necesarios para aportar algún beneficio.

Figura 25. Esquema del JCP



Una de las labores del JCP es la de controlar la evolución de las diferentes especificaciones que forman las plataformas basadas en *Java*. Este organismo es el encargado de decidir que especificaciones se aprueban y de controlar las fases por las que pasan.

Después de ver los conceptos de JSR y JCP estamos preparados para definir lo que es J2EE. J2EE es una especificación, un JSR (concretamente el JSR-151), que define una plataforma de desarrollo empresarial, a la que llamaremos la plataforma J2EE. La plataforma J2EE está formada de varios componentes:

- Un conjunto de especificaciones que relatan por que es necesaria dicha tecnología, y que problemas va a solucionar.
- Una prueba de compatibilidad, el J2EE *Compatibility Test Suite* (CTS).
- La implementación de referencia de J2EE.
- Un conjunto de guías de desarrollo y de prácticas aconsejadas denominadas J2EE *BluePrints*.

Estas especificaciones definen con mucho más detalle los diferentes componentes de los servidores de aplicaciones (se describe posteriormente este concepto), como puedan ser un contenedor *Web*, un servidor de mensajería, el sistema de seguridad, etc. De algún modo, se podría decir que la especificación J2EE engloba a un gran conjunto de especificaciones. Por poner un ejemplo en la especificación de J2EE 1.4 se definen las siguientes especificaciones:

- JSR-109, (*Servicios Web*)
- JSR-101, (*JAX-RPC, Java API for XML-based RPC*)
- JSR-67, (*JAXM, Java API for XML Messaging*)
- JSR-93, (*JAXR, Java API for XML Registries*)
- JSR-77, (*Configuración y control*)
- JSR-88, (*API de despliegue*)
- JSR-115, (*Interfaz de servicios de autorización*)
- JSR-56, (*JNLP, Ejecución remota de aplicaciones*)
- JSR-112, (*JCA 2.0, Arquitectura de conectores*)
- JSR-152, (*JSP 1.3, Java Server Pages*)
- JSR-152, (*Servlets 2.4*)
- JSR-153, (*EJB 2.1, Enterprise Java Beans*)
- JSR-9XX, (*JAXP 1.2, Soporte de esquemas XML*)
- JSR-9XX, (*JMS 1.1, API de mensajería*)

Como se puede ver, todas estas especificaciones tienen asociado un JSR, regido por un comité de empresas, asociaciones o individuos que se aseguran de crear dichas especificaciones y de que vayan evolucionando. Cualquiera puede descargar las especificaciones y a la vez puede participar en la creación de estas.

La gran importancia de toda esta enorme lista de especificaciones radica en que cuando se utiliza un servidor de aplicaciones que implementa la plataforma J2EE, se cuenta de manera automática con todos estos servicios. Es decir, se ponen a nuestra disposición una gran caja de herramientas que podemos aprovechar para realizar aplicaciones de una manera mucho más eficaz.

2.2.3. Ventajas de J2EE

J2EE, nos ofrece entre otras las siguientes ventajas:

- 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*.
- Organismo de control: La plataforma J2EE está controlada por el JCP, un organismo formado por más de 500 empresas. Entre las empresas que lo forman están todas las más importantes del mundo informático (SUN, IBM, *Oracle*, SAP, HP, AOL, etc.) lo que garantiza la evolución de la misma.
- Competitividad: Muchas empresas crean soluciones basadas en J2EE y que ofrecen características como rendimiento, precio, etc., muy diferentes. De este modo el cliente tiene una gran cantidad de opciones a elegir.
- Madurez: Creada en el año 1997 como respuesta a la tecnología MTS de *Microsoft*, J2EE tiene ya cinco años de vida y una gran cantidad de proyectos importantes a sus espaldas.

- Soluciones libres: En la plataforma J2EE es posible crear arquitecturas completas basadas única y exclusivamente en productos de *software* libre. No sólo eso, sino que los arquitectos normalmente disponen de varias soluciones libres para cada una de las partes de su arquitectura.

2.2.4. Desventajas de J2EE

Aún así, la plataforma de J2EE también tiene desventajas, algunas importantes:

- Depende de un único lenguaje: 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.
- Complejidad: 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. Es aquí donde herramientas como RAD pueden ayudar.
- Heterogeneidad: Existe una gran heterogeneidad en las soluciones de desarrollo. No existe en J2EE un símil a Visual Studio .NET. La gran cantidad de herramientas disponibles causa confusión dentro de los desarrolladores y puede crear dependencias dentro de las empresas.

Existe mucha confusión, sobre todo entre la gente alejada del mundo de *Java*, sobre lo que es en realidad J2EE. La confusión más habitual es pensar que J2EE es un producto concreto que distribuye SUN *Microsystems* y que se puede descargar desde su página *Web*. Nada más lejos de la realidad. No existe un J2EE concreto, no se puede ir a la página *Web* de SUN *Microsystems* y descargar "el J2EE".

2.2.5. Integración con otros sistemas

En cuanto a la integración con otros sistemas, gran parte del modelo de J2EE (en especial los EJB) está basado en CORBA lo que quiere decir que es posible comunicarse sin ningún tipo de problema con otras aplicaciones creadas en otros lenguajes diferentes de *Java* y viceversa. Las posibilidades de integración todavía son mayores ya que cualquier EJB que hayamos creado se puede exponer como un servicio *Web*. Por último otra tecnología que merece la pena nombrar es la de los conectores, que por explicarlo de una manera clara, son un puente entre J2EE y sistemas *legacy* (por ejemplo un conjunto de ficheros de datos en COBOL), Los conectores tienen un API estándar que nos permite acceder a estos sistemas *legacy* de una manera transparente y sin apenas esfuerzo.

Otra de las grandes ventajas de J2EE viene del hecho de que está basado en *Java*, ya que la variedad de clientes que pueden conectarse a una arquitectura J2EE es inmensa, desde aplicaciones de escritorio, hasta móviles, pasando por televisiones, PDAs, etc., cualquiera de estos productos puede conectarse ya sea a través de cualquier protocolo de comunicación lo que nos da una gran gama de posibilidades en cuanto a la conectividad de nuestros sistemas.

2.3. Arquitectura de múltiples capas

2.3.1. El modelo de desarrollo de J2EE

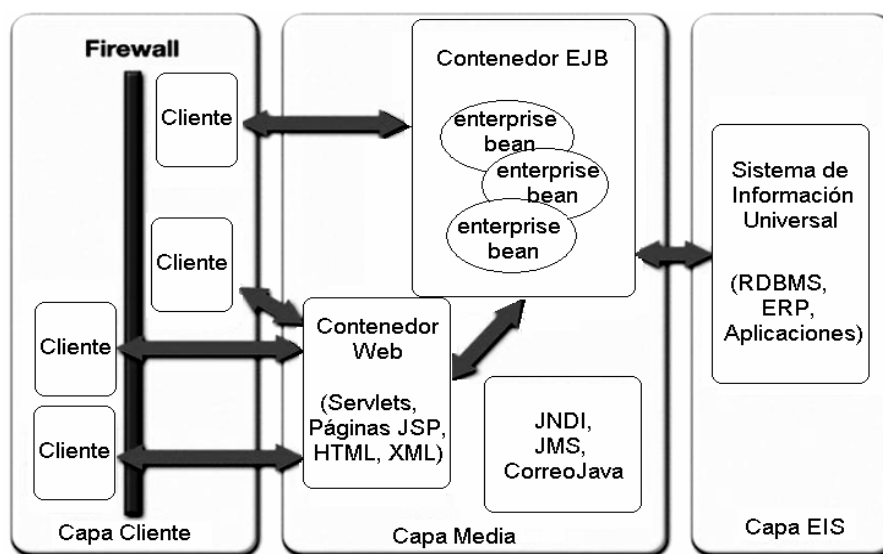
La plataforma de J2EE define un modelo de programación encaminado a la creación de aplicaciones basadas en n-capas. Típicamente una aplicación puede tener cinco capas diferentes:

- Capa de cliente: Representa el interfaz de usuario que maneja el cliente.
- Capa de presentación: Representa el conjunto de componentes que generan el la información que se representará en el interfaz de usuario del cliente. Típicamente se creará a través de componentes basados en *Servlets* y *JSP*.
- Capa de lógica de negocio: Contiene nuestros componentes de negocio reutilizables. Normalmente se forma a partir de componentes *EJB*.
- Capa de integración: Aquí se encuentran componentes que nos permiten hacer más transparente el acceso a la capa de sistemas de información. Por ejemplo este es el lugar idóneo para implementar una lógica de objetos de acceso a datos, *DAO (Data Access Objects)*.
- Capa de sistemas de información: Esta capa engloba a nuestros sistemas de información: bases de datos relacionales, bases de datos orientadas a objetos, sistemas *legacy*, etc.

Las ventajas de un modelo como este son muy importantes. Al tener las capas separadas tenemos que existe poco acoplamiento entre las mismas, de modo que es mucho más fácil hacer modificaciones en ellas sin que interfieran en las demás.

Todo esto redundando en la obtención de mejoras en cuanto a mantenibilidad, extensibilidad y reutilización de componentes. Otra de las ventajas que obtenemos es que se promueve la heterogeneidad de los clientes ya que añadir nuevos tipos de cliente (móviles, *set-top-boxes*, PCs, etc.) se reduce a añadir nuevas capas de interfaz de usuario y presentación, sin tener que modificar todo el resto de capas.

Figura 26. Esquema de un modelo mixto de tres y cuatro capas



En la figura 26 se puede ver lo que sería una posible arquitectura J2EE. El modelo que aparece en la figura está dividido en varias capas, con una separación clara entre presentación, lógica de negocio y sistemas de información empresariales. En este caso además, podemos ver como se trata de un modelo mixto. En la parte de arriba tenemos que se recorre un camino por una estructura de tres capas (aplicación - lógica de negocio - sistemas de información), mientras que por el segundo camino recorreremos una estructura de cuatro capas (aplicación - lógica de presentación - lógica de negocio - sistemas de información).

Como ya hemos dicho, el modelo de desarrollo con J2EE está basado en componentes reutilizables, con el objetivo de aumentar la reusabilidad de las aplicaciones. Estos componentes, además, gracias a las especificaciones, son intercambiables entre servidores de aplicaciones, por lo que la portabilidad de nuestros desarrollos es máxima. Si hay un tipo de componentes que requieren una atención especial estos son los *Enterprise Java Beans*. Se trata de objetos distribuidos, que como hemos dicho contienen la lógica de negocio de nuestras aplicaciones y que hacen transparente al programador operaciones como la persistencia, la seguridad, la gestión de transacciones, etc.

2.3.2. Servidores de aplicaciones

Los servidores de aplicaciones son el corazón de la plataforma J2EE. En ellos residirán todos los componentes de una empresa, ya sean objetos distribuidos accesibles remotamente, páginas *Web*, o incluso aplicaciones completas. Un servidor de aplicaciones que implemente la plataforma J2EE nos tendrá que ofrecer de manera automática todas las especificaciones que define este estándar. Esto es muy importante, ya que de una manera sencilla el desarrollador se encuentra con una gran caja de herramientas. De este modo, en un servidor de aplicaciones normal tendremos un contenedor de *servlets*, un contenedor de EJBs, sistemas de mensajería, y un gran número de herramientas que nos ayudarán a incrementar la productividad del equipo.

La pieza más importante dentro de un servidor de aplicaciones es el contenedor de EJBs. Los EJBs son objetos reutilizables que contienen la lógica de negocio de nuestro sistema. Los contenedores de EJBs son servidores que se encargan de controlar todos estos componentes, esto es muy importante ya que se automatizan tareas como la gestión del ciclo de vida, la gestión de las transacciones, la gestión de persistencia, etc.

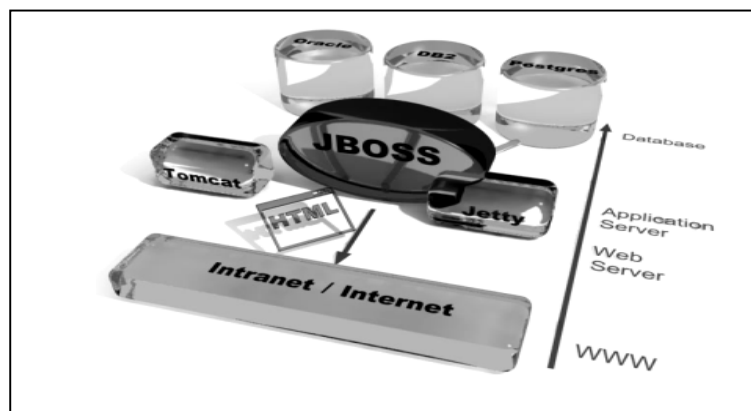
Los desarrolladores, de este modo, no tienen que centrarse en crear servicios de bajo nivel y pueden centrarse en la creación de lógica de negocio empresarial.

En la actualidad el mercado de los servidores de aplicaciones es uno de los más activos y todas las empresas están luchando ferozmente por ser los líderes del sector. Estos dos servidores son realmente maravillas tecnológicas pero su precio hace que muchas empresas no se puedan permitir el lujo de comprar sus licencias. Por suerte existen una serie de servidores de aplicaciones absolutamente libres que no tienen demasiado que envidiarles.

2.3.2.1. JBoss

JBoss es el servidor de aplicaciones libres por excelencia (figura 27). Su licencia es LGPL y está implementado al 100% en *Java*. Tiene más de 150.000 descargas mensuales lo que le convierte en el servidor de aplicaciones más descargado del mundo. JBoss está avalado por un grupo de desarrollo formado por arquitectos con gran experiencia y repartido por todo el globo.

Figura 27. JBoss es el servidor de aplicaciones que está más de moda actualmente



2.3.2.2. JOnAS

No tan conocido como JBoss este servidor de aplicaciones, por sus características, es uno de los proyectos más ambiciosos del mundo del *software* libre en *Java* y pronto superará a JBoss en aceptación.

2.3.2.3. OpenEJB

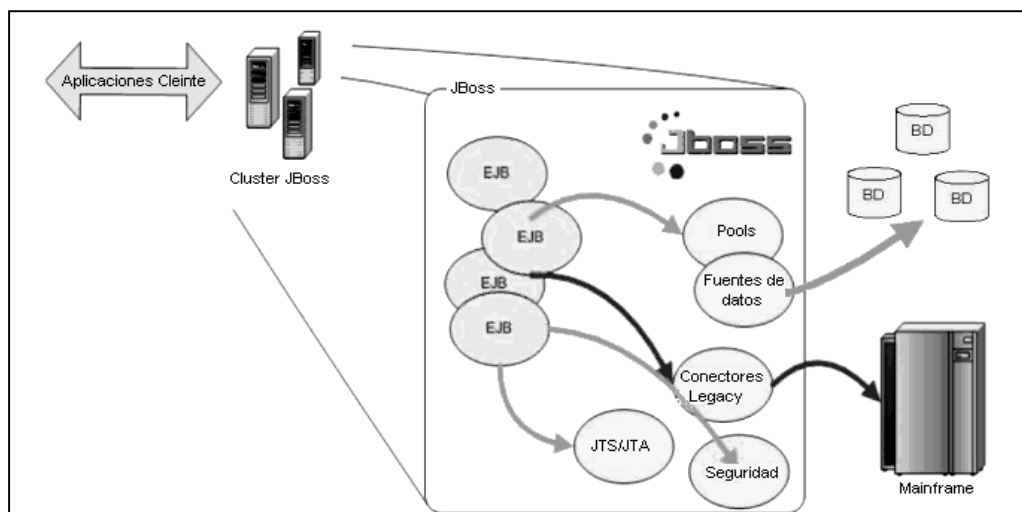
Estrictamente, OpenEJB no es un servidor de aplicaciones. *OpenEJB* es un contenedor de EJBs pero lo hemos incluido dentro del apartado de servidores de aplicaciones ya que es compaginado con otros productos de la empresa que lo desarrolla.

2.3.2.4. Ejemplo práctico de servidor de aplicaciones

Se desea montar un sistema capas de ejecutar las peticiones de los clientes situados en diferentes delegaciones lejanas a este sistema, todo esto por medio de la *Web*, ofreciendo las ventajas del trabajo por medio de la *Web*; también se pretende tener una gran cantidad de clientes utilizando este sistema, o sea que soportar gran cantidad de procesos. También se deberán acceder bases de datos localizadas en otros lugares por medio del sistema, para el procesamiento de la información que el usuario necesite. A continuación se presenta una solución factible y sencilla utilizando los servidores JBoss:

Primero se instalará un clúster de servidores JBoss, de modo que se pueda aprovechar las características de balanceo de carga que ofrecen estos sistemas. Este se localizará en la central de algún lugar lejano y desde ahí se servirán las peticiones de los clientes situados en las diferentes delegaciones.

