



SEP
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TOLUCA

Reporte Final del Proyecto de Semestre Sabático

Manual de Prácticas de la Materia de Fundamentos de Ingeniería de Software

Período: 11 de agosto del 2014 al 11 de febrero del 2015

Profesor: Ing. Eugenio Falcón Isunza

Metepec, Edo. de México a 11 de febrero del 2015

Índice

	Página
Práctica 1. Investigación y selección de una herramienta CASE	1
Práctica 2. Aplicación de los procesos básicos de la Ingeniería de Requisitos	14
Práctica 3. Generación de un modelo de requisitos con la ayuda de una Herramienta CASE	26
Práctica 4. Elaboración de un modelo de análisis para un sistema.	54
Práctica 5. Generación de un modelo de análisis con la ayuda de una Herramienta CASE	59
Práctica 6. Elaboración de un modelo de diseño basado en responsabilidades para un sistema.	66
Práctica 7. Generación de un modelo de diseño con la ayuda de una Herramienta CASE	81
Práctica 8. Elaboración de un modelo de implementación para un sistema de aplicación real	108

Práctica No.1

1.1 Nombre: Investigación y selección de una herramienta CASE.

1.2 Objetivo: Que el alumno investigue las herramientas CASE más utilizadas en el mercado y en base a una selección previa de los más utilizados a la fecha, desarrolle una metodología de comparación entre las mismas de manera que fundamente la selección de una de ellas para ser utilizada dentro del desarrollo de un sistema utilizando la metodología RUP.

1.3 Introducción. En el desarrollo de sistemas de información existe el interés por mejorar la productividad y calidad al momento de crear productos de software. Para ello, es indispensable establecer un entorno disciplinado y estructurado que genere una ventaja competitiva frente a las demás organizaciones y en el cual predomina el trabajo en equipo, esto se logra con la utilización de herramientas de auxilio en el desarrollo de sistemas de cómputo tipo CASE (sistemas de cómputo para la asistencia en el desarrollo del software).

El lenguaje de especificación o modelado utilizado en las herramientas CASE, representan el vehículo indispensable para la comunicación entre los participantes y para la representación de conceptos a lo largo de todo el proyecto, utilizando una semántica común

El UML2.1 es el lenguaje de mayor difusión que la OMG (Object Management Group) ha convertido desde 1997 en el estándar para definir, organizar y visualizar los elementos que configuran la arquitectura de una aplicación, sea o no de software.

Además de la aproximación metodológica y el lenguaje, uno de los factores determinantes en el éxito de un proyecto de software, lo constituye la acertada selección de las herramientas de trabajo que se utilizarán durante su desarrollo, es por eso que en la actualidad la administración de proyectos pone especial atención en la selección de la herramienta CASE como una tarea de vital importancia.

Las herramientas CASE están tomando cada vez más relevancia en la planeación y ejecución de proyectos que involucren sistemas de información, pues suelen inducir a sus usuarios a la correcta utilización de metodologías que le ayudan a desarrollar con facilidad productos de software. Todas las herramientas CASE prestan soporte a un lenguaje de modelado para acompañar la metodología y es lógico suponer, que un alto porcentaje de ellas soportan UML, teniendo en cuenta la amplia aceptación de este lenguaje y el valor conceptual y visual que proporciona, así como su facilidad para extender el lenguaje en la representación de elementos particulares para determinados tipos de aplicaciones.

Para cumplir con el objetivo propuesto para esta práctica, revisaremos a continuación la forma en que es posible su evaluación, para poder hacer la selección adecuada a nuestras necesidades.

Evaluación de herramientas de modelado con UML. La toma de una acertada decisión a

la hora de escoger una herramienta de modelado con UML, puede ser un factor importante para lograr la calidad de un proyecto. Por tal razón, se necesita tener un amplio conocimiento en aspectos como:

- Las herramientas que existen en el mercado (Con licencia o libres).
- Las características de una buena herramienta de modelado con UML.
- La manera como las herramientas satisfacen las necesidades de las personas que participan en un proyecto, para apoyar el proceso de Ingeniería de Software.

Para facilitar el entendimiento de los criterios a tener en cuenta al momento de evaluar una herramienta de modelado con UML, se han analizado cuatro aspectos en los que estas herramientas deben satisfacer resultados, que son las siguientes:

1. Enfoque Procedimental. El enfoque procedimental se refiere a la forma como las herramientas hacen uso de las metodologías para guiar al usuario a través de un proceso de Ingeniería de Software, este enfoque se refiere a los siguientes tópicos:

a) Apoyo Metodológico. Una herramienta de modelado con UML debe apoyar el uso de una o varias de las metodologías orientadas a objetos para el desarrollo de sistemas de información; es importante que dicha herramienta garantice la unicidad y coherencia ya sea entre los diferentes modelos del sistema que representan vistas complementarias del sistema (estructurales, dinámicos, funcionales, etc.) o entre modelos que representan diferentes niveles de abstracción (de negocio, de análisis, de diseño, etc.). Esto permite guiar al usuario a través de etapas o fases permitiendo una trazabilidad entre diagramas de alto nivel que representan aspectos del dominio del problema y diagramas detallados que representan detalles de implementación.

Para conseguir tal propósito, es importante que el navegador de la herramienta, muestre las etapas de la metodología o los modelos que propone la misma para cada etapa, fase o arquitectura, y de ser necesario, que le ayude al usuario a evolucionar o refinar un modelo al momento de pasar de una etapa a otra o de un aspecto o contexto arquitectónico a otro.

b) Soporte completo UML. Esta característica se refiere a la capacidad de las herramientas de construir todos los diagramas que propone UML, o por lo menos los más relevantes: el diagrama de casos de uso, que representa la funcionalidad o alcance del sistema; el diagrama de clases, que describe la estructura de objetos y sus relaciones; el diagrama de interacción (secuencia o colaboración) y, en algunos casos, el diagrama de estados, que visualiza el ciclo de vida de un objeto.

También es importante la versión que se soporta de UML y la manera como la herramienta cubre los conceptos tratados en ella. Aunque la OMG liberó recientemente la versión 2.1 de UML, todavía existen en el mercado muchas herramientas que trabajan con la versión 1.3.

c) Facilidad de extensión del lenguaje. A pesar de que UML proporciona riqueza sintáctica y semántica para la especificación de modelos, se presentan condiciones particulares de los ámbitos de desarrollo que no logran ser satisfechas. Para tal efecto, UML ofrece mecanismos de extensibilidad tales como restricciones, estereotipos y valores

etiquetados (OMG, 2003), que deben ser contemplados e implementados por las herramientas CASE. Además, es deseable que dichas herramientas contemplen la posibilidad de edición de símbolos gráficos que permitan representar aspectos particulares de un dominio no contemplados por UML.

d) Modelado de datos. El modelado de la información persistente constituye uno de los factores importantes cuando se está construyendo un sistema de información. En términos prácticos dicha persistencia, estará soportada en bases de datos relacionales (Arce, 2003). Es por esto, que las herramientas de modelado con UML le deben permitir al usuario facilidades para definir el esquema relacional, bien sea a través de una interfaz para la construcción de diagramas entidad - relación o modelos de datos, o a través de la extensión de UML, modelando clases persistentes que se puedan implementar en motores relacionales.