Figura 28. Ejemplo de servidor de aplicaciones



Como se puede apreciar en la figura 28, dentro del servidor de aplicaciones estará el contenedor de EJBs donde residirán los diferentes EJBs del sistema que contienen la lógica de negocio. Estos componentes reciben de manera automática una serie de servicios que proporciona el contenedor: transacciones, persistencia, seguridad, etc. Además, estos componentes pueden aprovecharse de los conectores que proporciona el servidor de aplicaciones para acceder a sistemas *legacy*, por ejemplo para acceder a ficheros COBOL, etc.

2.4. Enlace del RUP, UML, RAD y J2EE

Como ya se indicó el RUP nace a partir del lenguaje UML, por lo tanto trae todas las características del mismo, y con las mejoras que fueron presentadas, incluyendo las herramientas utilizadas por el mismo.

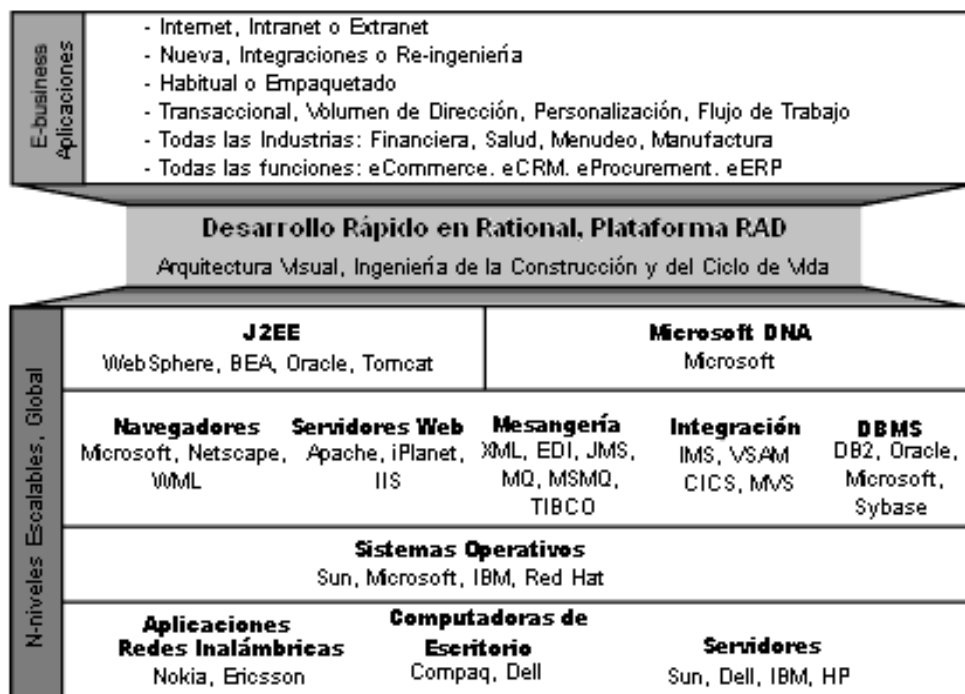
2.4.1. Diseñador rápido de *rational* (RRD)

El Diseñador Rápido de *Rational* (*Rational Rapid Developer* RRD) integra el modelado visual y automatiza la construcción del sistema que permite el diseño, desarrollo y despliegue rápido, en aplicaciones *e-business* (comercio electrónico) y *end-to-end* (extremo a extremo). El RRD se enfoca particularmente a encontrar los desafíos inmediatos de una empresa IT (Tecnología de Información), incluyendo el comercio B2B (*business to business*, negocio con negocio), la integración de aplicaciones de empresa, y creando, publicando, descubriendo e integrando servicios *Web*.

RRD transforma el desarrollo de la aplicación, de una forma de programación metódica a una disciplina arquitectónica y diseñada, ya que se desarrollan los modelos de las aplicaciones utilizando un juego de herramientas arquitectónicas visuales, esto logra un desarrollo rápido de aplicaciones con alta calidad sin tener que saber las complejidades de la programación directa de código en aplicaciones de n capas, ya que la generación de código por medio de los modelos ofrece características de seguridad y fiabilidad en la funcionalidad del sistema. Debido a la evolución que están viviendo estas aplicaciones, cambiando los requerimientos del negocio y de tecnología, el uso del RRD combina la utilización del modelado basado en el RUP para la fácil generación y modificación de estas aplicaciones.

En la figura 29 se muestra la manera en que el RRD basado en la plataforma RAD, une cualquier problema de la vida real, con los diversos niveles, los cuales representan lo necesario para que se pueda realizar el sistema y que funcione de acorde a los requerimientos planteados por el problema en cuestión. Como se puede apreciar, en cada nivel se muestra la diversidad de opciones que se pueden tomar.

Figura 29. Unión de las aplicaciones con los diversos niveles mediante el RRD basado en la plataforma RAD



En la figura 29 se puede observar que el RRD que utiliza la plataforma RAD necesita de la plataforma J2EE como una base en la generación de componentes mediante la generación de código creado con un desarrollo rápido o simplemente utilizar los existentes, basados en los estándares definidos en la sección de especificaciones de J2EE. Utilizando el J2EE se tiene una aplicación con todas las ventajas del código *Java*. Utilizando estas plataformas se pueden tener servidores de aplicaciones que utilicen los demás niveles o capas para entrelazar funcionalidades diversas según sea la necesidad de la aplicación a realizar.

2.4.2. Modelando el sistema con RRD

El desarrollo de una aplicación empieza con la obtención de los requerimientos funcionales, el Modelado del Sistema en RRD se usa para crear modelos detallados basados en los requerimientos planteados. El modelado empieza en la fase de inyección de la aplicación, al momento de la obtención de requerimientos, llevando en paralelo la fase de diseño, cuando se esta creando el nivel mas alto mostrado en la figura 29, en de la aplicación que se desee implementar. Finalmente, durante la fase de aplicación, se planean las partes detalladas de la aplicación, que incluyen aspectos como: objetos del negocio, interfaces de usuario, mensajería, e integración. El proceso de modelado, debe incluir alguna persona que codifique, pero este trabajo se limita a la creación esencial de la lógica del negocio, necesitada para implementar las reglas del negocio y de procesos, y está comprendida en menos del 5% del total de código en la aplicación.

RRD provee un modelado detallado de una aplicación, implementando estos modelos dentro de las fases del RUP, se obtienen todas las ventajas antes mencionadas de un desarrollo rápido de aplicaciones. El RRD conjuntamente con el RAD, implementan las disciplinas del RUP, aunque minimiza el uso de todas estas y las agrupa en 4 principales, para poder tener las ventajas de un desarrollo rápido (ver figura 30). Los diversos modelos que se pretenden realizar durante la elaboración de una aplicación, se muestran en la figura 30, varios de estos modelos fueron explicados en la sección de los modelos (ver Índice), siendo estos los que utiliza la metodología RUP, más sin embargo, aparecen otros modelos que conjuntamente con los anteriores, son utilizados en el RRD. Estas disciplinas o flujos de trabajo conjuntamente con sus modelos son explicados a continuación:

- **Requerimientos:** (*Modelo de Casos de Uso*), cuyo objetivo fundamental es el modelado de los casos de uso, mediante los requerimientos planteados.
- **Modelando Objetivos del Negocio:** (*Modelo de Clases, Modelo de Procesos, Modelo de Base de Datos, Modelo de Análisis, Modelo de Seguridad, Modelo de Reglas del Negocio*) este empieza con el modelado de las clases, con lo cual se define la estructura estática de la aplicación; se define la persistencia de clases a través del modelado de la base de datos. La dinámica de la aplicación es modelada utilizando *Servicios Web*, modelando componentes y modelando procesos; el modelado del sistema también provee el modelado de seguridad y modelado de reglas del negocio.
- **Modelando Interfase de Usuario:** (*Modelos de Sitios, Modelos de Estilo, Modelos de Tema, Modelos de Página*) provee un sistema de modelado de sitios de navegación para los usuarios y poder así lograr interfaces comprensivas y fácil de utilizar. Con el modelado de temas y estilos se pueden lograr apariencias consistentes y agradables. La parte más importante de este sistema de modelado, es el modelo de transacción que es incluido en cada página, que interrelaciona los objetos del negocio y datos; este modelo de transacción permite a la página proporcionar interacción total con la aplicación.
- **Modelando Mensajería:** (*Modelo de Protocolo, Modelos de Mensajes, Modelos de Integración*) provee un sistema de modelado de mensajería, para poder enviar y recibir mensajes en muchos formatos, incluyendo XML; se pueden enviar y recibir mensajes utilizando una variedad de plataformas incluyendo series MQ, Sistema de Mensajería de *Java*, y otros.

- **Modelando el Despliegue:** (*Modelo de Tecnología, Modelo de Despliegue, Modelo de Código, Modelo de la Prueba*) provee un sistema de modelado de la tecnología a utilizar, si es servidor o una computadora personal, la marca y el tipo, con lo cual se puede modelar esta tecnología; también modelar las pruebas a realizar, así como el despliegue que tendrá la aplicación.

Figura 30. Modelando en RRD basado en la plataforma RAD, y en la metodología RUP

Flujos de Trabajo Funcionales	Fases			
	Incepción Alcance de los Casos del Negocio	Elaboración Requerimientos Análisis Diseño Plan del Proy	Construcción Implementación Construcción de Unidades de Prueba	Transición Pruebas del Sistema Prueba Beta Descarga
Requerimientos		Modelo de Casos de Uso		
Modelando Objetivos del Negocio		Modelo de Clases Modelo de Procesos Modelo de DB Modelo de Análisis Modelo de Seguridad M.Reglas del Negocio		
Modelando Interfase de Usuario		Modelos de Sitios M. de Estilo M. del Tema M. de Página		
Modelando Mensajería y EAI		Modelo de Protocolo M. de Mensajes M. de Integración		
Modelando el Despliegue			Modelo de Tecnología Modelo de Despliegue Modelo del Código Modelo de la Prueba	

El objetivo es aprender a utilizar los principios, actividades y artefactos definidos por RUP para identificar requisitos, analizarlos y transformarlos en un diseño robusto en el entorno J2EE, también aplicar las diferentes fases de RUP en la plataforma RAD obteniendo el RRD sobre proyectos de desarrollo basados en la plataforma J2EE. El lenguaje unificado de modelado, UML, se implementa de forma intensiva para representar y refinar los diferentes artefactos de la metodología. Con estos enlaces se es capaz de comprender las diferentes perspectivas de una arquitectura de *software* y hacer uso apropiado de ella para la definición y programación de componentes de la plataforma J2EE, basados en un esquema de desarrollo rápido RRD.

3. HERRAMIENTAS A UTILIZAR PARA UN DESARROLLO RÁPIDO DE APLICACIONES

3.1. Herramienta XDE

3.1.1. Introducción a XDE

XDE es una herramienta diseñada especialmente para desarrolladores, como un ambiente extendido de desarrollo (*eXtended Development Environment*), integrando herramientas de diseño y desarrollo de código en un único ambiente de desarrollo tanto para la plataforma .NET como J2EE; permite que los usuarios trabajen en un único ambiente, evitando la necesidad de cambiar entre muchos no integrados.

Cuenta con un poderoso motor de soporte de patrones que hace posible su fácil adopción y permitirá acelerar los proyectos ya que no hay necesidad de comenzar con una página en blanco ya que estos patrones nos proporcionan código en el cual se puede empezar a especificar el propio código a partir de un esqueleto. Con esta herramienta se puede realizar en menos tiempo lo deseado ya que provee visualización UML, *templates* de código, y sincronización automática o manual de código y modelos, presentada de forma tal que se puede aprender *Unified Modeling Language* (UML) a medida que se produce.

Con esta herramienta se optimiza la forma de trabajo, basado en las siguientes características:

- Sincronización automática o manual de patrones.
- *Templates* de patrones y código definible por el usuario para automatizar las tareas repetitivas de codificación.
- Modelado de formato libre para formas personalizadas específicas del dominio.
- Referencias cruzadas entre modelos y versionado hasta el nivel de clase y diagrama permiten la estructuración que se ajusta al cualquier proyecto.

Este entorno completo de desarrollo combina las características de: Modelado, Codificación, Construcción, Compilación, *Debug*.

3.1.2. Características de *rational XDE*

- Desarrollado en *Java*
- Entorno totalmente integrado de desarrollo con el modelado de diseño
- Sincronización de código con modelado de diseño
- Ingeniería inversa
- Editor de código *Java*
- Plantillas de código que nos permiten automatizar tareas repetitivas.
- Personalización de patrones o uso de los ya incorporados de GoF.
- Plantillas de *frameworks*.
- Integración con *Clear Case* para todo el proyecto: código, documentación y diseño.

- Múltiples Modelos. Podemos trabajar con múltiples modelos de abstracción manteniendo trazabilidad.
- Modelado libre. Podemos trabajar en un formato libre sin las restricciones del lenguaje, para así poder reflejar nuevas ideas.

3.2. Herramienta *WebSphere*

Esta herramienta proporciona funcionalidad *e-business* (negocios por la *Web*) bajo demanda, y opera en más de 35 sistemas operativos diferentes, incluyendo Linux.

WebSphere es una herramienta que ayuda a incrementar la eficiencia operacional, aportando agilidad y escalabilidad. Los servidores de aplicaciones para entornos de desarrollo y ejecución J2EE para la administración de transacciones con una mayor seguridad, rendimiento, disponibilidad y conectividad, permitiendo el equilibrio entre servicios *Web* y activos de *software* existentes.

Otra característica es que permite conectar sistemas distintos entre sí con el objeto de formar uno solo, mediante la provisión de una estructura de mensajería abierta, escalable y fuerte. Además proporciona una integración flexible y fiable entre cualquier tipo de aplicaciones, así como también adaptar los flujos de información entre los usuarios según sus necesidades. Esta herramienta permite optimizar la integración de las aplicaciones con los procesos de flujo de trabajo.

Se puede decir que esta herramienta es una plataforma de *software* completa para *e-business*, debido a que abarca la integración, escalabilidad, flexibilidad y compatibilidad.

Con esta herramienta se puede construir una aplicación de acuerdo con los objetivos, estrategias y otros factores, sin enfrentar problemas de incompatibilidad y sin perder tiempo tratando de integrar diversas tecnologías.

Además se pueden desarrollar e integrar aplicaciones *e-business*, como aplicaciones para *business to business* (B2B) que van más allá de simplemente publicar información en la *Web* y transacciones.

Esta herramienta cuenta con divisiones, estas divisiones son grupos de herramientas con un fin específico y que en conjunto solucionan un problema de aplicación, integrándolo como que fuera una sola herramienta, esta son:

- Bases y Herramientas: responde a en una base de alta calidad para rápidamente construir e implementar aplicaciones para *e-business* de alto desempeño sobre demanda.

Sirve para construir, implementar y administrar *e-business*. El núcleo de cualquier *e-business* es colocar sus procesos y transacciones de negocio en la *Web*, para que los clientes, socios y empleados puedan comprar, vender, y compartir informaciones en cualquier momento, en cualquier lugar. La Base de Infraestructura *WebSphere* para *e-business* está toda dirigida a habilitar los negocios en la *Web* de una forma rápida y flexible, permitiendo una infraestructura confiable y escalable construida.

- Portales de Negocios: provee acceso a información personalizada a un variado espectro de usuarios y a través de diversos dispositivos móviles.

Cuando el *e-business* (comercio electrónico) de una empresa empieza a crecer, esta herramienta permite administrar la información que maneja el cliente en *Internet*, para asegurar una operación tranquila y un así mantener la satisfacción de los clientes logrando un crecimiento continuo y escalable del negocio de la empresa, mientras se habilitan las transacciones comerciales y es transparente para el cliente.

- Integración de Negocios: para integrar aplicaciones y automatizar los procesos de negocio para obtener eficiencia operativa y flexibilidad en los negocios.

La Integración de Negocios es el corazón del *e-business* sobre demanda. Trata de integrar datos, aplicaciones, procesos y personas, dándole la posibilidad de apalancar las inversiones existentes en TI mientras proporciona la flexibilidad de adaptarse rápidamente a condiciones cambiantes de los negocios. *WebSphere* permite optimizar operaciones y obtener más flexibilidad con aplicaciones integradas y procesos automatizados.

- Servidor de Transacción y Herramientas:

El cambio de sistemas centrales tradicionales, aplicaciones y datos, permite a un usuario mejorar si el cambio es conveniente, por lo tanto esta herramienta provee esto ya que acorta el tiempo de implementación para nuevas aplicaciones de negocio y crear ventajas competitivas reales.

3.2.1. Enfoque de soluciones de la herramienta

Esta herramienta esta enfocada en la solución primaria de los siguientes factores:

- **e-commerce (comercio electrónico):** venta de mercancías y servicios online a un mercado global y móvil, con B2C, B2B o modelos de negocio de transacciones privadas.
- **Integración Cruzada de Negocios entre Industrias y Soluciones Industriales:** es un mercado específico, que necesitan estándares de datos y procesos de negocios, para obtener resultados más rápidos, más fáciles y más eficaces.
- **Soluciones de Integración B2B:** conexiones a servicios *Web* para solucionar la intercomunicación entre empresas en la realización de negocios.
- **Portal:** crear un único punto de interacción con informaciones, aplicaciones y procesos dinámicos.

3.3. Pasos para modelar en la herramienta XDE

Con esto nos referimos a que podemos implementar en esta herramienta, o sea que tenemos que conocer de ella para proceder a la realización de lo que deseamos. Primero conoceremos los tipos de entorno con lo que cuenta la herramienta, luego conoceremos que modelos se pueden realizar con la misma, para entender la interrelación que tiene la herramienta con la logística que presenta el UML en la elaboración de los diagramas en la resolución de proyectos de *software*. Es necesario aclarar que en este trabajo se presenta XDE para *Java*, pero también existe una versión para *.Net*

3.3.1. Tipos de entornos

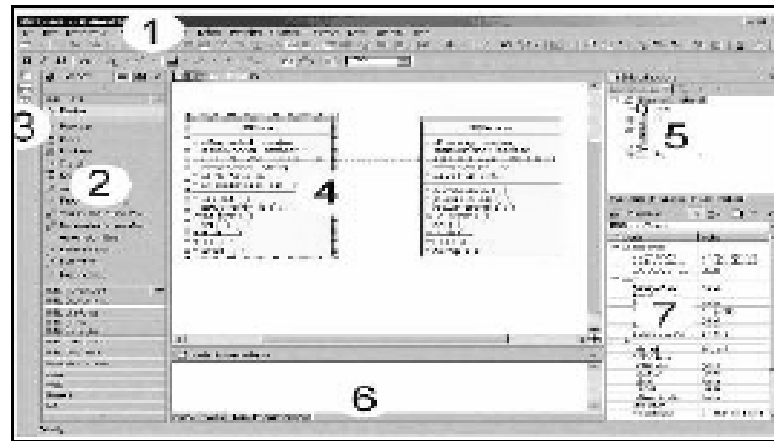
Para el modelado en XDE se proporcionan diversos entornos, en los cuales se podrán realizar los diagramas, conjuntamente con su interrelación con código al mismo tiempo, con lo cual se podrá visualizar de ambas maneras un modelo, tanto de código como de diagramas, se pueden tener 3 tipos de entornos:

3.3.1.1. Entorno de modelado

En este entorno se podrán realizar los modelos mediante los diagramas, o sea, se pueden realizar los diagramas de nuestro modelo para proceder a insertar el código necesario para la resolución de nuestro propósito. El entorno de modelado ofrece las siguientes características (ver figura 31 en la cual solo se ejemplifica la ubicación de las partes):

1. Menús y barras de herramientas
2. *Toolbox* para especificar elementos de modelado
3. Vistas. Permiten cambiar la perspectiva de nuestro proyecto.
4. Área de diagramas donde trabajaremos normalmente
5. Área de vistas. Permite ver la estructura de las diferentes vistas.
6. Documentación de elementos del modelo
7. Editor de propiedades

Figura 31. Entorno de modelado

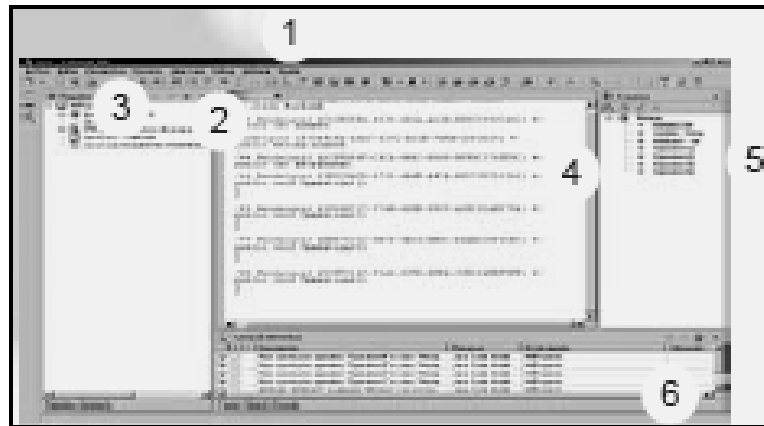


3.3.1.2. Entorno de desarrollo

Este entorno nos permite la inserción del código o programación de lo necesario a elaborar, aquí se ofrecen todas las herramientas a utilizar para una elaboración fácil y rápida de código. El entorno de desarrollo ofrece las siguientes características (ver figura 32 en la cual solo se ejemplifica la ubicación de las partes):

1. Menús y barras de herramientas
2. Estructura de paquetes *Java*
3. Vistas. Permiten cambiar la perspectiva de nuestro proyecto.
4. Editor de código *Java*
5. Esquema de estructura de clase
6. Vista de Tareas

Figura 32. Entorno de desarrollo

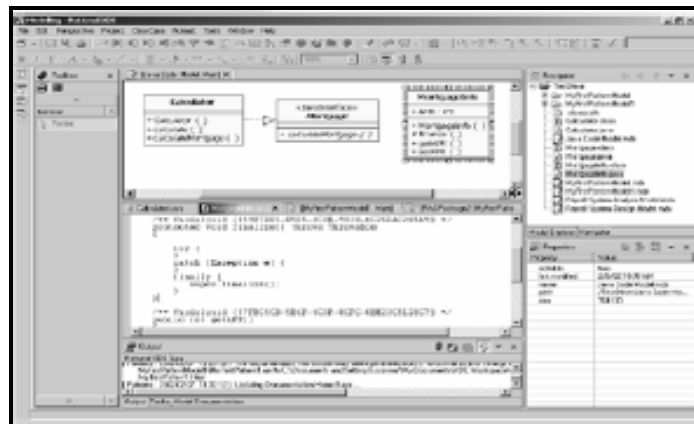


3.3.1.3. Entorno mixto

Este entorno ofrece las características de los entornos de modelado y de desarrollo, ofreciendo las siguientes características o ventajas (ver figura 33 en la cual solo se ejemplifica la ubicación de las partes):

- Las vistas son totalmente configurables.
- Podemos visualizar código a la vez que el modelado.

Figura 33. Entorno mixto



Se tiene una característica fundamental en la utilización de XDE, y es que las vistas están sincronizadas, eso quiere decir que cualquier modificación en cualquiera de las vistas se verá reflejada en la otra, esto nos favorece, ya que no tenemos que modificar ambas vistas por aparte para no perder la relación entre nuestro modelo y nuestro código.

3.3.2. Tipos de modelos

Al tener creado un proyecto en la herramienta, para poder trabajar en el, se puede proceder a la elaboración de los diagramas correspondientes a un modelo, la herramienta XDE ofrece 3 tipos distintos de modelos (ver figura 34), en base a los cuales se tiene que trabajar, estos son:

a) *Java*

- *Java Code Model.* Apropiado para EJB, *JavaBeans*, *servlets* y otros elementos del lenguaje
- *Java Content Model.* Cuando son elementos que no necesitan de la generación de código. Ej. Definición de patrones.

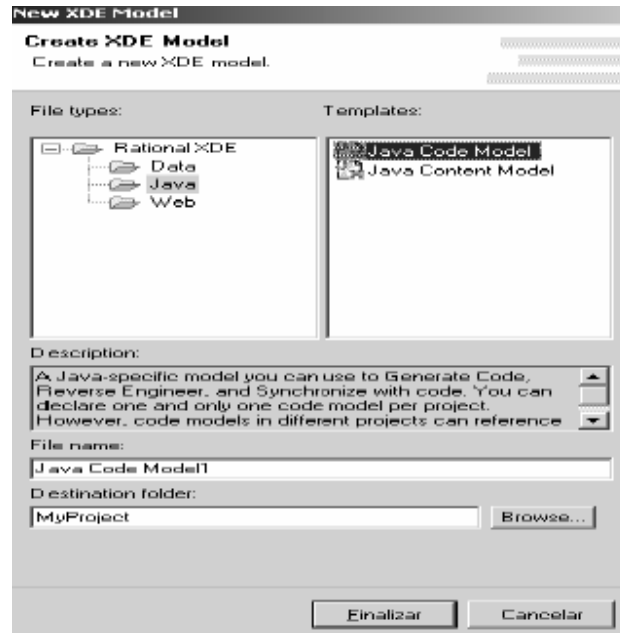
b) *Web*

- *JSP Tag Library Model.* Para modelar *tags* JSP.
- *Virtual Directory Model.* Para modelado de elementos *Web* como páginas HTML y JSP.

c) *Data*

- Modelos de Datos

Figura 34. Crear modelo en XDE



3.3.3. Modelado de UML en XDE

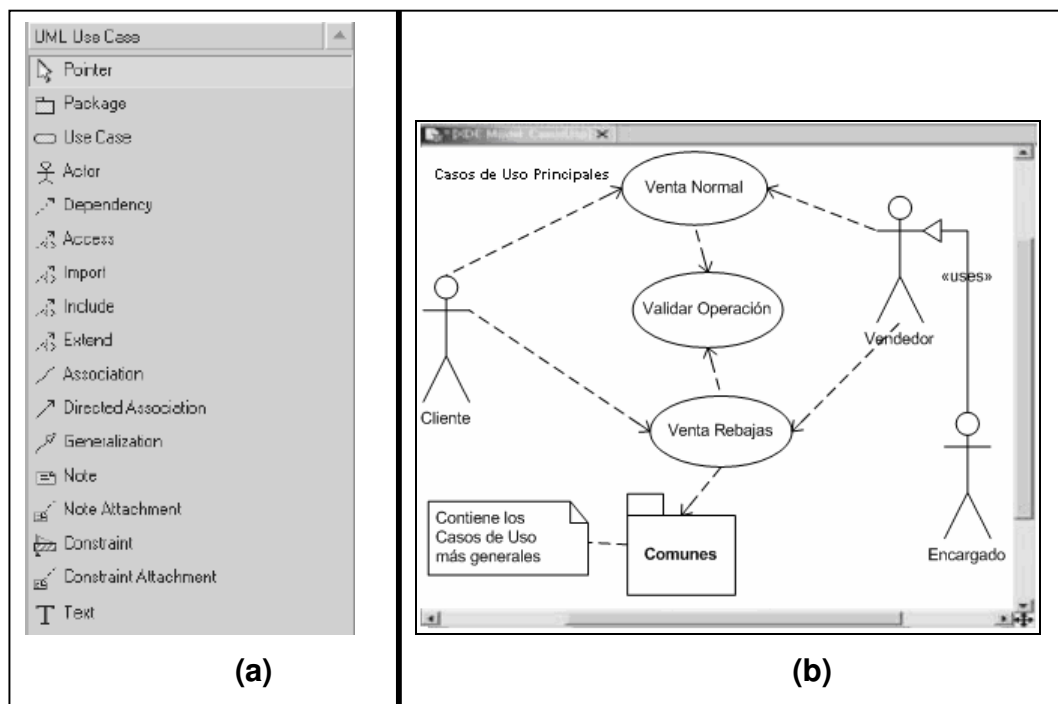
Como todo esto tiene sus bases en el UML, la herramienta cuenta con la logística que este lenguaje presenta y por lo cual se puede entender de mejor manera la interrelación del RUP con UML, ya que como se menciona en el capítulo 1, el enlace entre el RUP y UML, se podrá observar la implementación de lo antes explicado, todo esto en una herramienta capaz de relacionar estos lenguajes y encontrar su funcionalidad.

Se tiene que tener en cuenta que para el modelado, tenemos que tener lo que deseamos hacer, por lo tanto se pueden construir los diferentes diagramas a utilizar, a continuación se presenta como se pueden realizar los diagramas de casos de uso, de clases, de interacciones, de estados, de actividades, de componentes, de implementación.

3.3.3.1. Casos de uso en XDE y sus diagramas

La barra de herramientas de los Casos de uso nos presenta todos los elementos de los diagramas de casos de uso, la figura 35a) muestra el modo de presentar las herramientas para la elaboración de los diagramas de casos de uso, con este menú podemos utilizar a nuestro gusto los elementos con los cuales damos vida a estos diagramas. El diagrama de Casos de Uso (figura 35b) no nos restringe a solo usar elementos definidos para este diagrama. Podemos usar otros elementos, si estos son compatibles, además podemos modificar las propiedades de los elementos definidos por medio de la ventana de propiedades.

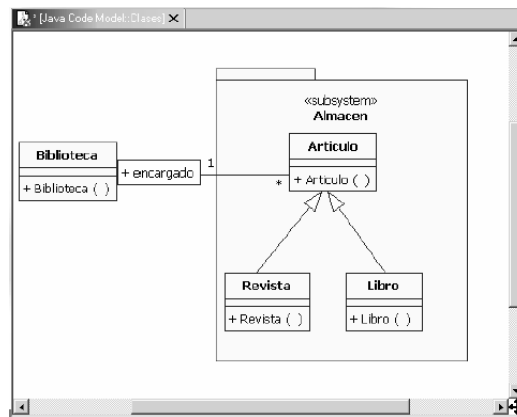
Figura 35. Barra de objetos para crear casos de uso (a) y caso de uso creado en XDE (b)



3.3.3.2. Diagrama de clases con XDE

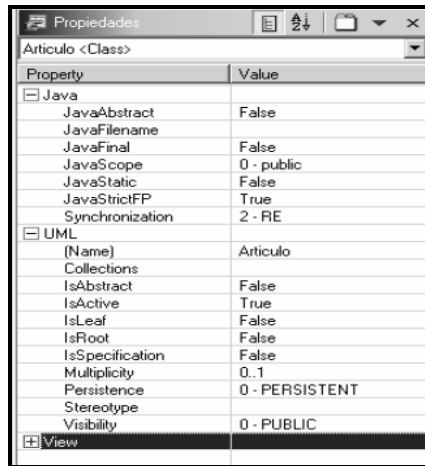
Muestra el conjunto de clases y objetos importantes que forman la estructura estática de un sistema, junto con las acciones existentes entre estas clases y objetos. Una clase describe un conjunto de objetos con características y comportamiento idéntico. En la figura 36 se muestra el modelado de un diagrama de clases en XDE.

Figura 36. Diagrama de clases en XDE



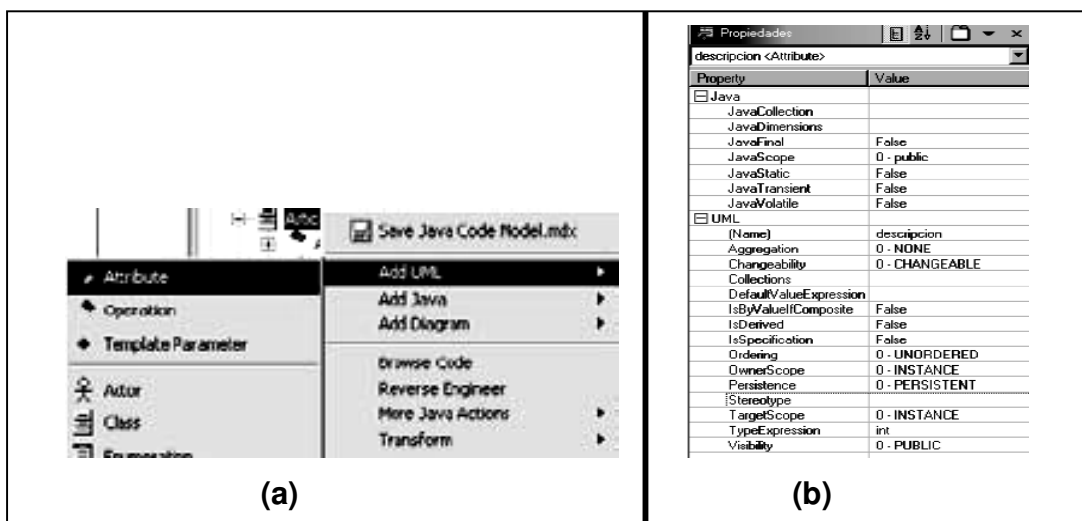
En la creación de una clase podemos indicar si la clase es: Abstracta, Final, Ámbito, Estática, Activa, Multiplicidad, Persistencia, Estereotipos, Visibilidad, etc. Algunas propiedades se pueden modificar desde las propiedades de UML o de las de *Java* como ocurre con otras propiedades con sentido en ambos estándares, para ello utilizamos la ventana de propiedades mostradas en la figura 37.

Figura 37. Ventana de propiedades para el diagrama de clases en XDE



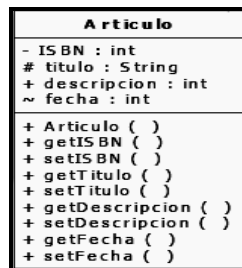
A partir de la clase podemos crear atributos. En la ventana de propiedades podemos modificar las propiedades del atributo, entre estas tenemos: Visibilidad, Estático, Multiplicidad, Persistencia, etc., como podemos observar en las figuras 38a y 38b.