Los modelos que se construyan utilizando la herramienta deben proveer la facilidad de generar scripts SQL, con las instrucciones DDL para la creación de los objetos de la base de datos.

e) Autogeneración. Es una característica que podría tener dos sentidos, generar código a partir de modelos y generar modelos a partir de código. Varias de las herramientas disponen de la generación de la definición estática (propiedades y signature de los métodos) de las clases en diferentes lenguajes (como Java, C++) y para diferentes arquitecturas (como CORBA y J2EE).

También es recomendable que la herramienta genere el código que implementa la funcionalidad de mapeo (materialización y desmaterialización) entre el modelo de objetos y el modelo relacional, generalmente conocido como esquema de persistencia.

La generación de código puede presentar dos problemas:

- Cuando el código ya se ha generado y se desean realizar cambios sobre éste, la herramienta suele sobrescribir el código que el desarrollador ha introducido directamente.
- Al momento de generar el código, la herramienta CASE pone un gran número de líneas de comentario llamadas polución. Para solucionar esto en la actualidad, los editores de los lenguajes de programación pueden colapsar y expandir las líneas de comentarios, mostrándole al usuario sólo lo que él quiere ver.

f) Ingeniería inversa. Mientras que un proceso clásico de Ingeniería de Software, conduce al usuario desarrollador a través de unas etapas para que, a partir de unos modelos, llegue a un producto de software construido, la Ingeniería Inversa parte del producto y lo transforma en modelos.

La generación de diagramas a partir de código puede tener dos vertientes:

- La generación a partir de lenguajes OO, o a partir de lenguajes estructurados.

La transformación a partir de programas OO se realiza con el propósito de documentar, o bien, reconstruir modelos con base en la versión en producción de un sistema. En este caso es ideal, que además de la generación del diagrama de clases, se puedan también generar diagramas de interacción (colaboración y secuencias) y diagramas de estados (OBD, 2003). Esto puede ser posible a partir de archivos de rastreo durante la ejecución de la aplicación, extrayendo los mensajes enviados entre clases para los diagramas de interacción y los cambios en los estados de una clase para los diagramas de estado. Los modelos así obtenidos son fuente valiosa para la revisión, transformación y generación de la versión en una nueva iteración del proceso del software (Mejía, 2003).

Para aplicaciones construidas en lenguajes estructurados (C, VB, COBOL, PASCAL y SQL), se hace interesante la posibilidad de realizar ingeniería inversa, facilitando ya sea la migración de sistemas con fines de actualización o la identificación de aspectos estructurales y relacionales de los sistemas difíciles de leer desde el código.

g) Métricas. Medir la bondad de un conjunto de modelos que representan un sistema suele ser una tarea de mucha utilidad para construir sistemas viables, de fácil mantenimiento y de buen rendimiento.

Las métricas son generalmente presentadas en reportes con información como: el número de atributos y métodos por clases, el índice de métodos públicos, privados y protegidos, el número de líneas de código por método, el grado de acoplamiento y el grado de cohesión .

h) Apoyo a lenguajes formales. Un lenguaje formal es una técnica de base matemática para describir las propiedades de un sistema, con el propósito de evitar la ambigüedad y minimizar la interpretación subjetiva de la especificación de un sistema.

OCL es el lenguaje formal adoptado alrededor de UML; es utilizado para especificar las invariantes, precondiciones y postcondiciones y otras restricciones de modelos orientados a objetos con el fin de otorgar mayor formalidad a las especificaciones y aspectos visuales de UML.

Integrar un lenguaje formal en una herramienta CASE, es una de las características sobresalientes que distinguen una herramienta CASE robusta. El lenguaje formal dota los diagramas con una semántica precisa y consistente, lo cual permite que la generación de código se pueda hacer de manera completa.

i) Actualización. El mejoramiento continuo es una práctica de la mayoría de herramientas de modelado con UML.

2. Soporte al modelado Arquitectónico. Con la revisión de este aspecto se pretende evaluar la capacidad que tienen las herramientas de modelado UML para apoyar la definición de la arquitectura de un sistema. La arquitectura es el elemento clave de diseño que permite establecer acuerdos de alto nivel referentes a la forma del sistema (Módulos y subsistemas) y sus relaciones.

El énfasis en componentes y arquitecturas es una de las aproximaciones para desarrollo de software más prometedoras (Anaya, 2000), es por lo tanto importante analizar el soporte que una herramienta CASE provee para la definición de arquitecturas.

a) Soporte a ADL. Un Lenguaje de Definición de Arquitecturas (ADL) proporciona notaciones para descomponer un sistema en componentes y conectores y especificar cómo se combinan estos elementos para formar cierta configuración, teniendo la ventaja de ofrecer una notación descriptiva directa para las principales abstracciones arquitectónicas. De este modo, los arquitectos pueden definir la estructura de sus sistemas evaluando posibles cambios, y guiar su implementación evitando la ambigüedad.

Entre algunos de los ADL's más conocidos están: WRIGHT, Aesop, Darwin, Rapide y UniCon (Georgas, 2003). Una de las características sobresalientes de la versión de UML 2.0 es su extensión como lenguaje ADL.

Es importante que las herramientas CASE den un soporte adecuado a la definición de arquitecturas, de manera básica, facilitando la descripción y consistencia de vistas de acuerdo al modelo de 4+1 vistas (Kruchten, 1995) de UML o, de forma avanzada, proveyendo soporte para conceptos propios de un ADL.

b) Apoyo al modelado por capas. El estilo o patrón de arquitectura más utilizado es el que especifica un sistema estableciendo una clara separación de la funcionalidad de éste a través de capas o niveles. El patrón de arquitectura por capas establece como mínimo la definición de tres capas: la capa de presentación, que es la parte del sistema que ofrece interface al entorno; la capa de lógica del negocio que determina el comportamiento de la aplicación; y la capa de datos, que administra la información persistente que maneja el sistema.

El nivel de aceptación de esta arquitectura se debe a los beneficios que proporciona como: escalabilidad del sistema a través de la fácil distribución de componentes del sistema, confiabilidad por medio de implementación de múltiples niveles de redundancia, flexibilidad ante cambios debido al bajo acoplamiento entre la lógica de presentación y la lógica del negocio, y soporte a la reusabilidad por la identificación de elementos con una clara funcionalidad.

Es importante que la herramienta CASE permita la visualización de la arquitectura de alto nivel del sistema por medio de paquetes y sus relaciones de dependencia y que además, provea soporte especializado en cada una de las capas de la arquitectura.

c) Servicios de presentación. Tal como se mencionó, la capa de presentación contempla los componentes que especifican la interacción del sistema con el entorno (usuario, sistemas externos, dispositivos, etc.), los elementos propios de esta capa deben poderse representar haciendo uso del estereotipo, con su respectivo ícono asociado, establecido por UML para especificar objetos de tipo vista (<borde>).

d) Servicios de lógica del negocio. Las reglas del negocio pueden normalizarse de la misma manera como se hace con los datos, para disminuir costos y mantenimientos,

incrementar productividad y reducir esfuerzos. Por esto, el proceso de administración de las reglas del negocio, por parte de las herramientas CASE, debería caracterizarse por facilitar la unicidad, consistencia e independencia en la implementación de las reglas, con el fin de apoyar el sostenimiento de la integridad del sistema (CLARION,2003). Es en este punto en el que lenguajes como el OCL contribuyen enormemente a describir reglas, restricciones y particularidades de los algoritmos que reflejen la lógica del negocio.

e) Servicios de persistencia. La capa de persistencia administra la información que debe perdurar en el sistema, y para este conjunto de componentes, los servicios prestados por las herramientas de modelado le deben permitir al usuario construir diagramas de bases de datos o extender el UML, de tal manera que pueda representar las clases persistentes, haciendo uso de estereotipos.