Figura 38. Creación de atributos (a) y ventana de propiedades para los atributos (b) utilizados en los diagramas de clases en XDE



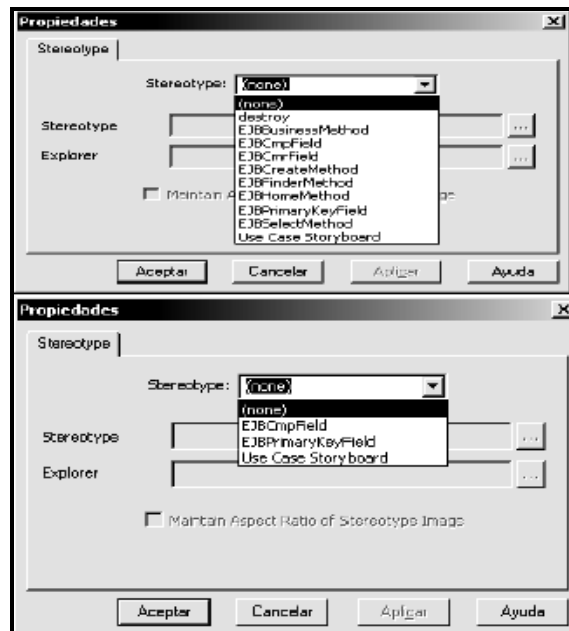
Por medio de la ventana de propiedades podemos indicar la visibilidad de los atributos y métodos, entre estos tenemos: - Privado, + Publico, # Protegido, ~ Paquete, como se muestra en la figura 39.

Figura 39. Visibilidad de atributos y métodos de una clase utilizada en un diagrama de clases en XDE



Podemos definir nuestros propios estereotipos o utilizar los definidos por defecto, esta definición la podemos observar de mejor manera en la figura 40.

Figura 40. Definición de estereotipos para un diagrama de clases en XDE



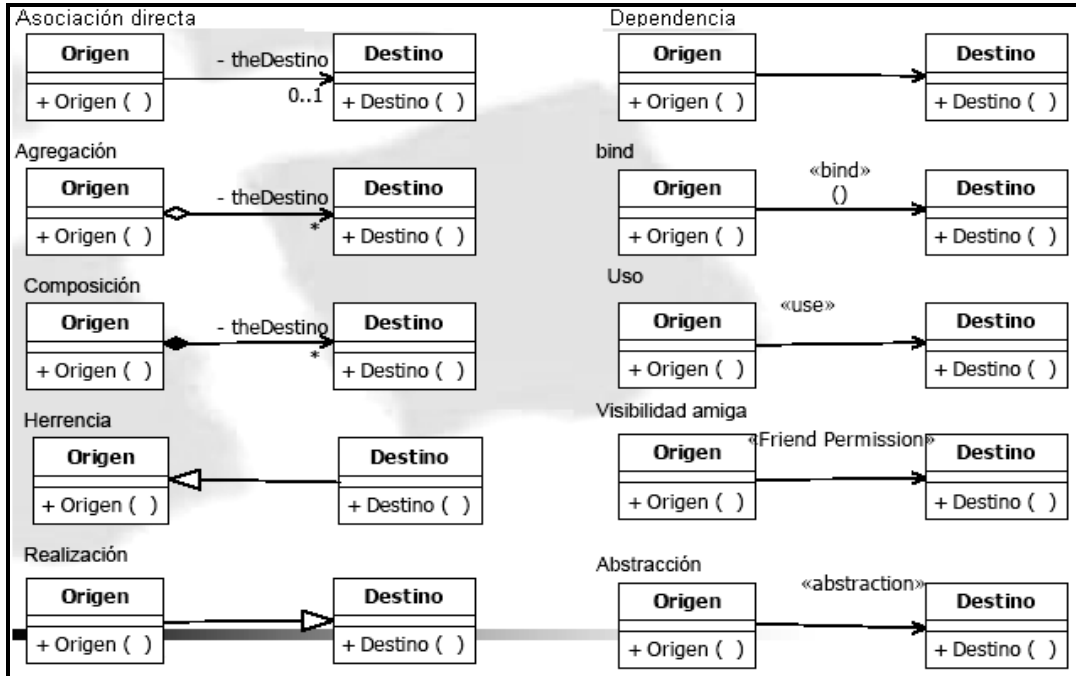
Una línea que une dos o más artefactos. Pueden tener varios tipos de adornos, que definen su semántica y características, como lo son: Asociación binaria, Composición, agregación, Generalización, en la figura 41 se muestra la manera de presentar estas opciones para ser elegidas.

Figura 41. Líneas para unir artefactos utilizadas en un diagrama de clases en XDE



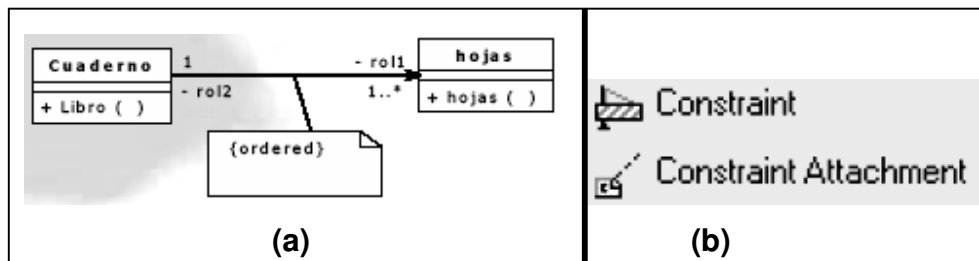
Las relaciones también tienen sus propiedades divididas en cada uno de los extremos de la relación, como las siguientes: Posibilidad de cambio, Navegación, Multiplicidad, Tipo, Ámbito, Visibilidad, Persistencia, Rol, etc. Estas relaciones se muestran de una manera más entendible en la figura 42.

Figura 42. Relaciones entre artefactos, utilizadas en un diagrama de clases en XDE



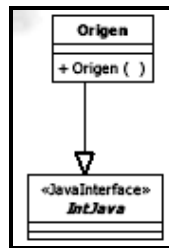
Podemos indicar restricciones a cualquier elemento, la manera de realizarlo es eligiendo la opción de restricción y luego diagramarla, estas se presentan como en la figura 43.

Figura 43. Restricciones entre artefactos, utilizadas en un diagrama de clases en XDE, (a) vista modelada de la restricción (b) selección de la restricción



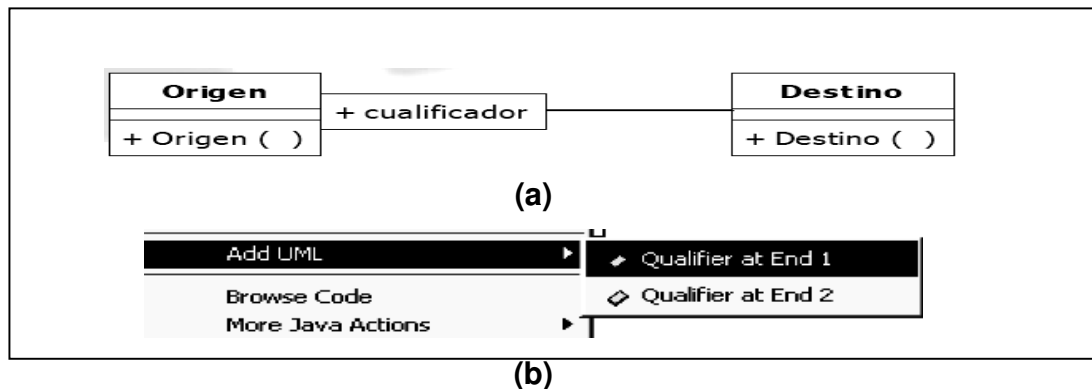
Los interfaces se representan por medio de clases estereotipadas y están restringidos al concepto de interfase por lo que no podemos realizar todas las operaciones que tenemos disponibles en las clases, además se puede utilizar el interfase UML o de *Java*, pero si estamos interesados en la generación de código es vital que usemos el interfaz *java*. En la figura 44 se muestra como se diagrama una interfase y como quedaría representada.

Figura 44. Diagrama de una interfase utilizado en un diagrama de clases en XDE



Para añadir un cualificador a una relación es simple, únicamente se selecciona y se diagrama, en la figura 45 se muestra la manera como se realiza tal opción y como quedaría diagramado en el entorno de modelado:

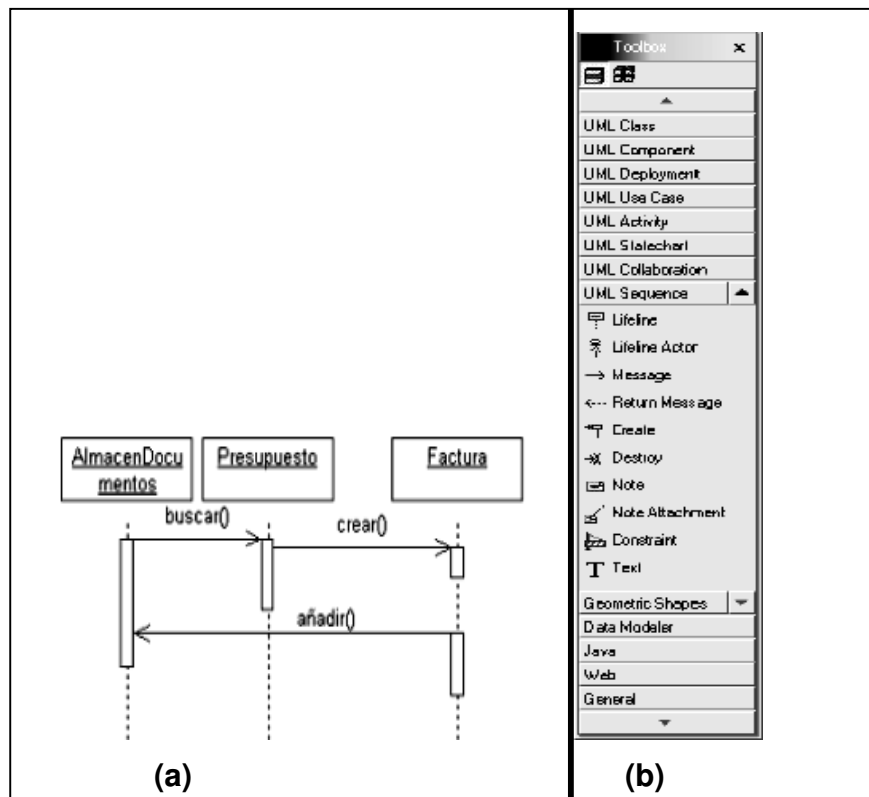
Figura 45. Cualificadores entre artefactos, utilizados en un diagrama de clases en XDE, (a) vista modelada de la cualificación (b) selección de la cualificación



3.3.3.3. Diagramas de secuencia

Muestra una interacción a lo largo del tiempo, además muestra las instancias participantes en una interacción y su “línea de vida”, con la excepción de no mostrar las asociaciones entre objetos, a continuación se presentan en las figuras 46a y 46b la manera de elegir la elaboración de un diagrama de secuencia, por medio del *toolbox*, proporcionado por XDE, y los diversos objetos que conforman este diagrama




Figura 46. Diagrama de secuencia (a) y barra de objetos para crear diagramas de secuencia (b)



Los diagramas de secuencias pueden definir el comportamiento de una colaboración o, el comportamiento de una instancia de una colaboración. Así usaremos un tipo u otro: *Sequence: Instance* ó *Sequence: Role*.

Se pueden añadir objetos anónimos, sin nombre de clase, así como también se pueden arrastrar clases, o actores, definidos al diagrama o a un objeto, con el objetivo de diagramarlo.

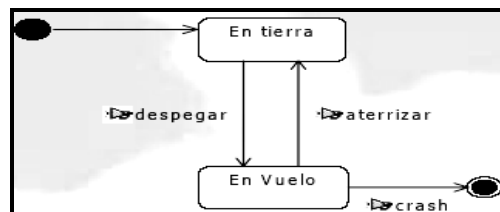
Es de suma importancia la inclusión de mensajes dentro de estos diagramas, para ello se utilizan los símbolos siguientes dependiendo del uso que se le va a dar, ya sea:

Llamada a procedimiento	
Llamada asíncrona	
Retorno de una llamada a procedimiento	

3.3.3.4. Diagrama de estados con XDE

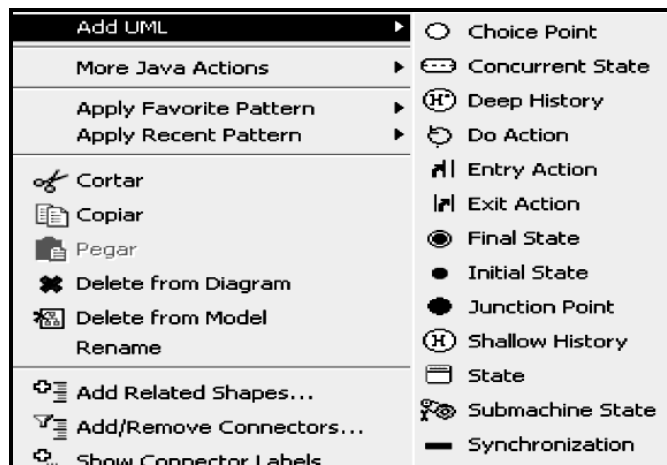
Los diagramas de estado sirven para representar aspectos dinámicos de una clase aislada representando sus estados y sus transiciones ó de un caso de uso, en la figura 47 se muestra un diagrama que dependiendo en que estado se este, se van a tener transiciones para poder migrar hacia el otro estado.

Figura 47: Diagrama de estados en XDE



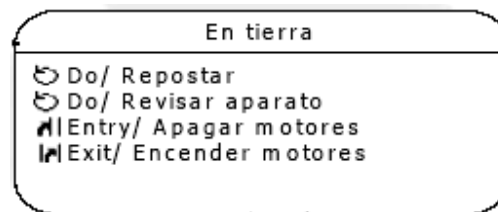
Al momento de crear un estado se representa el comportamiento de las entidades con contenido dinámico interesante; un estado puede contener numerosos elementos, así como sub-elementos, solamente seleccionando uno, podemos optar por estos. En la figura 48 se muestra la manera de optar por la creación de un estado, únicamente seleccionando “State” de la lista mostrada en esta figura.

Figura 48. Forma de crear un estado, utilizado en un diagrama de estados en XDE



Al momento de tener un estado, se le pueden crear acciones de entrada, de salida y de hacer, en la figura 49 se muestra un estado con estas acciones.

Figura 49. Acciones creadas a un estado, utilizado en un diagrama de estados en XDE



Para poder cambiar de un estado a otro se utilizan las transiciones, las cuales son definidas como conexión entre dos estados o hacia el mismo estado. También se cuentan con los eventos, los cuales son la ocurrencia que puede causar la transición de un estado a otro de un objeto. Por medio de las propiedades podemos indicar la condición de guarda y la acción a realizar para una transición. Entre estos eventos se tienen varios tipos:

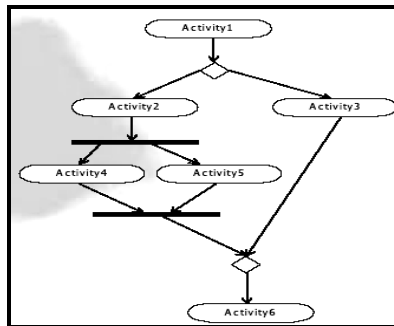
- *Call Event*.- Cuando el evento se produce por la llamada de un objeto a una operación que implementa la transición.
- *Change Event*.- Se produce cuando se da una condición definida (*true*, *false*)
- *Signal Event*.- Se produce cuando un objeto recibe una señal de otro objeto.
- *Time Event*.- Cuando el evento se produce a causa del paso de un periodo de tiempo, o por la llegada a un momento concreto (hora, fecha)

Para añadir eventos a una transición se utiliza el menú *popup*, que es similar a todos los menús que se ha utilizado para la elección de diagramas.

3.3.3.5. Diagrama de actividades con XDE

Un diagrama de actividades es un caso especial de un diagrama de estados en el cual casi todos los estados son estados de acción y casi todas las transiciones son enviadas al terminar la acción ejecutada en el estado anterior. En la figura 50 se muestra un diagrama de actividad, con todas sus partes

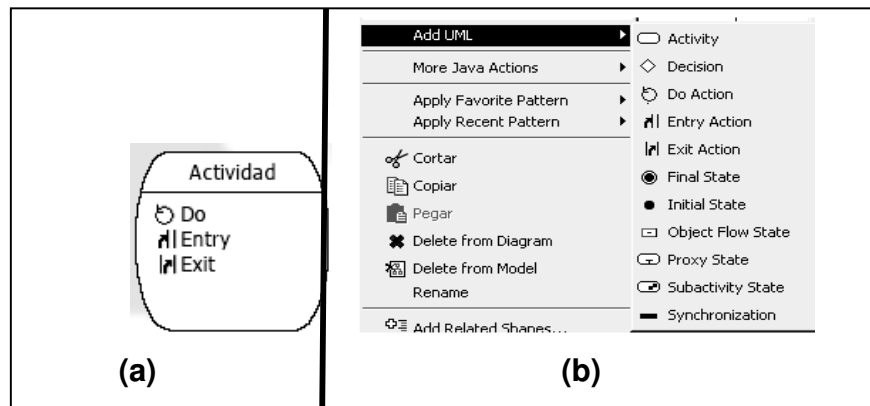
Figura 50. Diagrama de actividades en XDE



Para crear una actividad únicamente se selecciona el tipo de objeto a utilizar y se diagrama. Se entiende como actividad a secuencias ejecutables de un objeto que representan la realización de una acción, se considera que no se invierte tiempo en realizarlas y no pueden ser interrumpidas.

Para una actividad (figura 51a) podemos crear acciones de la misma forma que en los Estados, estas acciones son igualmente proporcionadas en la barra de objetos de diagrama de actividad, en la figura 51b se muestra la manera de seleccionar los objetos de este diagrama, las acciones, así como también la forma de percibir una actividad diagramada.

Figura 51. Modelado de una actividad (a) y forma de crear una actividad utilizada en un diagrama de actividades en XDE



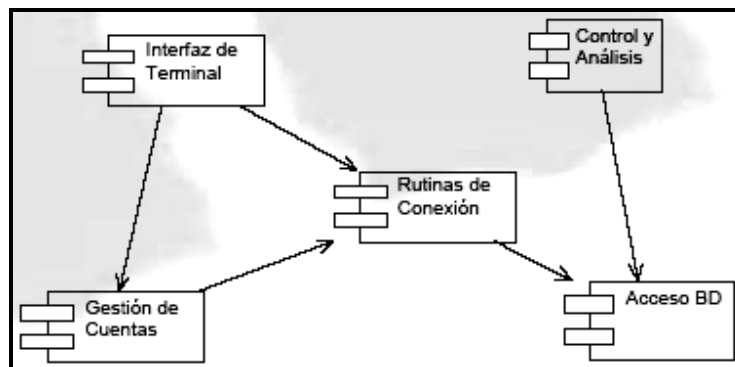
Dentro de las actividades podemos crear sub-actividades, así como también podemos crear estados de sub-actividades, únicamente se tiene que arrastrar la sub-actividad de estados definida desde el Explorador del modelo al “*SubActivity State*”.

Al momento de diagramar una actividad, también es de suma importancia la utilización de transiciones, bifurcaciones y barras de sincronización. Entendemos como transiciones al camino entre un estado de actividad o estado de acción al siguiente, también entendemos como bifurcación a la condición booleana de transito, y como barra de sincronización a la división o unión del flujo de caminos en el diagrama.

3.3.3.6. Diagramas de componentes con XDE

Estos diagramas describen componentes del *software* y las dependencias de unos con otros representando la estructura del código, estos componentes son típicamente los ficheros de implementación en el entorno de desarrollo, en la figura 52 se muestra un diagrama de componentes elaborado en la herramienta.

Figura 52. Diagrama de componentes en XDE



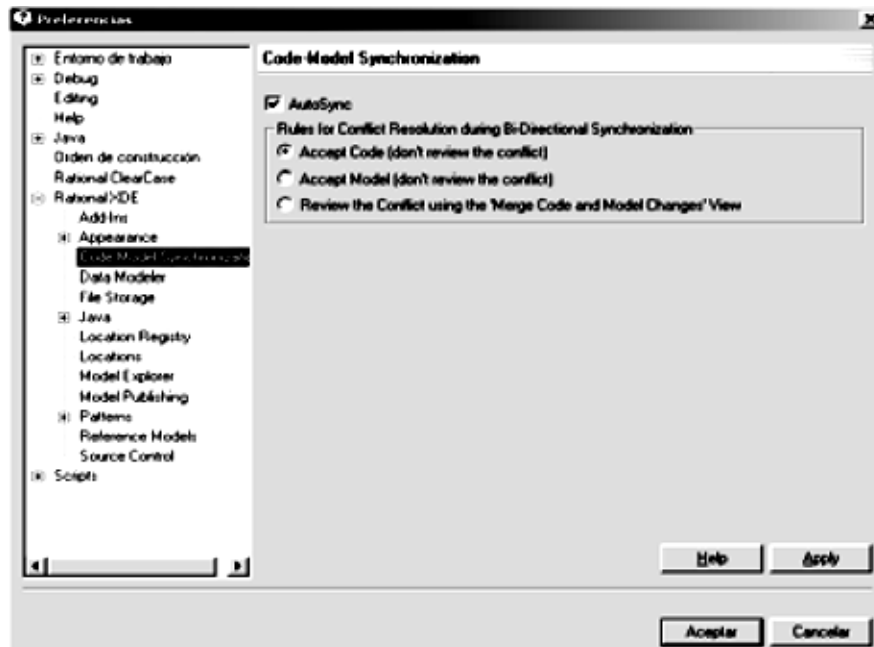
Es una gran ventaja la elaboración de componentes, ya que en el Modelado *Java* existe una relación entre clases y componentes, ya que al crear una clase también se crea su componente de forma automática.

3.4. Generación de código

Para la generación de código se tiene la opción en el menú, juntamente con otras opciones para el manejo del código.

AutoSync realiza la generación de código de forma automática, sincronizando el modelo automáticamente con el código, de manera tal que si este cambia, también lo hará el código, en la figura 53 se muestra la forma de seleccionar esta opción.

Figura 53. Selección de *AutoSync* para generar código automáticamente en XDE



Además para mejorar el rendimiento podemos eliminar la construcción automática.

Se tienen también preferencias para el código, como el estilo que va a tener el código generado, también se cuenta con la opción *Assisted Modeling* que sirve para establecer que elementos queremos que se creen de forma automática al crear las clases, en la figura 54 se muestra la manera de elegir esta opción.

Figura 54. Selección del estilo de código a generar en XDE

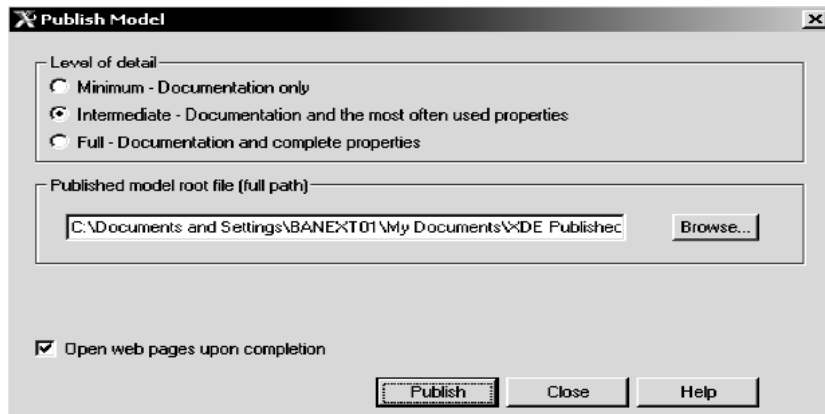


Podemos también generar código basado en plantillas predefinidas, a estas plantillas se les llama “Plantillas de Código”, y son enlazadas con el modelo por medio de un “*binding*”. Al crear la plantilla se tiene que especificar: Nombre, Descripción y Lenguaje de desarrollo a utilizar.

3.5. Publicación de código

La publicación de código no es más que la puesta en práctica de todo lo mencionado, ya que al momento de crear modelos o utilizar plantillas o generar código mediante modelos, en fin, al momento de tener todo nuestro proyecto elaborado, procedemos entonces a su publicación para utilizarlo, esto se hace por medio de un *wizard*, en el cual definimos lo que deseamos, en la figura 55 se muestra un ejemplo de esto.

Figura 55. Publicación de código en XDE



4. EXPLICACIÓN ASOCIATIVA DE LAS METODOLOGÍAS ESTÁNDARES Y HERRAMIENTAS PARA EL DESARROLLO RÁPIDO DE APLICACIONES

4.1. Título del sistema

Sistema de Subastas en Línea de Vehículos

4.2. Descripción general del sistema

El siguiente es un ejemplo de la implementación de un sistema capaz de ofrecer subastas de automóviles por medio de *Internet* a múltiples compradores con la posibilidad de poner ofertas sobre los automóviles en subasta.

Este ejemplo fue creado para demostrar el uso de las Herramientas *Rational*, la Tecnología J2EE y la metodología RUP en un Desarrollo Rápido de Aplicaciones, y poder así crear una aplicación *Web* en un dominio al cual tenga acceso cualquier usuario. Este sistema también ejemplifica una solución general para los problemas del Comercio Electrónico en el área general de la Arquitectura de Aplicación J2EE.

Debido a que la implementación completa del sistema es demasiado extensa, únicamente se realizará cierta parte del mismo, pero con la cual se podrán entender y alcanzar los objetivos planteados.

Se realizará el análisis y diseño del sistema, utilizando cierto Caso de Uso y omitiendo otros que únicamente extenderían el ejemplo.

A continuación se presenta un análisis corto hecho que incluye los requerimientos captados así como otras especificaciones que servirán para la implementación del sistema. Aquí se estarán aplicando un Desarrollo Rápido en el ámbito de Análisis del sistema.

4.3. Requerimientos a satisfacer

A continuación se presenta una lista de requerimientos de los cuales se partió para la implementación del sistema, aunque estos requerimientos definen de manera completa el sistema, recalamos que en este ejemplo únicamente se incluirán ciertas partes para motivos de explicación:

- 👉 Los vendedores necesitan alcanzar el mayor número posible de compradores para obtener competitividad de ofertas de compra sobre los vehículos.

- 👉 Los compradores deben poder tener acceso a los diversos catálogos de vehículos y mantener el control sobre la cantidad de dinero que esta dispuesto a pagar por esos vehículos. Así como también poder dar su oferta sobre ese vehículo.

- ☞ Ambos, vendedores y compradores desean que sean anónimas sus transacciones hasta el momento de finalizar la transacción de intercambio.

- ☞ El sitio *Web* de subastas debe proveer un ambiente en el cual tanto vendedores y compradores satisfagan frecuente, anónima y completamente los intercambios, además debe de proveer un beneficio por los intercambios realizados.

- ☞ El sistema debe proveer un ambiente seguro y virtual en el cual los vendedores puedan anunciar sus vehículos, los compradores puedan comprarlos y el administrador pueda de una manera efectiva dar soporte a esta red de mercado recuperando sus costos y obteniendo un beneficio. Así mismo manejar la autenticación de usuarios, y así estar seguro de permitir únicamente las operaciones a las cuales tiene autorización dicho usuario.

- ☞ Permitir crear cuentas a usuarios y poder modificar la información de dichas cuentas.

- ☞ Los vendedores puedan crear una subasta describiendo el vehículo, definiendo el precio inicial y el incremento de la oferta, pudiendo también incluir fotos de los vehículos.

- ☞ Los vendedores pueden rechazar las ofertas o aceptar las mismas; cuando acepte la oferta el estatus de la subasta deberá cambiar de abierta a cerrada y deberá empezar la transacción de intercambio.

4.3.1. Otros requerimientos a satisfacer

El Sistema de Subastas en Línea de Vehículos deberá ser desplegado en un ambiente J2EE, corriendo en un dominio público y depositando sus componentes en un Contenedor *Web*, Contenedor EJB y una base de datos. La aplicación deberá ser desarrollada utilizando los productos *Rational* y siguiendo el RUP, utilizando el Desarrollo Rápido en *Rational*. De la parte Arquitectónica solo mostrar los diagramas y enlaces, así como características necesarias para comprender la solución.

4.4. Stakeholder y descripciones de usuarios

Los *Stakeholder*, son las personas u organizaciones que están directamente envueltas en la elaboración o tomas de decisiones claves a cerca de la funcionalidad y propiedades del Sistema. Para este sistema los *Stakeholders* son: Dueño del Sitio de Subastas en Línea y Desarrolladores del Sistema de Subastas.

Los usuarios son las entidades individuales que utilizan el Sistema de Subastas de Automóviles, estos son: Vendedores, Compradores, Visitantes Casuales y Administrador.

4.5. Ambiente a utilizar

Se asume que los usuarios accesarán y utilizarán el Sistema vía *Internet* mediante un navegador *Web*. También se asume que el Administrador trabajará dentro de una red privada (la misma donde estará instalado el sistema de servidores).

4.6. Modelo de casos de uso

Se identificaron los siguientes casos de uso:

- 👉 *Oferta en Vehículo*: Utilizado cuándo el comprador puede optar a poner una oferta sobre el vehículo.
- 👉 *Ver Catálogo de Subasta*: Utilizado para visualizar el catálogo de vehículos a subastar.
- 👉 *Cerrar Subasta*: Sirve para cerrar una subasta, por distintos motivos, ya sea cancelarla o simplemente alguien compró el vehículo.
- 👉 *Crear Subasta*: Sirve para que un Vendedor pueda crear una subasta.
- 👉 *Crear Cuenta*: El usuario puede crear cuentas para realizar los pagos de las subastas.
- 👉 *Administrar Cuenta*: Todo lo concerniente a la administración de las cuentas.
- 👉 *Entrar Al Sistema*: Utilizado para poder ingresar al sistema y poder optar a realizar ofertas sobre las subastas existentes.

Entre los actores se identificaron los siguientes: Vendedor, Usuario, Comprador, FinTiempoDeSubasta, Oficina de Servicio de Crédito (Estos dos últimos son otros sistemas con los que se tiene interrelación).

Como hemos explicado con anterioridad no se implementarán todos los casos de uso, pero si una interrelación entre ellos, y luego se utilizará el Caso de Uso Oferta en Vehículo y su Flujo Básico, con el cual se podrá ejemplificar toda la secuencia de pasos necesarios hasta llegar a entender la interrelación de las Herramientas y Metodologías expuestas en este trabajo.

En la figura 56 se presenta el modelo de Casos de Uso General que se implementó para dar solución al problema. En este modelo se muestran las interrelaciones que existen entre los diversos Casos de Uso y los Actores que interactúan en el mismo.

A continuación se desglosará un análisis completo del caso de uso Oferta en Vehículo, con el cual se ejemplificará la interrelación de las metodologías, estándares y herramientas.

4.7. Análisis del caso de uso Oferta en Vehículo

Lo primero que tenemos que tener claro es que al pasar al análisis de este caso de uso se va a tener la Realización del Caso de Uso, esto quiere decir que se va a trabajar el Caso de Uso en cuestión, mostrando los diversos diagramas vistos con anterioridad (Secuencia, Colaboración, Clases) y con ellos poder observar con mayor claridad el objetivo del Caso de Uso, la forma de representar la Realización del Caso de Uso es la que se muestra en la figura 57, lo cual indica que ambos están unidos la vista lógica, con el modelo de análisis, con lo cual no se pierde la relación o relaciones antes establecida/s en el modelo de análisis.