3. Apoyo al repositorio. Los repositorios son herramientas que permiten centralizar, administrar y gestionar las versiones o estados de un proyecto en el que se requieren revisiones frecuentes. En toda herramienta CASE, se hace relevante la presencia de un repositorio, no sólo si la herramienta es multiusuario, sino en general para la organización de los cambios reflejados en el refinamiento de los modelos, en el avance de las etapas o en la colaboración entre usuarios.

Para resaltar la relevancia del apoyo al repositorio por parte de una herramienta CASE, abordamos cuatro aspectos: la robustez de la herramienta a partir del uso de un repositorio, las herramientas de administración, la colaboración entre usuarios y el intercambio.

a) Robustez. La robustez se refiere a la capacidad de los sistemas de reaccionar apropiadamente ante condiciones excepcionales (Meyer1999), como casos no previstos en la especificación del sistema (por ejemplo, fallas eléctricas, entrada errónea de datos, incluso la corrupción por virus); y la capacidad de recuperarse ante tales situaciones, protegiendo de forma segura la información almacenada en un repositorio.

b) Herramientas de administración. La administración en una herramienta CASE, podría clasificarse en cuatro categorías:

- De repositorio. El administrador de la plataforma CASE, necesita realizar tareas como, creación, actualización y borrado del repositorio, mantenimiento de usuarios, copias de seguridad, recolección de estadísticas y manejo de las extensiones que usan los usuarios en los archivos, al momento de la autogeneración.
- De ejecución de proyectos. El administrador del proyecto, necesita realizar tareas como: identificación de riesgos, planeación del proyecto, asignación de tareas a personas, asignación de fechas y recursos a tareas, seguimiento de las tareas o fases de un proyecto.
- De áreas de trabajo (ORACLE, 2002). Las áreas o espacios de trabajo (*workarea* o *workspace*) son la manera en que la herramienta facilita la administración de todos los objetos sobre los que puede trabajar un usuario, tanto los objetos de su propiedad, como los que otros usuarios le compartieron. La administración de las áreas de trabajo las realiza cada usuario a través de un navegador de objetos del

repositorio. Con frecuencia existe un área de trabajo global compartida, en la que se ubican los objetos a los que todos los usuarios tienen acceso.

- De aplicaciones. En esta categoría se ubican tareas como la administración de versiones, el manejo de dependencias y la comparación entre modelos.

c) Colaboración entre usuarios. En la ejecución de un proyecto de Ingeniería de Software participan muchas personas (analistas, arquitectos, diseñadores, revisores, administradores, etc.). Cada uno puede necesitar al mismo tiempo, información del mismo sistema, por tanto es imprescindible compartir información. Esto significa que las herramientas CASE que soportan el acceso multiusuario deben poseer estrategias de control de concurrencia, ya sea soportados por una base de datos, o esquemas de concurrencia más sofisticados como los propuestos para el desarrollo de sistemas distribuidos.

Para facilitar dicha colaboración entre usuarios, las herramientas proporcionan funcionalidades para: definir roles, otorgar y revocar privilegios para cada rol establecido, de tal manera que se puedan compartir objetos del repositorio.

d). Intercambio. Es la posibilidad de que la información de los proyectos guardada por una herramienta en cierto formato, pueda ser leída por otras herramientas CASE, que acepten dicho formato. Si la herramienta CASE soporta UML, la funcionalidad de intercambio de modelos con otras herramientas CASE se convierte en una meta alcanzable, teniendo en cuenta la estandarización sintáctica y semántica que persigue dicho lenguaje.

En esta línea de estandarización alrededor de UML, la OMG definió un estándar de intercambio llamado XML que basado en el metamodelo de UML, sirve para el intercambio de metadatos utilizando XML.

4. Enfoque funcional. El último enfoque, estudia las utilidades adicionales que le ayudan al usuario a desarrollar un sistema de información con mayor o menor facilidad, los aspectos a considerar son el versionamiento, la facilidad de navegación, el manejo de diagramas y la edición de código.

a) **Versionamiento.** Las iteraciones en el proceso de Ingeniería de Software, proponen que se construyan versiones de los artefactos y que, de forma incremental, se llegue a un producto maduro. Por tal motivo, es importantísimo que la herramienta de modelado, le permita al usuario guardar versiones del sistema o de sus artefactos, de tal manera que cuando comience otra iteración, la versión previa esté disponible.

b) **Navegación.** Con frecuencia los usuarios de las herramientas de modelado con UML, necesitan acceder directamente a la información de una clase o método, o también revisar los diagramas que han construido. Para facilitar éstas y otras labores, las herramientas deben estar dotadas de un navegador de objetos del repositorio, que presente toda la información de los modelos en una forma organizada, garantizando el rápido acceso a cualquier información.

c) **Manejo de diagramas.** Para la manipulación de diagramas existen tres labores fundamentales que se deben tener en cuenta:

- Visualización (ORACLE, 2002). Al momento de visualizar un diagrama, hay varias funciones que resultan de mucha utilidad para el usuario, como:
 - *Zoom in y Zoom out.* Para acercar o alejar el diagrama.
 - *Refrescar:* Para que las clases o componentes del diagrama se actualicen con base en el repositorio.
 - *Ajustar:* Para que el área de visualización, sea ocupada totalmente por el diagrama o parte de éste.
 - *Autolayout.* : Para arreglar la presentación de los objetos dentro del diagrama, de tal manera que estos queden lo mejor dispuestos posible.
 - *Show y Hide.* Para mostrar u ocultar información y adornos de los diagramas, y de esta forma no saturarlos, dependiendo del nivel de detalle deseado.
- Impresión. La herramienta debe tener características que le permiten al usuario ver el documento al momento del procesamiento como saldrá impreso, es decir la Vista Previa. También suelen necesitarse funcionalidades como la impresión en múltiples páginas o a escala, para ajustar el diagrama al número de páginas deseadas.
- Exportación. Poder manipular los diagramas construidos con la herramienta de modelado, se convierte en una posibilidad que resulta bastante útil. Por esto, la exportación de los diagramas a los formatos gráficos más populares como GIF, JPEG o BMP, se presenta como una característica deseable para las herramientas de modelado.

d) Edición de código. Para que la integración entre los modelos y su representación en un lenguaje de programación sea mayor, la herramienta de modelado debe disponer de editores de código integrados, que le faciliten al usuario la edición del código fuente de las clases, los métodos y todos los artefactos que involucre la solución ya implementada. Es conveniente que las herramientas de modelado tengan integración directa con entornos avanzados de desarrollo, conocidos como IDE.

1.4 Especificar la correlación con el o los temas y subtemas del programa. El contenido de esta práctica se relaciona con el programa de la materia ya que el conocimiento de herramientas CASE y la caracterización correspondiente, así como la selección de una de ellas para ser utilizadas a lo largo del curso son los temas que debe cubrir en la primera unidad del programa de la materia.

1.5 Material y equipo necesario. Se recomienda que el alumno contraste al menos 4 herramientas CASE con licencias o libres que se utilicen en la programación orientada a

objetos, y puedan modelarse con UML en un ambiente de Windows ver. 7.0 o 8.0, o si el maestro lo prefiere en un sistema operativo Linux.