Figura 56. Caso de uso sistema de subastas en línea de vehículos

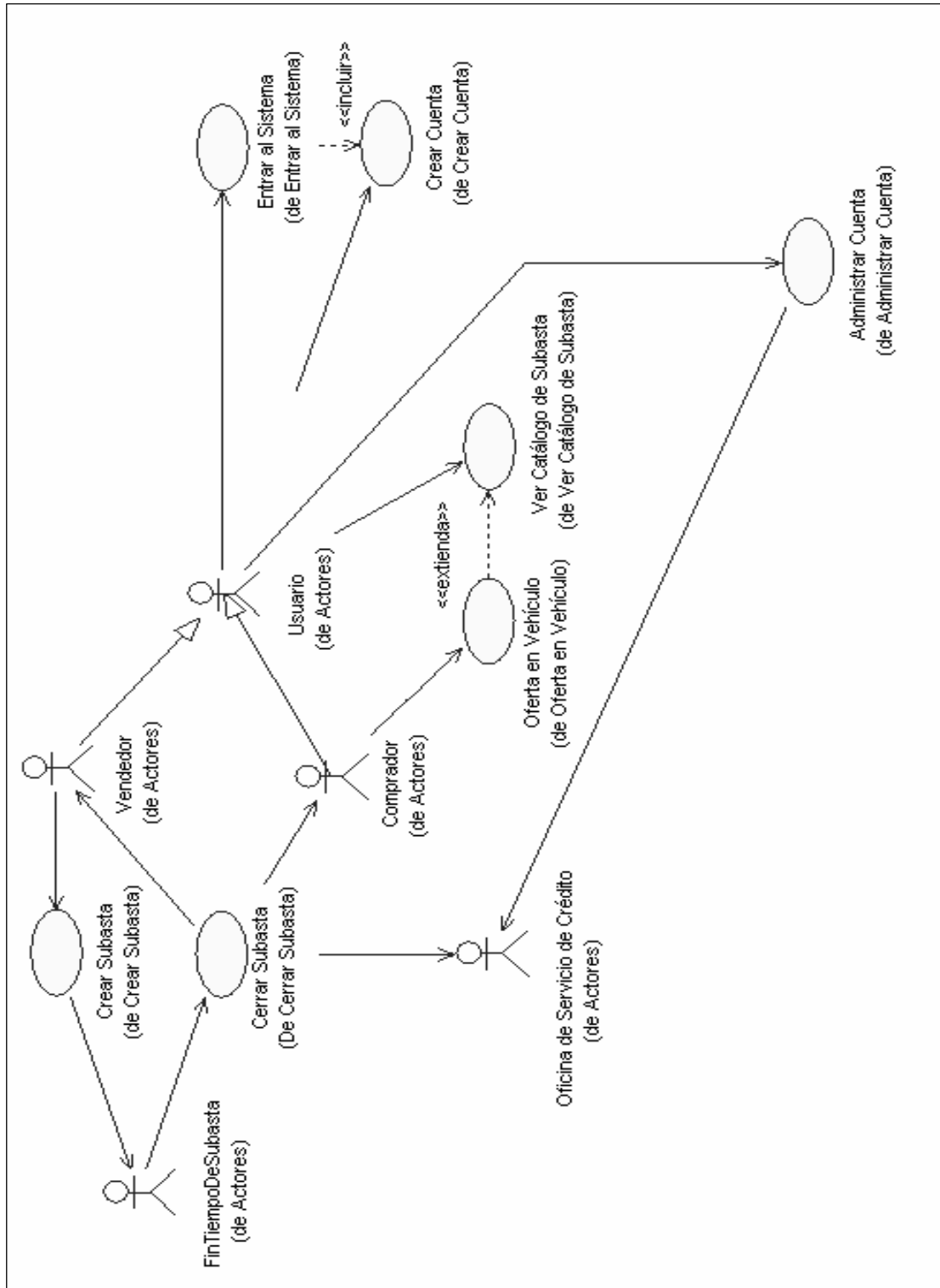
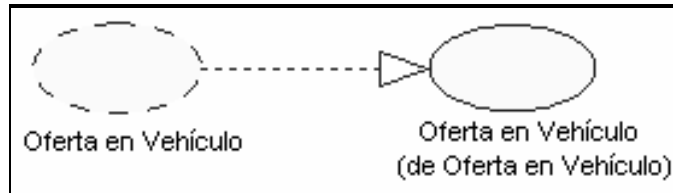


Figura 57. Realización del caso de uso sistema de subastas en línea de vehículos



Para comenzar con la Realización del Caso de Uso, describimos el caso de uso en sí, paso a paso en su flujo básico y sus flujos alternos, del cual se partirá para la realización de los diagramas de secuencia, colaboración y clases.

👉 Descripción del caso de uso

Cuando se está observando un Vehículo disponible en una subasta (con el Caso de Uso Ver Catálogo de Subasta no implementado en este ejemplo), el comprador puede optar a poner una oferta sobre el vehículo, la oferta puesta debe ser mayor que la oferta actual y como mínimo la oferta debe ser mayor que el especificado por el vendedor, una vez aceptada la oferta se realiza una nueva oferta para ver que comprador la puede dar.

El usuario debe estar dentro de la subasta para poder realizar una oferta sobre el vehículo (con el caso de uso Entrar al Sistema no implementado en este ejemplo).

Si la subasta ha estado cerrada, la oferta no es aceptada (con el caso de uso Cerrar Subasta no implementado en este ejemplo).

Si el comprador tiene algunos avisos pendientes de pago, un mensaje será desplegado al comprador, recordándole que el pago debe ser hecho (nueva información de tarjeta de crédito debe ser ingresada) antes que el Usuario pueda participar en cualquier subasta (tanto comprador como vendedor, la información de la nueva tarjeta de crédito puede ser ingresada por medio del Caso de Uso Administrar Cuenta no implementado en este ejemplo).

PRE-Condiciones

- ***Vendedor Entra al Sistema***

El Comprador debe entrar al sistema después que el Vendedor pueda crear una subasta.

- ***El Vendedor ha Exhibido el Vehículo***

El Comprador ha desplegado el vehículo en que está interesado realizar una oferta.

Flujo Básico

1. Este Caso de Uso ocurre cuando el Comprador elije hacer una oferta en un vehículo actualmente disponible en una subasta.
2. El sistema despliega una forma para especificar la cantidad de la oferta.
3. El Comprador ingresa el monto de la oferta. El sistema valida el monto de la oferta. La oferta incorporada debe ser mayor que la existente en un monto mayor que el mínimo incremento especificado para la subasta.

4. El sistema fija la oferta para la subasta. La oferta ingresada se convierte en la oferta actual.
5. El sistema envía un mensaje de correo de confirmación al comprador, confirmando la cantidad de la oferta para un vehículo específico de la subasta, así como la notificación al comprador cuando la subasta se anticipa para ser cerrada.
6. El sistema exhibe un mensaje al comprador, informándole que la oferta se ha incorporado al vehículo. El mensaje debe incluir el nombre del comprador, la dirección de correo electrónico del comprador, el nombre del artículo y la cantidad de la oferta.
7. Continúa el Caso de Uso Ver Catálogo de Subasta donde se quedó.

👉 Flujos Alternos

Subasta es Cerrada

Al principio del caso del uso, si la subasta para el artículo ha estado cerrada, un mensaje se exhibe al comprador que indica que la subasta ha estado cerrada, y la oferta no se acepta. El caso de uso termina.

El Comprador está Pendiente de Pagos

Al principio del caso del uso, si el comprador tiene pagos pendientes, el sistema exhibe un mensaje al comprador, recordándole que tiene pagos pendientes y que mientras no se realicen los pagos no puede participar en una subasta, como un comprador o vendedor. Para que ya no tenga avisos del pago, el comprador debe incorporar la nueva información de la tarjeta de crédito. La información de la tarjeta de crédito se puede ingresar por medio del Caso de Uso Administrar Cuenta no implementado en este ejemplo). El caso de Uso termina (la oferta no es fijada).

Oferta Ingresada es Inválida

Si la oferta incorporada es inválida, el sistema exhibe un mensaje al usuario que explica la razón que la oferta es inválida (Ejemplo: la oferta incorporada no era mayor que la oferta más grande por una cantidad mayor que el incremento de la oferta del mínimo especificado para la subasta). El sistema entonces incita a usuario volver a ingresar la información de la oferta. El flujo básico continúa en el punto donde el comprador incorpora la información de la oferta (véase el paso 3 en el flujo básico).

👉 Post-Condiciones

Una oferta se ha incorporado a la Subasta

La oferta incorporada se convierte en la actual (es decir, el más grande) oferta para la subasta y el comprador ha sido notificado.

4.8. Diagrama de clases del análisis de la realización del caso de uso

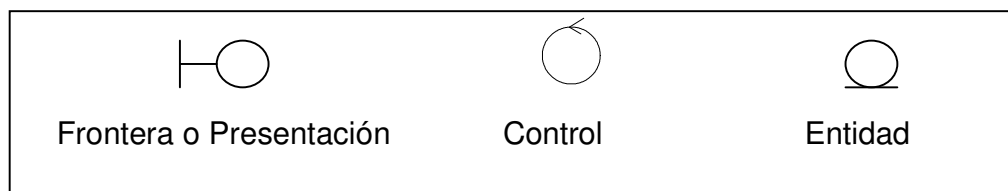
Es de suma importancia mencionar que es distinto hablar del Modelo de Caso de Uso en si y de la Realización del mismo, y para diferenciar esto es preciso entender que un Modelo de Caso de Uso en si muestra los pasos a seguir para llegar a un objetivo, con sus flujos alternos que ejemplifican escenarios o sea diversos caminos por donde se puede dirigir el Caso de Uso, esto es lo que entiende el cliente; lo mismo pasa en la Realización del Caso de Uso, únicamente que en este caso varía la forma de realizar los diagramas ya que se convierte o divide el Caso de Uso en varias clases de diferentes estereotipos, dependiendo de cuantos Entidad (*entity*), Presentación o Frontera (*boundary*) y Control (*control*) se identifiquen..

Una clase entidad suele reflejar datos o entidades del mundo real o sea elementos necesarios dentro de un sistema. Una clase frontera maneja comunicaciones entre el entorno del sistema y el sistema, suelen proporcionar la interfaz del sistema con un usuario o con otro sistema. Una clase control modela el comportamiento secuenciado específico de uno o varios casos de uso. Se trata de clases que coordinan los eventos necesarios para llevar a cabo el comportamiento que se especifica en el caso de uso, representan su dinámica. (Esto fue explicado con anterioridad únicamente se está haciendo un breve recordatorio)

En la figura 59 se muestran las clases participantes en el Caso de Uso en cuestión, así como las diferentes relaciones que existen entre ellas. Estas clases ya están divididas dependiendo de sus funciones, o sea que pueden ser Frontera o Presentación, Control y Entidad según las necesidades encontradas.

Además se puede observar que cada clase representada en el Diagrama de Clases tiene su simbología, como en las figura 58:

Figura 58. Estereotipos



De las clases existentes en todo el sistema, las de la figura 59 son las que tiene lugar en el Caso de Uso Oferta en Vehículo, por tal razón son las participantes dentro del diagrama de Clases de este Caso de Uso.

Este diagrama responde a los requerimientos planteados para este Caso de Uso, y dado que son clases utilizan sus atributos y métodos para la intercomunicación entre ellas, esta intercomunicación está establecida por medio de las líneas que unen las clases, y su funcionalidad se podrá apreciar en los diagramas de secuencia y colaboración, ya que estos demuestran los caminos a seguir para representar los distintos escenarios, (en este ejemplo se recalca que solo se utilizará el escenario compuesto por el Flujo Básico).

Siguiendo con estos estereotipos (Frontera, Control, Entidad) pasaremos de cómo lo entiende el cliente a como lo entiende el desarrollador del sistema, ya que se entra mas directamente en la implementación del sistema pero de una manera genérica, o sea, sin especificar la plataforma de desarrollo ya que esto corresponde directamente a la parte de Diseño del Sistema, la cual veremos posteriormente.

A continuación se explica la funcionalidad de cada una de las clases vistas en la figura 59:

Interfaz Servicio Correo

Administra la interfase con el sistema de correos electrónicos. Este es un solo servicio de correo electrónico, por lo tanto solo es una única interfase de servicio de correo electrónico.

Oferta en Vehículo

Esta es la forma en la cual el Comprador pondrá su oferta sobre un vehículo disponible en subasta.

Sesión

Contiene información sobre una sesión de un Usuario en particular con la subasta. Es creada cuando un Usuario tiene su primer acceso a la subasta y es eliminado dentro de una cantidad de tiempo prudencial o si el usuario no ha tenido actividad. Se crea una instancia de Sesión por cada usuario. Los objetos de sesión son globales.

Control Oferta en Vehículo

Esta clase controla el proceso que ocurre cuando un Comprador desea hacer una oferta en un vehículo.

Forma Información Subasta

Esta forma contiene información detallada sobre un vehículo disponible para subasta, el detalle incluye:

- Información del Vehículo (marca, modelo, descripción, figura)
- Nombre de Usuario del Vendedor o Vendedores
- Tiempo de inicio y duración de la subasta
- Precio de Oferta Actual
- Mínimo incremento de oferta
- Precio inicial
- Numero de ofertas fijadas hasta el momento
- Reseña Histórica de Ofertas (fecha, monto, nombre usuario de comprador interesado)

Subasta

Una venta en la cual un vehículo se vende al licitador más alto. Para este sistema, hay una subasta separada para cada vehículo.

Cuenta Usuario

Contiene información que se mantiene para un usuario en el sistema. Un Usuario maneja su propia lista de Noticias de Pagos Pendientes.

Oferta

Información sobre una oferta que se ha puesto en un vehículo disponible para subasta. Una oferta tiene una cantidad y un comprador.

4.9. Diagrama de secuencia del análisis de la realización del caso de uso

En la figura 60 se observa un diagrama de secuencia que muestra los pasos de la Realización del Caso de Uso Oferta en Vehículo, con sus respectivos estereotipos (Frontera o Presentación, Control y Entidad).

Este Diagrama de Secuencia utiliza los métodos ya establecidos y que son mostrados en el Diagrama de Clases donde se entrelazan todas las clases *boundary*, *control* y *entity* que interactúan en todo el sistema, en el cual se ingresan los atributos y métodos de cada clase, por lo que en los diagramas de secuencia y colaboración (que se verá posteriormente) únicamente se hace referencia a estos elementos, y se suponen que ya fueron entrelazados; en otras palabras por ejemplo en el diagrama de Secuencia se parte de una Clase hacia otra por medio de un método, que ejecuta cierta acción en una secuencia en el tiempo, y así sucesivamente hasta completar el escenario que en este ejemplo será el Flujo Básico del Caso de Uso, al observar la figura 60 se entenderá de mejor manera lo antes dicho.

A continuación se presentan ciertas notas para entender mejor la secuencia de este Diagrama de Secuencias:

- ☞ Sobre la Clase *Boundary* FormalInformacionSubasta, despliega toda la información del Caso de Uso Ver Catálogo de Subasta en su flujo básico (no implementado en este ejemplo).

- ☞ En el paso 6, se despliega una forma al comprador para que ingrese su oferta. La información es pasada y estará disponible en todo el proceso.

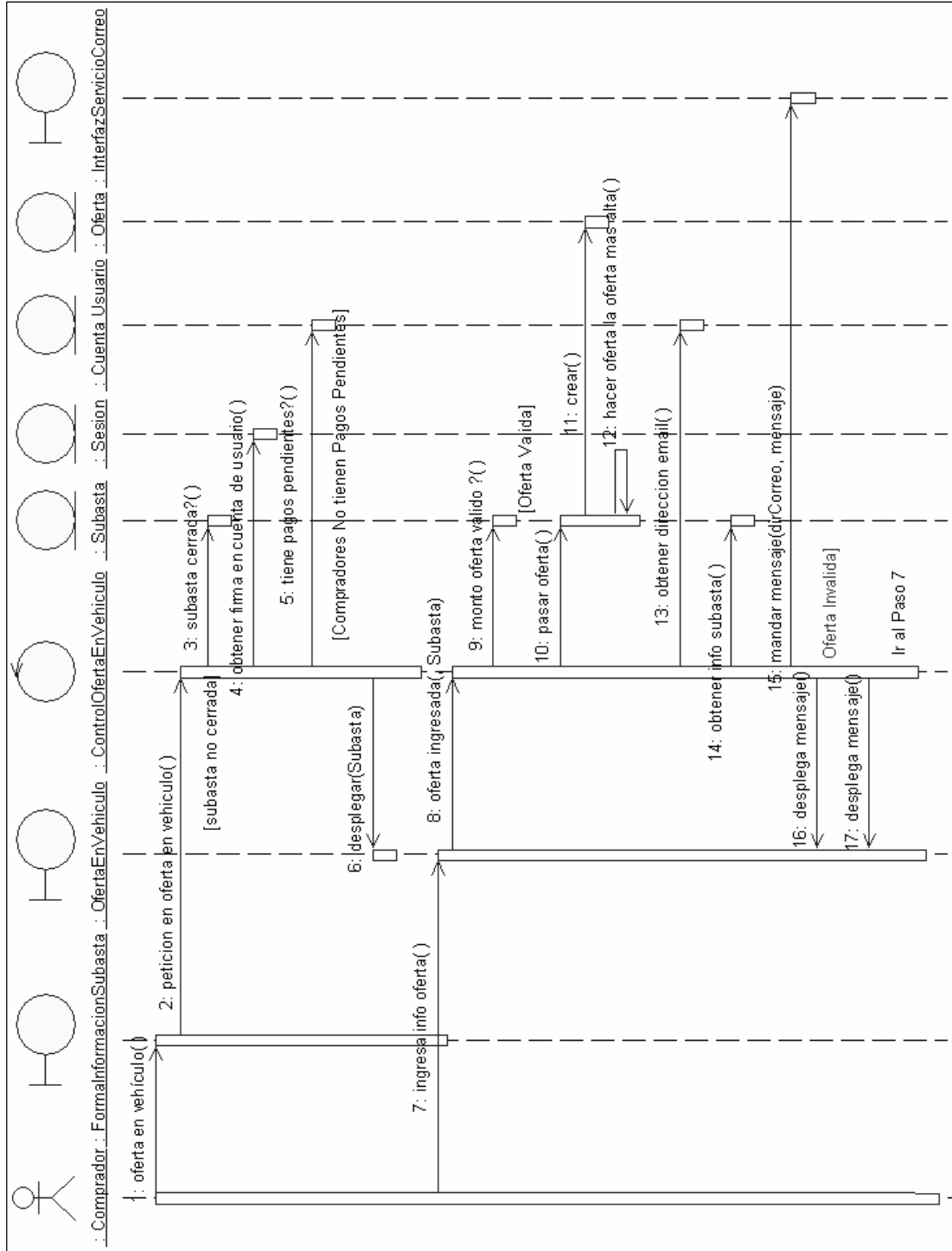
- 👉 En el paso 11, se crea una oferta por el usuario que esta dentro del sistema, o sea el comprador.