1.6 Metodología Para hacer la selección de las herramientas CASE a analizar se recomienda utilizar la siguiente metodología:

- a) Investigación de herramientas CASE existentes. En la actualidad, existe un amplio número de herramientas CASE libres y con licencia para desarrollar una aplicación orientada a objetos, de las diferentes fuentes bibliográficas existentes del tema se recomienda hacer un listado de al menos 10 herramientas CASE para aplicaciones orientadas a objetos.
- b) Agrupar las herramientas CASE seleccionadas, de acuerdo a la tabla 1, en donde a partir de dicha lista se realizará una investigación con el objetivo de establecer para cada una lo siguiente:

CASE	Sitio Web	Página disponible	Última versión	Fecha primera versión	Fecha Última Versión.

Tabla1. Lista de herramientas CASE para aplicaciones orientadas a objetos.

- c) Con las herramientas CASE anteriores, llenar la tabla siguiente considerando el criterio de madurez siguiente:

Madurez. Define a este criterio como el crecimiento del CASE a través del tiempo, desde su creación hasta la actualidad, adaptándose a las nuevas tecnologías, procesos y modelos, con el objetivo de cubrir de mejor manera los requerimientos del usuario y adaptarse a nuevos entornos, sin perder rendimiento y compatibilidad, logrando con esto mantener o mejorar su presencia en el mercado.

De acuerdo a la definición anterior, llenar la tabla 2, en donde se muestre una lista de los CASE seleccionados ordenados de forma ascendente de acuerdo a la liberación de la primera versión, pero cuya última versión corresponda al año 2014.

CASE	Última Versión	Fecha primera versión	Fecha última versión.

Tabla 2. Lista de herramientas CASE para aplicaciones orientadas a objetos en base a la última versión

- d) Con las herramientas CASE anteriores, llenar la tabla siguiente considerando el criterio de documentación siguiente:

Documentación. La cantidad de documentación como manuales, tutoriales, blogs y foros de discusión es un aspecto importante, ya que con ello permite conocer conceptos básicos y resolver problemas. Para tomar en cuenta la documentación existente de cada CASE, es necesaria la realización de una serie de consultas en tres sitios de búsqueda de internet más usados en el mundo, en este caso Google, Yahoo!, Live Search, Con esta información se recomienda llenar la tabla 3.

CASE	Google	Yahoo!	Live Search

Tabla 3. Lista de herramientas CASE para aplicaciones orientadas a objetos en base a la documentación (número de búsquedas encontradas).

e) Al mismo tiempo, se deberá de realizar una búsqueda del número de libros existentes en el mercado ya que representan el mejor medio de aprendizaje a dichas tecnologías, en el sitio www.amazon.com y llenar la tabla 4.

CASE	No. de consultas en amazon.com

Tabla 4. Consulta en Amazon.com

f) Con esta información determinar las 4 herramientas CASE orientadas a objetos más utilizadas en la actualidad, y para cada una de ella y de acuerdo a las características revisadas en la introducción completar la tabla 5.

PROPIEDADES	CASE I	CASE II	CASE III	CASE IV
ASPECTOS GENERALES/ENFOQUE PROCEDIMENTAL				
Observaciones Generales				
Ámbitos de Utilización.				
Plataforma				
Precio				
Apoyo Metodológico				
Soporte Completo en UML.				
Ingeniería Inversa				
Métricas				
Apoyo a lenguajes formales				
Actualización				

SOPORTE AL MODELO ARQUITECTÓNICO
Soporte ADL Servicios de Presentación Servicios de Lógica de Negocio Servicios de Persistencia
APOYOS VARIOS
Robustez Proyectos Repositorio Área de Trabajo Aplicación Colaboración entre usuarios Intercambio Autogeneración Versionamiento Visualización de Diagramas Impresión Exportación Edición de Código

Tabla 5. Comparación de las características de herramientas CASE orientada a objetos.

g) Generar una conclusión de las 4 herramientas seleccionadas, resaltando ventajas y desventajas de las mismas para definir cuál sería la herramienta optima a utilizar en las prácticas siguientes.

1.7 Sugerencias didácticas.

En un trabajo inicial por equipos se recomienda que el alumno genere un trabajo que cubra los siguientes puntos:

Clasificación y caracterización de las diferentes herramientas CASE.
 Investigación de 10 herramientas CASE orientadas a objetos y que utilicen UML.
 Llenado de las tablas 1 a 4.

En una sesión de debate cada equipo presentará el resultado de sus tablas estableciendo coincidencias y diferencias para establecer en grupo las 4 herramientas a caracterizar de acuerdo a la tabla 5.

El docente revisará con los alumnos los criterios de la tabla 5 a fin de comprenderlas exhaustivamente y poder llenarla por equipo.

Una vez llena se hará otra sesión de debate a fin de determinar la herramienta más recomendable a utilizar por el grupo.

Es recomendable que dentro de las herramientas seleccionadas el alumno considere a la herramienta de Enterprise Architect (EA), como una opción de herramienta CASE licenciada ya que el tecnológico tiene una serie de licencias que puede actualizar y se trata de una herramienta de gran actualidad y altamente competitiva en el mercado, además de ser muy utilizada por los maestros del área.

1.8 Reporte del alumno (resultados)

Reporte denominado: Especificación del sistema XX que contenga:

- -Propuesta del proyecto XX. Con el contenido descrito en la metodología.
- -Las cinco tablas incluidas en la metodología.
- -Conclusiones y experiencias por alumnos del trabajo práctico realizado.

1.9 Bibliografía preliminar

1. Amazon, la mejor para hacer compras online. (2012). El economista .Recuperado el 1 de febrero del 2013, de <http://eleconomista.com.mx/tecnociencia/2012/12/27/amazon-mejorhacer-compras-online>.
2. Ángel Hernández Cruz (2013). Análisis Comparativo de Frameworks para el desarrollo de aplicaciones Web en Java. Tesis de licenciatura en Informática de la Universidad de la Sierra Juárez.
3. Clarion software (2003). Clarion Business Rules Manager Application Guide. SoftVelocity. <http://www.softvelocity.com/products/applicationguides.htm>.
4. Dewayne E. Perry; WOLF, Alexander L. (2003) Foundations for the Study of Software Architecture.
5. Dorsey, Paul; Hudicka, Joseph R. (1999). Oracle 8: Diseño de bases de datos con UML. México: Mcgraw Hill Interamericana. 394 p.
6. Garlan, David; Shaw Mary (2003). An Introduction to Software Architecture. School of Computer Science. Carnegie Mellon University, Pittsburgh. <http://www.fi.uba.ar/materias/7572/review0000.pdf>
7. Georgas, John(2003). Recommendations for Architecture-Centric Software support selfadaptive behavior. http://sunset.usc.edu/gsaw/gsaw2003/s8a/bo_georgas.pdf
8. Gómez del Moral, Martín; Consuegra, Diego (2003). Tendencia de futuro de las herramientas UML. <http://www.diatel.upm.es/malvarez/UML/Tendencias.html>.
9. Juan Bernardo Quintero, Raquel Anaya de Páez, Juan Carlos Marín, Alex Bilbao López (2005). Un estudio comparativo de herramientas para el modelado con UML

10. Mc Connell, Steve(1997). Desarrollo y gestión de proyectos informáticos. España: McGraw Hill Interamericana.
11. Mejía Álvarez, Pedro (2003). Reingeniería de software. Departamento de Ingeniería Eléctrica CINVESTAV-IPN, México. <http://delta.cs.cinvestav.mx/~pmejia/softeng/>
12. Moreno García, María (2003). Técnicas Formales de Especificación. Universidad de Salamanca, España. <http://lisisu02.usal.es/~mmoreno/> ASTema.pdf
13. OMG (2003). UML 2.0: DraftSpecification. <http://www.uml.org/uml>
14. Sanchez de La Cruz, María Mar;Casar Tenorio, MariSol (2003). Reingeniería de bases de datos relacionales. Universidad de Castilla, La Mancha. (<http://alarcos.inf-cr.uclm.es/doc/bdavanzadas/reingenieria.pdf>)

Práctica No.2

2.1 Nombre. Aplicación de los procesos básicos de la Ingeniería de Requisitos

2.2 Objetivo. Que el alumno comprenda y aplique los 4 procedimientos básicos que conforman la Ingeniería de Requisitos para la definición y especificación de un sistema de cómputo real.

2.3 Introducción. La Ingeniería de Requerimientos es un área de la Ingeniería del Software que se orienta a los procesos que nos permiten la definición y especificación de sistemas a ser elaborados por un desarrollador, su meta es crear y mantener un documento de requerimientos del sistema.

En otras palabras la ingeniería de requerimientos se considera como el proceso de aplicar un método de análisis que comprende analizar el sistema y desarrollar un conjunto de modelos gráficos del mismo, como los modelos de casos de uso, que sirven como una especificación del sistema. El conjunto de modelos describe el comportamiento del sistema al cual se le agregan notas con información adicional que detallan, por ejemplo, el rendimiento o fiabilidad requeridos.

El proceso principal de la Ingeniería de los Requerimientos se compone de cuatro subprocesos de alto nivel que son los siguientes:

- **El estudio de viabilidad:** Que nos permiten evaluar si el sistema es útil para el negocio.
- **La Obtención y análisis:** Que nos permite hacer el descubrimiento de los requerimientos que presenta el usuario acerca del sistema a desarrollar.
- **La especificación de los requerimientos.** Que nos permite transformar estos requerimientos en formularios estándares.
- **La validación de los requerimientos.** Que nos permite verificar que los requerimientos realmente definen el sistema que quiere el cliente.

En la figura 1 se puede observar la relación entre estos 4 subprocesos, así como los entregables básicos que pueden obtenerse de ellos.

Su descripción se hace a continuación.

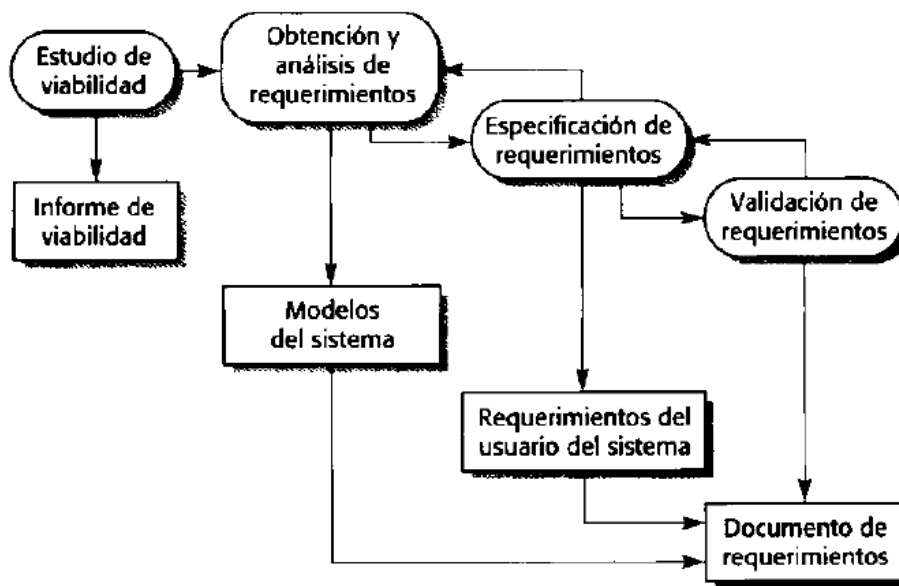


Figura 1. Los subprocesos del Proceso de Ingeniería de Requerimientos.

Estudio de viabilidad. Para todos los sistemas nuevos, el proceso de ingeniería de requerimientos debería empezar con un estudio de viabilidad. La entrada de éste es un conjunto de requerimientos de negocio preliminares, una descripción resumida del sistema y de cómo éste pretende contribuir a los procesos del negocio. Los resultados del estudio de viabilidad deberían ser un informe que recomiende si merece o no la pena seguir con la ingeniería de requerimientos y el proceso de desarrollo del sistema, es básico el resolver las siguientes preguntas: ¿Contribuye el sistema a los objetivos generales de la organización? ¿Se puede implementar el sistema utilizando la tecnología actual y dentro de las restricciones de coste y tiempo? ¿Puede integrarse el sistema con otros sistemas existentes en la organización?

Llevar a cabo un estudio de viabilidad comprende la evaluación y recopilación de la información, y la redacción de informes. La fase de evaluación de la información identifica la información requerida para contestar las tres preguntas anteriores. Una vez que dicha información se ha identificado, se debería hablar con los posibles usuarios para descubrir las respuestas a estas preguntas. Algunos ejemplos de preguntas posibles son:

1. ¿Cómo se las arreglaría la organización si no se implementara este sistema?
2. ¿Cuáles son los problemas con los procesos actuales y cómo ayudaría un sistema nuevo a aliviarlos?
3. ¿Cuál es la contribución directa que hará el sistema a los objetivos y requerimientos del negocio?
4. ¿La información se puede obtener y transferir a otros sistemas de la organización?
5. ¿Requiere el sistema tecnología que no se ha utilizado previamente en la organización?
6. ¿A qué debe ayudar el sistema y a qué no necesita ayudar?

En un estudio de viabilidad, se pueden consultar las fuentes de información, como los jefes de los departamentos donde se utilizará el sistema, los ingenieros de software que están familiarizados con el tipo de sistema propuesto, los expertos en tecnología y los usuarios finales del sistema. Normalmente se debería intentar completar un estudio de viabilidad en dos o tres semanas.

Una vez que se tiene la información, se redacta el informe del estudio de viabilidad. Debería hacerse una recomendación sobre si debe continuar o no el desarrollo del sistema. En el informe se pueden proponer cambios en el alcance, el presupuesto y la confección de agendas del sistema y sugerir requerimientos adicionales.

Análisis Costo Beneficio. La técnica de Análisis de Costo/Beneficio, tiene como objetivo fundamental proporcionar una medida de la rentabilidad de un proyecto, mediante la comparación de los costos previstos con los beneficios esperados en la realización del mismo. Esta técnica se debe utilizar al comparar proyectos para la toma de decisiones.

Un análisis Costo/Beneficio por sí solo no es una guía clara para tomar una buena decisión. Existen otros puntos que deben ser tomados en cuenta, como son: la moral de los empleados, la seguridad, las obligaciones legales y la satisfacción del cliente.

El análisis Costo-Beneficio, permite definir la factibilidad de las alternativas planteadas o de un proyecto a ser desarrollado.