- 👉 En el paso 15, un correo electrónico es mandado al comprador, confirmando el monto de su oferta de un vehículo específico, así como la notificación al comprador cuando la subasta se anticipa para ser cerrada.

- 👉 En el paso 16, se despliega un mensaje al comprador, informándole que la oferta ha sido aceptada para el vehículo elegido. El mensaje desplegará el nombre del comprador, dirección de correo electrónico, el nombre del vehículo y el monto de la oferta.

- 👉 En el paso 17, se despliega un mensaje al comprador, informándole que la oferta incorporada es inválida, explicando porqué es inválida (ejemplo, la oferta incorporada no era mayor que la oferta más grande, por una cantidad mayor que el incremento de la oferta del mínimo especificado para la subasta). Se debe incitar al comprador volver a entrar la información de la oferta.

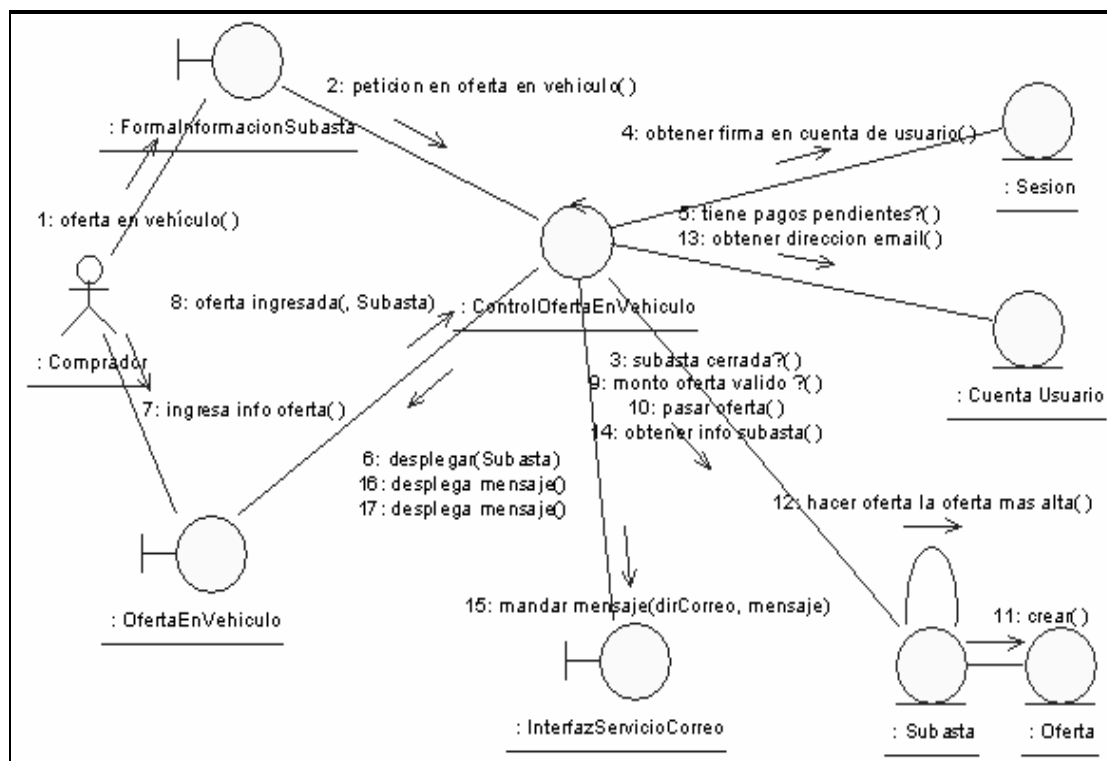
Figura 60. Diagrama de secuencia del análisis de la realización del caso de uso



4.10. Diagrama de colaboración del análisis de la realización del caso de uso

Es una forma alternativa de representar los mensajes intercambiados entre los objetos, aquí se enfatiza en los objetos en si y su interrelación, aunque siguiendo la secuencia de los mensajes también se puede entender los pasos del Caso de Uso. En la figura 61 se muestra el diagrama de Colaboración respectivo.

Figura 61. Diagrama de colaboración del análisis de la realización del caso de uso



4.11. Diseño de la realización del caso de uso

Ya conociendo el análisis del sistema a implementar, pasaremos a diseñarlo en el entorno J2EE, por lo que se trabajará el diseño aplicándolo con componentes *Java*, pudiendo comprender el enlace que existe entre los componentes de diseño y los mecanismos del J2EE utilizados. En este ejemplo se utilizó un patrón ya predefinido, el cual ya ofrece muchas funciones extras las cuales no serán utilizadas, pero ofrecen la facilidad de no crear todo de cero; este patrón se puede elegir a la hora de crear un nuevo modelo dentro de XDE. Para este ejemplo se utilizaron dos mecanismos:

1. Proceso de Presentación de Petición, y
2. Acceso a Sesión EJB.

Ambos mecanismos han sido contruidos combinando implementaciones de estrategias de algunos Patrones Sun J2EE, el primero ha sido derivado de la combinación de los patrones siguientes:

1. *Front Controller*
2. *Service to Worker* y
3. *Business Delegate*

(Estos patrones así como los mecanismos expuestos no se explicarán en este trabajo ya que no representan el objetivo del mismo)

Las siguientes decisiones de la arquitectura implementada en este ejemplo están contempladas en el mecanismo Proceso de Presentación de Petición:

- 👉 Todas las peticiones de usuario son manejadas por un solo Controlador de Presentación de Petición.
- 👉 El controlador principal examina cada petición para verificar consistencia y si el usuario necesita ser autenticado y/o autorizado.
- 👉 Si se autentica y autoriza al usuario, el Controlador de Presentación de Petición transmite a la petición un despachador apropiado.
- 👉 Un despachador maneja las interacciones del usuario para un Caso de Uso específico (hay un despachador para cada realización de Caso de Uso). Los despachadores "entienden" el flujo del Caso de Uso y coordinan interacciones entre el usuario y el Caso de Uso.
- 👉 Los despachadores no producen ninguna salida al usuario. En lugar de eso, llaman a vistas para crear código HTML o XML u otro código que regrese al dispositivo del usuario.
- 👉 Los despachadores no ejecutan la "lógica del negocio". En su lugar, utilizan a delegados del negocio como *proxies* a los servicios de servidor de aplicaciones.
- 👉 El controlador principal y los despachadores son implementados como *Servlets*. Por lo tanto, pueden ser colocados en diversos procesos y/o en diversas máquinas, que pueden ser deseables por razones de seguridad.
- 👉 Las vistas son implementadas como JSPs por facilidad de desarrollo.

- 👉 Los Delegados del Negocio son implementados como *Java Beans* que funcionan en el conducto de los delegados que llaman.

Las siguientes decisiones de la arquitectura implementada en este ejemplo están contempladas en el mecanismo Acceso a Sesión EJB:

- 👉 Todos los componentes del servicio de negocio son implementados como *Session EJBs*.
- 👉 Un cliente (en la mayoría de los casos es el despachador) que requiere el acceso a un componente del servicio de negocio, crea una instancia de un delegado de la sesión EJB.

Como consecuencia de utilizar los dos mecanismos, la aplicación tiene una separación distinta en el estereotipo *boundary*, entre la presentación lógica y la lógica del negocio. Esa *boundary* es la delegada del negocio ese es el Proxy a los servicios de negocio de la aplicación. Esta separación es muy deseable por algunas razones:

- 👉 Proporciona un contrato explícito entre los diseñadores y los desarrolladores del nivel de presentación y el nivel del negocio del sistema.
- 👉 Permite que el nivel del negocio cambie independientemente del nivel de la presentación.
- 👉 Permite el desarrollo concurrente de los dos niveles.

Es de suma importancia mencionar que estos mecanismos parten de un patrón ya existente por lo que todos los componentes y enlaces se omitirán, ya que al utilizar un patrón se nos proporcionarán todas las opciones ya entrelazadas solo para manejar las que a nosotros nos interesen, por lo tanto únicamente se explicarán el diagrama de clases, secuencia y colaboración, para ver la diferencia existente entre estos y los del análisis.

4.12. Diagrama de clases del diseño de la realización del caso de uso

En la figura 62 se muestra el diagrama de Clases de los objetos que participan en el Caso de Uso Oferta en Vehículo, mostrando la realización de este Caso de Uso, ya que se puede observar la utilización de los mecanismos mencionados en la sección anterior, únicamente que ya implementados con la realización del Caso de Uso, y con la funcionalidad que este requiere en un ambiente *Java*, siguiendo el estándar J2EE. Para facilitar esta aplicación se utilizaron los patrones mencionados en la sección anterior, que ya proveen diversos objetos para ser utilizados y aplicados en diversas maneras.

Como ya hemos dicho se utilizó la herramienta XDE para la implementación de estos patrones en nuestro modelo, con esto se logra tener una interrelación previamente existente entre todos los componentes, y únicamente se adaptan los nuestros.

Es importante mencionar que todas las realizaciones de todos los Casos de Uso tienen una estructura similar. Esto se da debido a que todos los Casos de Uso están derivados de los dos mecanismos descritos en la sección anterior, y ambos mecanismos surgen a partir de patrones existentes en el mercado.

A continuación se explicará mas a detalle la funcionalidad de cada una de las clases del diagrama mostrado en la figura 62:

👉 **ReguladorPresentacionPeticion**

Existe un ReguladorPresentacionPeticion para el uso de todo el sistema. Es responsable de procesar peticiones de la presentación (todas las peticiones se encaminan con el). Proporciona la autenticación del usuario (es el usuario "conocido" al sistema), la autorización del usuario (verifica si el usuario tiene los derechos de realizar la acción solicitada), y el encaminamiento de la petición de la presentación (transmite la petición al despachador apropiado).

👉 **DespachadorDePeticion**

Todas las peticiones del ReguladorPresentaciónPetición al DespachadorOfertaEnVehiculo y de este al las vistas se realizan por medio del DespachadorDePeticion. Dentro del contenedor Web, todos los accesos a un URL se deben hacer por medio del DespachadorDePeticion. Hay un DespachadorDePeticion para cada *HttpRequest*. Este recopila la información del estado sobre la comunicación "cadena" para una petición específica (es decir, sigue el "encadenamiento" de la petición e incluye mensajes de respuesta de una sola petición). Así, el DespachadorDePeticion detecta cuando es un mensaje "petición" envía después un mensaje "incluir" y el mensaje es enviado para la misma petición.

👉 **Delegados del Negocio**

Los delegados proporcionan una vista de presentación de la lógica del negocio que provee el contenido dinámico que la presentación necesita.

Además pueden ser considerados para ser las vistas de presentación de la lógica del negocio. También son responsables de proveer el contenido dinámico a la presentación. Proporcionan un interfaz a la lógica del negocio y aíslan la funcionalidad de la presentación de los detalles *backend* (encapsulan el modelo de la información del *backend*). Los delegados representan un contrato entre la presentación y las capas del negocio. Y además son las entidades del lado del servidor de presentación que se ejecutan en el contenedor *Web*. Son utilizados para blindar a clientes ya que estos están utilizando servicios alejados.

Los delegados son clases actuales y no EJB *proxies*, ya que estos se esconden detrás de los delegados. Los delegados son *proxies* "listos" para un EJBs remoto. Un delegado encapsula el código para localizar un servicio remoto, para unirlo a él y retornar una referencia. Los delegados son inteligentes ya que depositan en memoria la referencia remota del servicio una vez que se obtenga. Los delegados son utilizados primariamente por los despachadores, pero pueden ser utilizados por otros clientes.

👉 **Despachador oferta en vehículo**

El `DespachadorOfertaEnVehiculo` es la máquina de estado para la petición remitida por el `ReguladorPresentacionPeticion`. Controla la ejecución del proceso de la petición de la presentación. Dada una petición específica, el `DespachadorOfertaEnVehiculo` chequea el estado interno y determina qué vista se debe "invocar". Hay un sistema de delegados para cada despachador declarado como atributos de la clase del despachador.

4.13. Diagrama de secuencia del diseño de la realización del caso de uso

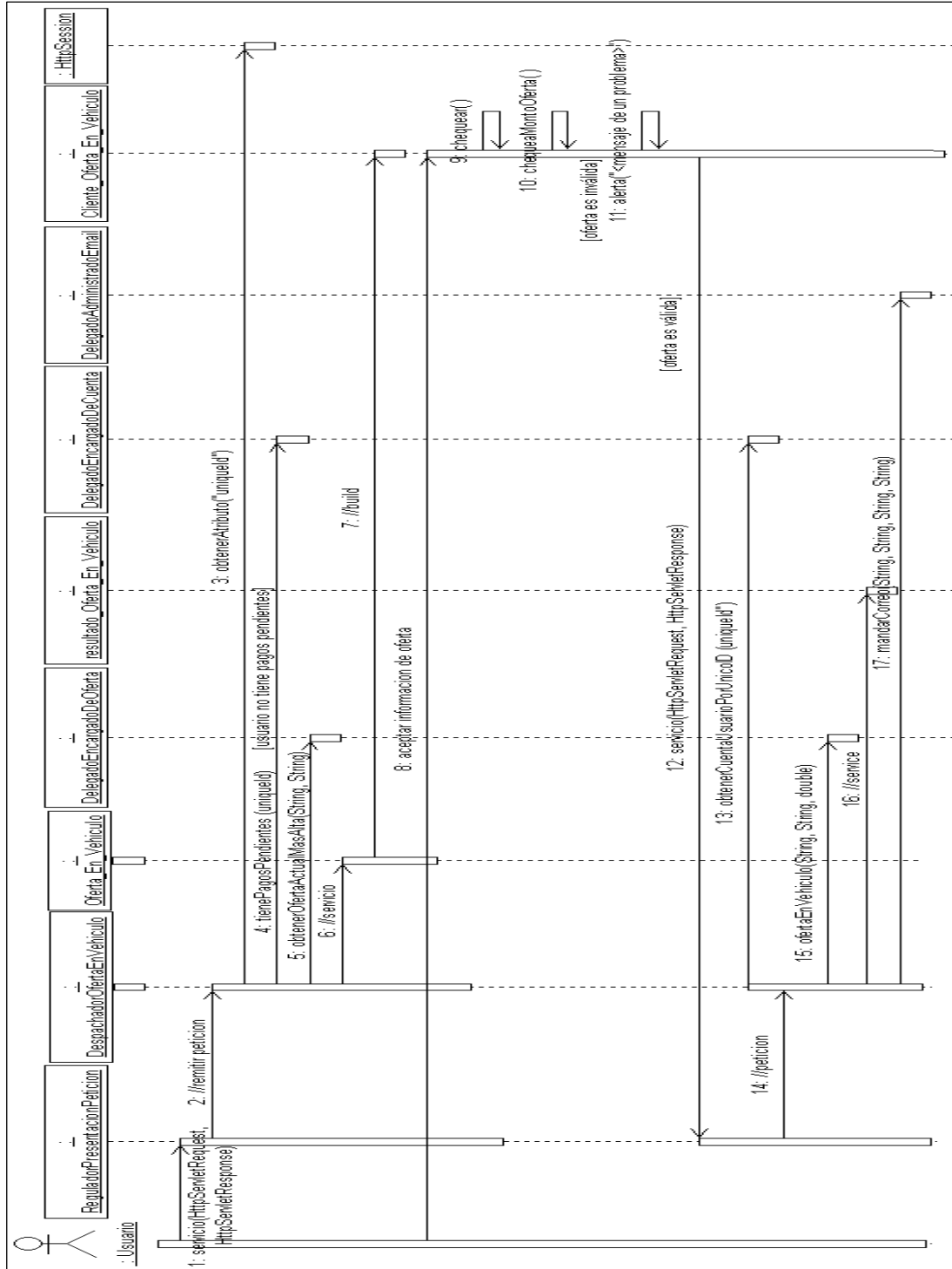
En la figura 63 se muestra un diagrama de Secuencia que representa la secuencia de pasos a seguir en el Caso de Uso Oferta en Vehículo, con la única diferencia que está expresado en la Realización del Caso de Uso en cuestión, esto quiere decir que tiene una arquitectura ideal para un desarrollador y ya no tanto para un usuario, además incorpora elementos que son propios del lenguaje de programación a utilizar, en este caso *Java* basado en el estándar J2EE.

Aunque esta dividido en dos figuras para una mejor visualización, pero es el mismo diagrama.

A continuación se muestra una secuencia seguida respecto al Diagrama de la figura 63:

- 👉 El usuario quisiera hacer una oferta en un vehiculo.
- 👉 Si el usuario no tiene ningún pago pendiente, el sistema exhibe una forma donde el usuario puede incorporar la información de la oferta (la cual será igual o mayor a la oferta ya establecida).
- 👉 El usuario ingresa la información incorporada de la oferta.
- 👉 Si la oferta ingresada es inválida, un mensaje se exhibe al usuario, solicitando que la ingrese información válida.
- 👉 Si la oferta ingresada es válida, la oferta se fija para el artículo y un *E-mail* se envía al usuario, notificándole que se ha fijado la oferta. Un mensaje también se exhibe al usuario que confirma el ingreso de la oferta.

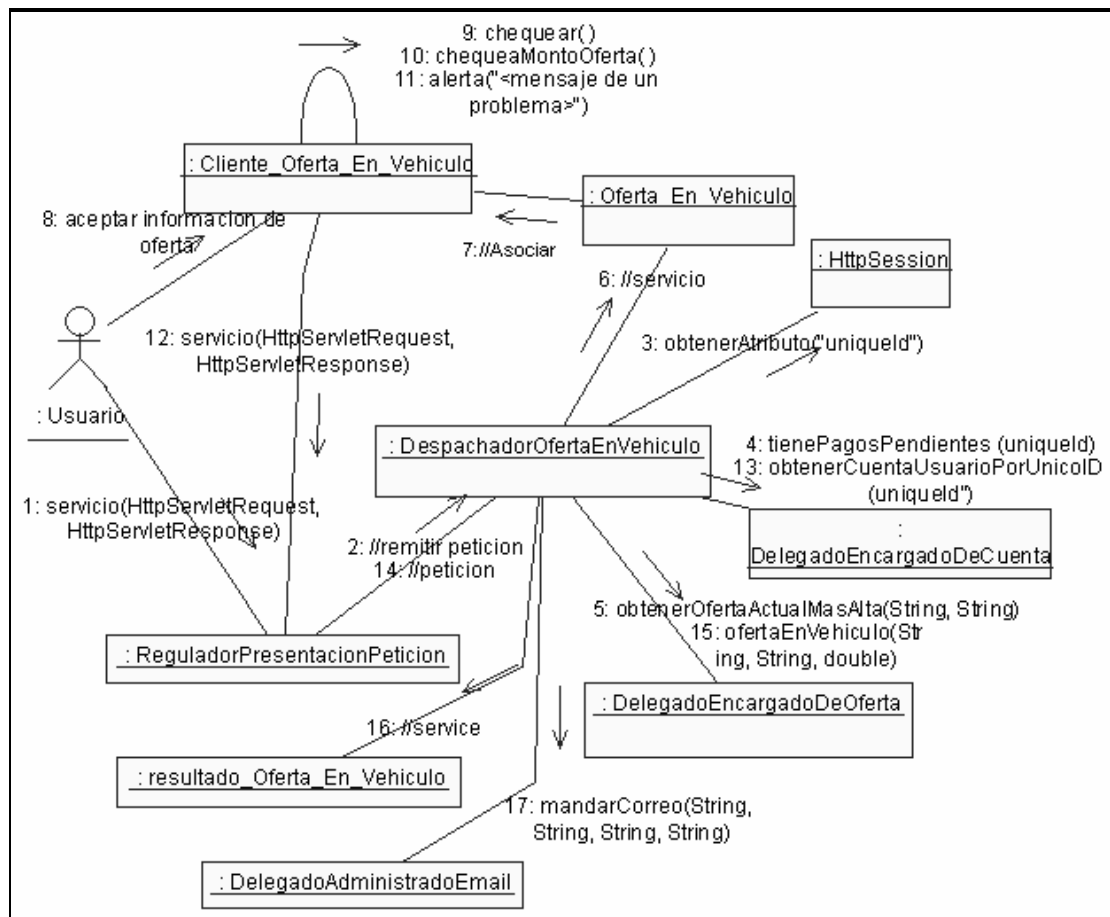
Figura 63. Diagrama de secuencia del diseño de la realización del caso de uso



4.14. Diagrama de colaboración del diseño de la realización del caso de uso

Este diagrama también es una forma alternativa de representar los mensajes intercambiados entre los objetos, enfatizando en los objetos en si y su interrelación, aunque siguiendo la secuencia de los mensajes también se puede entender los pasos de la realización del Caso de Uso. En la figura 64 se muestra el diagrama en cuestión.

Figura 64. Diagrama de colaboración del diseño de la realización del caso de uso



4.15. Arquitectura

Para la implementación de este sistema se utilizó la ayuda de la metodología *Rational* siguiendo un Desarrollo Rápido de Aplicaciones RAD, ya que se utilizaron guías que este provee, con las cuales se lograron definir de una manera clara los requerimientos deseados y la manera de resolverlos.

A partir de ello se procedió a implementar el modelo a seguir, que guiará a los desarrolladores durante todo el proceso. Este modelo se implementó completamente utilizando la Herramienta XDE, en la cual se elaboraron los diversos Casos de Uso y Diagramas necesarios. Todo esto se implementó utilizando guías que provee el Desarrollo Rápido en *Rational* (RRD), dentro de las cuales está el seguir patrones ya elaborados y que son proveídos por las herramientas utilizadas. Con estos patrones se obtiene la ventaja de no tener que elaborar todo de cero y únicamente utilizar las funciones deseadas y desechar las que no son aplicables a la resolución de nuestro problema.

El patrón utilizado aplica el Estándar J2EE el cual nos provee de todos los componentes *Java* necesarios para poder implementar el sistema y así poder interrelacionar dentro del mismo modelo los Casos de Uso establecidos con los componentes *Java* existentes y poder completar la Realización de los Casos de Uso en el Diseño del modelo del sistema.

El modelo creado va a servir para que los desarrolladores y cualquier persona que desee empezar a desarrollar el sistema, pueda entender de que se está hablando y por donde empezar, ya que se observan todas las interrelaciones que debe tener el sistema para funcionar como se especifican en los requerimientos.

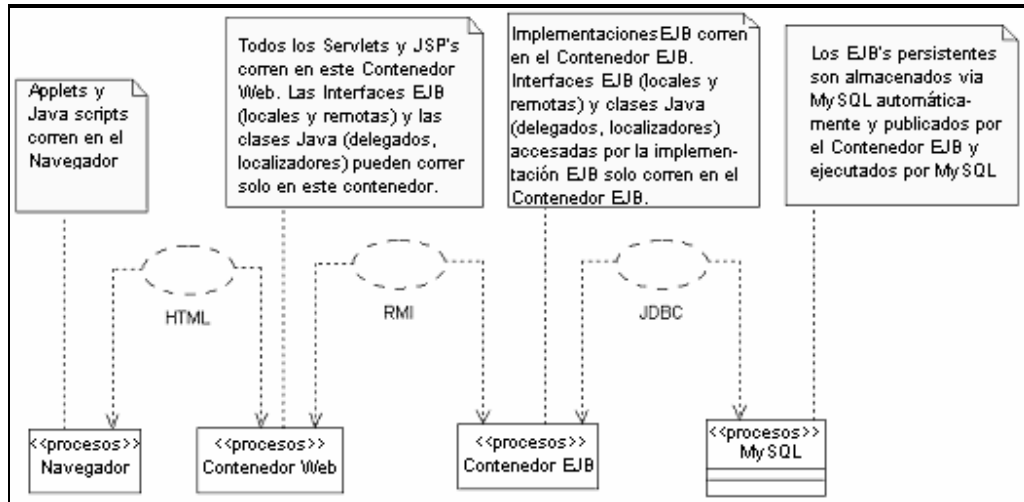
En base a los requerimientos planteados anteriormente se deberá implementar este sistema utilizando un servidor que funcionará como un contenedor *Web* (en este caso *Tomcat* por ser una implementación oficial de referencia para los *Java Servlet* y *Java Server Pager*) en el cual se tendrán los *Applets* y *Java Scripts*, *JSP's* a los cuales accederán los usuarios y serán vistos en un Navegador.

Además se debe de tener un servidor *JBoss* como contenedor de los *EJB*. Así como un sistema de Base de Datos como *MySQL*, para guardar la información necesaria como *Usuarios*, *Subastas* etc. *MySQL* facilita todo ya que provee una interfase *JDBC* que hace fácil la integración con el servidor *JBoss* para utilizarlo en un manejo del contenedor de la persistencia de las Clases *Entity* o entidades, ya que todo dato de la subasta debe estar almacenado de manera persistente y no perderse.

La decisión para utilizar dos contenedores separados *J2EE* corriendo al mismo tiempo en vez de un solo contenedor integrado, es para proveer al equipo de desarrollo la flexibilidad en cambiar o aumentar el contenido tanto del contenedor *Web* como del contenedor de *EJB*, y así minimizar el efecto que pueda tener uno de otro. Otra ventaja es que el equipo de desarrollo puede cambiar el contenedor por el último y más novedoso que exista. Aquí a lo que le llamamos contenedor es a un servidor con las características necesarias para soportar el contenido a guardar.

En la figura 65 se presenta una donde se ejemplifica la arquitectura utilizada y el manejo a gran escala que tendrá este ejemplo:

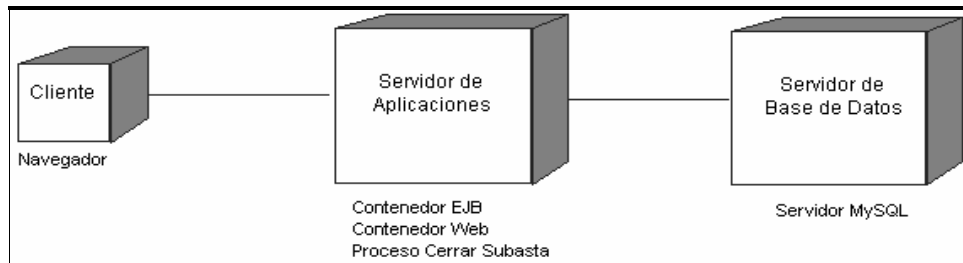
Figura 65. Arquitectura implementada en el sistema de subastas en línea de vehículos



4.16. Vista de despliegue

La vista de despliegue del sistema demuestra los nodos físicos en los cuales el sistema se ejecuta y la asignación del sistema procesando los nodos. La aplicación Sistema de Subastas en Línea de Vehículos se puede desplegar en diversas configuraciones de hardware. Por ejemplo, puede ser desplegado en una sola estación de trabajo NT. La figura 66 muestra la configuración más típica del despliegue que debe utilizar el equipo del desarrollo.

Figura 66. Vista de despliegue del sistema de subastas en línea de vehículos



CONCLUSIONES

1. La elaboración de distintos diagramas y artefactos siguiendo la metodología RUP proveen una fácil ejecución del proceso de elaboración de un Sistema de *Software*, ya que describen cómo está estructurado el sistema desde diferentes perspectivas orientadas a los diferentes involucrados en un proyecto.
2. Se puede reducir el tiempo de desarrollo de un Sistema de *Software*, aplicando la tecnología RRD basado en la metodología RAD ya que permite lograr de una manera fiable y rápida el desarrollo del Sistema deseado.
3. Colocando componentes en los distintos servidores que conformen el sistema a desarrollar, definidos en las especificaciones del estándar J2EE, se cuenta de una manera automática con todos los servicios prestados por dichos componentes, es decir, se ponen a disposición de los desarrolladores un gran número de herramientas que se pueden aprovechar en la realización del Sistema de *Software* de una manera mucho más eficaz.
4. El tener todo el procedimiento de desarrollo de un Sistema de *Software*, acoplado y entrelazado en un modelo particular elaborado en XDE, es una herramienta necesaria y efectiva para administrarlo; y así contar con una visión unificada de todo el proceso, con lo que se facilita la implementación del mismo.

5. La aplicación de cambios a un Sistema de Software, se puede implementar con mayor facilidad apoyándose en un Análisis de Impacto, para ver los efectos que tendrán dichos cambios, teniendo desde un principio todo el proceso de Análisis y Diseño implementado y unificado, en el cual se podrá verificar los enlaces que existen en el sistema y así detectar fácilmente los cambios que son necesarios realizar.

6. La interrelación de las metodologías RUP y RAD con el estándar J2EE y con herramientas XDE y RRD nos proveen una elaboración de aplicaciones de manera rápida, eficiente, ordenada y con el menor número de defectos.

RECOMENDACIONES

1. Para contar con un enfoque disciplinado en la asignación de tareas y responsabilidades dentro de una organización del desarrollo, es necesaria la aplicación de una metodología, con la cual se puede mantener una fácil administración de este proceso; como por ejemplo la metodología RUP.
2. Para obtener un máximo control de variables que conlleva un desarrollo de aplicaciones y poder mantener una ordenada implementación de éstas, es importante seguir metodologías y estándares que nos lleven a estar en competitividad en todo momento.
3. Al implementar un Desarrollo Rápido de Aplicaciones de un Sistema particular, es importante la utilización de Patrones, los cuales ya tienen una funcionalidad general y han sido predefinidos, y así contar con una base consistente y previamente elaborada para la implementación del Software.
4. Para finalizar de una manera eficiente, rápida y funcional la implementación de un Sistema de Software, se debe partir utilizando una metodología que guíe todo el progreso como lo es el RRD ya que por estar basada en el RUP, se puede contar con las ventajas que éste provee y adicionalmente la ventaja del RRD: “Un Desarrollo Rápido”.

5. Se debe contar con un Modelo de todo el Sistema de Software (por ejemplo implementado en XDE) y conjuntamente con los artefactos creados que provee alguna metodología para desarrollo (por ejemplo RUP), para tener las bases necesarias en la administración del proceso de elaboración, con una visión unificada del mismo, facilitando su implementación.

BIBLIOGRAFÍA

- 1 **Aprendiendo UML en 24 horas.**
Schmuller, Joseph. Editorial *Prentice Hall*, México, 2000.
- 2 **Rational Rapid Developer, Technical Overview.**
IBM, *Rational Software*, 2003
- 3 **Arquitectura empresarial y *software* libre, J2EE**
<http://www.javahispano.org/articles.article.action?id=70>
- 4 **Catálogo de patrones de diseño J2EE. I.- Capa de presentación**
<http://www.programacion.com/tutorial/patrones/>
- 5 **Curso 00 usando UML versión abril 2003**
<http://www.dsic.upv.es/~uml/>
- 6 **Documentación de ayuda al instalar el RUP**
/Rational/RationalUnifiedProcess/process/
- 7 **Estereotipos para clases**
<http://www.gris.det.uvigo.es/~avilas/UML/node38.html>
- 8 **IBM *WebSphere* - *e-commerce* dinámico**
http://es.morse.com/templates/generic-widepic.xml?content_id=3309&the_section_id=5035
- 9 **Integración de aplicaciones**
http://es.morse.com/templates/generic-widepic.xml?content_id=3380&the_section_id=5036
- 10 **Introducción al RUP con J2EE**
<http://www.awprofessional.com/articles/article.asp?p=30317>

- 11 **Introducción al UML**
<http://www.yoprogramo.com/articulo4.php>

- 12 **Metodología de desarrollo de sistemas basados en el ciclo de vida**
<http://comip.mendoza.gov.ar/metodologia%20para%20el%20desarrollo%20de%20sistemas.pdf>

- 13 **Modelos para el desarrollo de *software***
http://docentes.usaca.edu.co/wildiaz/INGSOF_MODELOS.html

- 14 **¿Para quiénes está desarrollado XDE?**
<http://www.rational.com.ar/herramientas/desarrollodesoftware.html#XDE>

- 15 **Plataforma de *software* WebSphere**
<http://www.w3c.org/TR/1999/REC-html401-19991224/loose.dtd>

- 16 **Portales**
http://es.morse.com/templates/generic-widepic.xml?content_id=338_2&the_section_id=5037

- 17 **Productos y servicios de *software* WebSphere**
<http://www.ibm.com/pe/products/software/websphere/>

- 18 **¿Qué es XDE?**
<http://www.rational.com.ar/herramientas/xdeprofessional-desarrolloconlibertad.html>

- 19 **Tecnología de la información aplicada a la administración tributaria**
http://www.ciat.org/doc/buen/metodologias_y_estandares_en_el_desarrollo_de_sistemas.pdf

- 20 **Trabajando la aplicación J2EE con el RUP**
<http://www.awprofessional.com/title/0201791668>