La utilidad de la presente técnica es la siguiente:

- Para valorar la necesidad y oportunidad de la realización de un proyecto.
- Para seleccionar la alternativa más beneficiosa de un proyecto.
- Para estimar adecuadamente los recursos económicos necesarios, en el plazo de realización de un proyecto.

El análisis Costo/Beneficio involucra los siguientes 7 pasos:

1. Llevar a cabo una lluvia de ideas o reunir datos provenientes de factores importantes relacionados con cada una de sus decisiones.
2. Elaborar dos listas, la primera con los requerimientos para implantar el proyecto y la segunda con los beneficios que traerá el nuevo sistema

Antes de redactar la lista es necesario tener presente que los costos son tangibles, es decir, se pueden medir en alguna unidad económica, mientras que los beneficios pueden ser tangibles y no tangibles, es decir pueden darse en forma objetiva o subjetiva.

3. Determinar los costos relacionados con cada factor. Algunos costos como la mano de obra, serán exactos mientras que otros deberán ser estimados.
4. Sumar los costos totales para cada decisión propuesta.

5. Determinar los beneficios en alguna unidad económica para cada decisión.
6. Poner las cifras de los costos y beneficios totales en una forma de relación donde los beneficios son el numerador y los costos son el denominador: $\text{beneficios /costos}$
7. Comparar las relaciones Beneficios a costos para las diferentes decisiones propuestas

La mejor solución, en términos financieros, es aquélla con la relación más alta.

Obtención y Análisis de requerimientos. En esta actividad, los ingenieros de requerimientos trabajan con los clientes y usuarios finales del sistema para determinar el dominio de aplicación, qué servicios debe proporcionar el sistema, el rendimiento requerido del sistema, las restricciones hardware, etcétera.

La obtención y análisis de requerimientos pueden afectar a varias personas de la organización. Nosotros utilizaremos el término usuario para referirnos a cualquier persona o grupo que se verá afectado por el sistema, directa o indirectamente.

Las actividades del proceso de obtención y análisis pueden ser en forma genérica las siguientes:

- Descubrimiento de requerimientos. Consiste en aplicar técnicas básicas de análisis como son: entrevistas, observación, escenarios, técnicas JAD, elaboración de prototipos etc. a fin de identificar los principales requerimientos funcionales y no funcionales a partir de las fuentes primarias y secundarias de información que estén a nuestras manos (usuarios, documentos, manuales, reportes, etc.).
- Clasificación y organización de requerimientos. Esta actividad toma la recopilación no estructurada de requerimientos, los agrupa y expresa mediante alguna ayuda para el caso como pueden los diagramas de requerimientos con la notación extendida de Sysml (Systems Modeling Language), que consiste en un lenguaje de especificación de sistemas y un subconjunto ampliado de UML 2.0.
- Ordenación por prioridades y negociación de requerimientos. Inevitablemente, cuando existen muchos usuarios involucrados, los requerimientos entrarán en conflicto. Esta actividad se refiere a ordenar según las prioridades los requerimientos, y a encontrar y resolver los requerimientos en conflicto a través de la negociación.
- Documentación de requerimientos. Se documentan los requerimientos. Se pueden producir documentos de requerimientos formales o informales, es un proceso iterativo con retroalimentación continua de cada actividad a las otras actividades. En la etapa de la documentación de requerimientos los

requerimientos que se han obtenido se documentan de preferencia con Sysml como lo citamos anteriormente. En esta etapa se puede producir una versión inicial del documento de requerimientos del sistema, pero faltara definir más requerimientos completar los descritos.

Especificación de requerimientos. Una vez resuelto el proceso anterior, el siguiente proceso tiene que ver con la especificación de requisitos del software mediante una documentación formal.

En esta actividad se generan las bases para el desarrollo del documento, con el mismo nombre, que contiene una descripción completa de las necesidades y funcionalidades del sistema que será desarrollado; describe el alcance del sistema y la forma en cómo hará sus funciones. Además se definen los requerimientos funcionales y los no funcionales que darán base al documento de requerimientos.

Es importante destacar que la especificación de requerimientos es el resultado final de las actividades de análisis y evaluación de requerimientos; el documento resultante será utilizado como fuente básica de comunicación entre los clientes, usuarios finales, analistas de sistema, personal de pruebas, y todo aquel involucrado en la implementación del sistema.

Los clientes y usuarios utilizan el documento generado en este punto para comparar si lo que se está proponiendo, coincide con las necesidades de la empresa. Los analistas y programadores la utilizan para determinar el producto que debe desarrollarse. El personal de pruebas elaborará las pruebas funcionales y de sistemas en base a este documento. Para el administrador del proyecto sirve como referencia y control de la evolución del sistema.

La especificación de los requerimientos descansa en estos principios básicos.

Principios de Especificación. La especificación, independientemente del modo en que se realice, puede ser vista como un proceso de representación. Los requerimientos se representan de forma que conduzcan finalmente a una correcta implementación del software. Baltzer y Goldman proponen ocho principios para una buena especificación:

Principio #1. Este principio tiene que ver con separar la funcionalidad de la implementación. Ya que por definición, una especificación es una descripción de lo que se desea, en vez de como se implementa.

Principio #2. Se necesita un lenguaje de especificación de sistemas orientado al proceso, o sea que debe emplearse una descripción orientada al proceso, en la cual la especificación se consigue mediante la especificación de un modelo del comportamiento deseado en términos de respuestas funcionales, a distintos estímulos al entorno.

Principio #3. Una especificación debe abarcar el sistema del cual el software es una componente. Un sistema está compuesto de componentes que interactúan. Solo dentro del contexto del sistema completo y de la interacción entre sus partes puede ser definido el comportamiento de una componente específica. En general, un sistema puede ser modelado

como una colección de objetos pasivos y activos. Estos objetos están interrelacionados y dichas relaciones entre los objetos cambian con el tiempo. Estas relaciones dinámicas suministran los estímulos a los cuales los objetos activos, llamados agentes, responden. Las respuestas pueden causar posteriormente cambios y, por tanto, estímulos adicionales a los cuales los agentes deben responder

Principio #4. Una especificación debe abarcar el entorno en el que el sistema opera. Similarmente, el entorno en el que opera el sistema y con el que interactúa debe ser especificado. Afortunadamente, esto tan solo necesita reconocer que el propio entorno es un sistema compuesto de objetos que interactúan, pasivos y activos, de los cuales el sistema especificado es una agente. Los otros agentes, los cuales son por definición inalterables debido a que son parte del entorno, limitan el ámbito del diseño subsecuente y de la implementación. De hecho, la única diferencia entre el sistema y su entorno es que el esfuerzo de diseño e implementación subsecuente opera exclusivamente sobre la especificación del sistema. La especificación del entorno facilita que se especifique la interfaz del sistema de la misma forma que el propio sistema, en vez de introducir otro formalismo.

Principio #5. Una especificación de sistema debe ser un modelo cognitivo. La especificación de un sistema debe ser un modelo cognitivo, en vez de un modelo de diseño o implementación. Debe describir un sistema tal como es percibido por su comunidad de usuario. Los objetivos que manipula deben corresponderse con objetos reales de dicho dominio; los agentes deben modelar los individuos, organizaciones y equipo de ese dominio; y las acciones que ejecutan deben modelar lo que realmente ocurre en el dominio. Además, debe ser posible incorporar en la especificación las reglas o leyes que gobiernan los objetos del dominio. Algunas de estas leyes proscriben ciertos estados del sistema (tal como "dos objetos no pueden estar en el mismo lugar al mismo tiempo"), y por tanto limitan el comportamiento de los agentes o indican la necesidad de una posterior elaboración para prevenir que surjan estos estados

Principio #6. Una especificación debe ser operacional. La especificación debe ser completa y lo bastante formal para que pueda usarse para determinar si una implementación propuesta satisface la especificación de pruebas elegidas arbitrariamente. Esto es, dado el resultado de una implementación sobre algún conjunto arbitrario de datos elegibles, debe ser posible usar la especificación para validar estos resultados.

Principio #7. La especificación del sistema debe ser tolerante con la incompletitud y aumentable. Ninguna especificación puede ser siempre totalmente completa. El entorno en el que existe es demasiado complejo para ello. Una especificación es siempre un modelo, una abstracción, de alguna situación real (o imaginada). Por tanto, será incompleta. Además, al ser formulada existirán muchos niveles de detalle.

Principio #8. Una especificación debe ser localizada y débilmente acoplada. Los principios anteriores tratan con la especificación como una entidad estática. Esta surge de la dinámica de la especificación. Debe ser reconocido que aunque el principal propósito de una especificación sea servir como base para el diseño e implementación de algún sistema, no es un objeto estático pre compuesto, sino un objeto dinámico que sufre considerables

modificaciones. Tales modificaciones se presentan en tres actividades principales: formulación, cuando se está creando una especificación inicial; desarrollo, cuando la especificación se está elaborando durante el proceso iterativo de diseño e implementación; y mantenimiento, cuando la especificación se cambia para reflejar un entorno modificado y requerimientos funcionales adicionales.

Validación de requerimientos. La validación de requerimientos trata de mostrar que éstos realmente definen el sistema que el cliente desea. Coincide parcialmente con el análisis ya que éste implica encontrar problemas con los requerimientos. La validación de requerimientos es importante debido a que los errores en el documento de requerimientos pueden conducir a importantes costos al repetir el trabajo cuando son descubiertos durante el desarrollo o después de que el sistema esté en uso. El costo de arreglar un problema en los requerimientos haciendo un cambio en el sistema es mucho mayor que reparar los errores de diseño o los de codificación. La razón de esto es que un cambio en los requerimientos normalmente significa que el diseño y la implementación del sistema también deben cambiar y que éste debe probarse nuevamente.

Durante el proceso de validación de requerimientos, se deben llevar a cabo verificaciones sobre requerimientos en el documento de requerimientos. Estas verificaciones comprenden:

- *Verificaciones de validez.* Un usuario puede pensar que se necesita un sistema para llevar a cabo ciertas funciones. Sin embargo, el razonamiento y el análisis pueden identificar que se requieren funciones adicionales o diferentes. Los sistemas tienen diversos usuarios con diferentes necesidades, y cualquier conjunto de requerimientos es inevitablemente un compromiso en el entorno del usuario.
- *Verificaciones de consistencia.* Los requerimientos en el documento no deben contradecirse. Esto es, no debe haber restricciones o descripciones contradictorias de la misma función del sistema.
- *Verificaciones de completitud.* El documento de requerimientos debe incluir requerimientos que definan todas las funciones y restricciones propuestas por el usuario del sistema.
- *Verificaciones de realismo.* Utilizando el conocimiento de la tecnología existente, los requerimientos deben verificarse para asegurar que se pueden implementar. Estas verificaciones también deben tener en cuenta el presupuesto y la confección de agendas para el desarrollo del sistema.
- *Verificabilidad.* Para reducir la posibilidad de discusiones entre el cliente y el contratista, los requerimientos del sistema siempre deben redactarse de tal forma que sean verificables. Esto significa que debe poder escribir un conjunto de pruebas que demuestren que el sistema a entregar cumple cada uno de los requerimientos especificados.

Pueden utilizarse, en conjunto o de forma individual, varias técnicas de validación de requerimientos:

- *Revisiones de requerimientos.* Los requerimientos son analizados sistemáticamente por un equipo de revisores.

- *Construcción de prototipos* En este enfoque de validación, se muestra un modelo ejecutable del sistema a los usuarios finales y a los clientes. Éstos pueden experimentar con este modelo para ver si cumple sus necesidades reales.
- *Generación de casos de prueba.* Los requerimientos deben poder probarse. Si las pruebas para éstos se conciben como parte del proceso de validación, a menudo revela los problemas en los requerimientos. Si una prueba es difícil o imposible de diseñar, normalmente significa que los requerimientos serán difíciles de implementar y deberían ser considerados nuevamente. Desarrollar pruebas para los requerimientos del usuario antes de que se escriba código es una parte fundamental de la programación extrema.

No deben subestimarse los problemas en la validación de requerimientos. Es difícil demostrar que un conjunto de requerimientos cumple las necesidades del usuario. Los usuarios deben imaginarse el sistema en funcionamiento y cómo éste encajaría en su trabajo. Para los informáticos expertos es difícil llevar a cabo este tipo de análisis abstracto, pero para los usuarios del sistema es aún más difícil. Como consecuencia, rara vez se encuentran todos los problemas en los requerimientos durante el proceso de validación de éstos. Es inevitable que haya cambios adicionales de requerimientos para corregir las omisiones y las malas interpretaciones después de que el documento de requerimientos haya sido aprobado.

Revisión de requerimientos. Es un proceso manual que involucra a personas tanto de la organización del cliente como de la del contratista. Ellos verifican el documento de requerimientos en cuanto a anomalías y omisiones. El proceso de revisión se puede gestionar de la misma forma que las inspecciones de programas. Alternativamente, se puede organizar como una actividad más amplia con diferentes personas que verifican diferentes partes del documento.

Las revisiones de requerimientos pueden ser informales o formales. Las informales sencillamente implican que los contratistas deben tratar los requerimientos con tantos usuarios del sistema como sea posible. Es sorprendente la frecuencia con la que finaliza la comunicación entre los desarrolladores y los usuarios del sistema después de la obtención de requerimientos y que no exista continuación de que los requerimientos documentados son realmente lo que dijeron que deseaban los usuarios.

Antes de llevar a cabo una reunión para una revisión formal, muchos problemas se pueden detectar simplemente hablando del sistema con los usuarios.

En la revisión formal de requerimientos, el equipo de desarrollo debe «conducir» al cliente a través de los requerimientos del sistema, explicándole las implicaciones de cada requerimiento. El equipo de revisión debe verificar cada requerimiento para la consistencia además de verificar los requerimientos como un todo para la completitud. Los revisores también pueden comprobar la:

- Verificabilidad. ¿Puede probarse el requerimiento de modo realista?

- Comprensibilidad. ¿Las personas que adquieren el sistema o los usuarios finales comprenden correctamente el requerimiento?
- Rastreabilidad. ¿Está claramente establecido el origen del requerimiento? Puede tener que volver a la fuente del requerimiento para evaluar el impacto del cambio. La rastreabilidad es importante ya que permite evaluar el impacto del cambio en el resto del sistema
- Adaptabilidad. ¿Es adaptable el requerimiento? Es decir, ¿puede cambiarse el requerimiento sin causar efectos de gran escala en los otros requerimientos del sistema?

Los conflictos, contradicciones, errores y omisiones en los requerimientos deben ser señalados por los revisores y registrarse formalmente en el informe de revisión. Queda en los usuarios, la persona que adquiere el sistema y el desarrollador de éste negociar una solución para estos problemas identificados.

2.4 Especificar la correlación con el o los temas y subtemas del programa. El contenido de esta práctica se relaciona con el programa de la materia en donde el alumno sea capaz identificar el concepto de Ingeniería de requerimientos así como los supprocesos que la componen y como se llevan a cabo estos para la generación de la especificación de un sistema. La idea planteada en el programa es que el alumno conozca la teoría del proceso de ingeniería de requerimientos y a partir de esta pueda aplicarla en un proyecto real.

2.5 Material y equipo necesario. Para esta práctica el alumno requerirá del uso de algunas técnicas para la obtención de requerimientos dependiendo de su proyecto, esto puede incluir: Guiones de entrevistas a usuarios, plantillas de cuestionarios, elaboración de prototipos y escenarios reales de solución de problemas, las cuales se deberán de acompañar del material necesario para la obtención de una lista preliminar de requerimientos.

Al mismo tiempo se recomienda el uso de herramientas para completar el material anterior, que pueden ser una cámara de video y una grabadora (o bien un teléfono celular con cámara de video) que auxiliarán adecuadamente en el registro de las entrevistas que se apliquen a los usuarios.

El uso de una herramienta CASE que de acuerdo a la primera práctica podría ser el Enterprise Architect ver X (la versión se definirá de común acuerdo con el profesor), se utilizará para elaborar un diagrama de requerimientos y el correspondiente diagrama de casos de uso, que complementarán la especificación del sistema y se elaborarán en la práctica 3.

También utilizará uno de los estándares más utilizados para elaborar un documento de requerimientos conocido como IEEE/ANSI 830-1998 (IEE1988)

2.6 Metodología. Para hacer el documento de requerimientos de un sistema se recomienda seguir la siguiente metodología.

a) La primera tarea será que el alumno organizado por equipos (no más de 4 alumnos sería lo ideal), identifique en alguna empresa, negocio familiar o en la propia escuela cerca de la zona un problema real que pueda ser resuelto mediante la aplicación de un sistema de cómputo y haga la propuesta que el maestro pueda validar y aceptar en base a una serie de requisitos que se pongan de inicio (algunos podrían ser: número de casos de uso, cantidad de subsistemas que podrían conformarlo, área de aplicación del sistema etc.).

b) Del sistema propuesto él alumno por equipos elaborará un pre-análisis utilizando para ello algunas de las herramientas de obtención de requerimientos conocidas como: entrevista, cuestionario, observación, escenarios, o algunos de los recomendados por el profesor (se recomienda hacer unas 2 visitas al sitio en donde se va a elaborar el sistema como mínimo y hacer el registro correspondiente de las necesidades del usuario mediante una grabadora o una cámara). En estas visitas deberá quedar definido claramente: El problema a resolver y su delimitación, tamaño del sistema o número de subsistemas que lo conformarán, cantidad y tipo de usuarios, objetivo general del sistema y objetivos específicos (subsistemas).

c) Después del pre-análisis los alumnos organizados por equipo elaborarán un estudio Costo-Beneficio que sirva de apoyo para concluir si es viable el sistema propuesto. Este estudio se hará considerando lo establecido en la introducción de esta práctica y el formato a llenar es el siguiente, adjuntar una conclusión del estudio para dejar claramente establecida su viabilidad, en caso de que esta sea positiva seguir con el inciso d de esta metodología mientras que en caso contrario se recomienda cambiar el proyecto.

Característica	\$ Costo	\$ Beneficio	Costo/Beneficio	Deseable	No Deseable

Tabla 1. Estudio Costo-Beneficio

d) Generar un diagrama de requerimientos del sistema utilizando el EA (Práctica 3).

e) Generar un diagrama de casos de uso del sistema utilizando el EA (Práctica 3).

f) Completar en el diagrama anterior la descripción del escenario y los requerimientos internos para cada caso de uso (Práctica 3).

g) Completar el documento de requerimientos de acuerdo al índice IEEE/ANSI 830-1998.

1. Introducción
 - 1.1 Propósito
 - 1.2 Alcance
 - 1.3 Definiciones, siglas, y abreviaciones
 - 1.4 Referencias
 - 1.5 Apreciación global
2. Descripción global
 - 2.1 Perspectiva del producto
 - 2.2 Funciones del producto
 - 2.3 Características del/los usuarios
 - 2.4 Restricciones técnicas generales.
 - 2.5 Suposiciones y dependencias
3. Diagramas de casos de uso del sistema (práctica 3)
4. Diagramas de Requerimientos del sistema (práctica 3).

Nota: Es importante indicar que los puntos 3 y 4 de este documento quedará pendiente para ser completados en la práctica 3.

h). Hacer un documento de “Validación de Requerimientos” descritos en el inciso d. El objetivo es enumerar una prueba de tipo funcional para cada requerimiento de manera de poder validar el cumplimiento del mismo, o sea que por cada requerimiento redactar un caso de prueba funcional.

2.7 Sugerencias didácticas. Es importante que al término de esta práctica el alumno (después de hacer una revisión de problemáticas que existan en su entorno en negocios, organizaciones empresariales, escuelas, etc.); defina de común acuerdo con el profesor el proyecto de un sistema de cómputo. Este proyecto deberá de cumplir con criterios de extensión y dificultad adecuados para poder elaborarse a lo largo de 3 semestres.

Se recomienda que la propuesta del proyecto se complete en no más de 3 semanas y considerar que es solamente a nivel de pre-análisis, lo que significa que en unas 2 visita de los alumnos a la empresa o negocio podrá sacar la información suficiente para elaborar el estudio costo-beneficio y determinar la factibilidad o no del sistema.

Se recomienda tomar en cuenta los siguientes lineamientos para el proyecto propuesto: Que pueda estructurarse en al menos un subsistema por cada alumno (4 subsistemas para equipos de 4 personas), que se pueda resolver adecuadamente por el paradigma MVC y que cuente con un buen número de usuarios (3 como mínimo). Además de que deberá de tratarse de un sistema necesario que resuelva una problemática real.

Otro punto importante a cubrir es que por equipo se seleccione al menos 3 de las metodologías de obtención de requerimientos para un sistema.

Para el segundo punto de la metodología se recomienda llevar a cabo un estudio de costo-beneficio que servirá para establecer la viabilidad del sistema que se pretende desarrollar.

Para el tercer punto el alumno deberá de conocer las características que debe de cumplir un requerimiento de un sistema y elaborar un listado de requerimientos, dividiéndolos en funcionales y no funcionales los cuales deberán ser revisados por el profesor de manera que cumplan con las características establecidas.

En el cuarto punto el alumno deberá de leer completamente la bibliografía indicada en la metodología y comentarla mediante un debate dentro del aula de forma que quede claramente entendido y pueda ser la base para que se desarrolle el documento respectivo. La parte de especificación se deberá de completar con un modelado de requerimientos y casos de uso que se completarán en la práctica 3.

Para el inciso h de la metodología el alumno deberá de elaborar al menos un caso de prueba por cada requerimiento, estableciendo el procedimiento de prueba del mismo.

2.8 Reporte de alumnos (Resultados).

El alumno deberá de completar los siguientes documentos como producto de esta práctica.

- Propuesta de proyecto que contenga al menos los siguientes puntos: Descripción del problema, justificación de la solución, antecedentes, estado del arte, propuesta de solución, problemas a resolver.
- Estudio costo beneficio, con conclusión de su viabilidad o no del proyecto.
- Documento de requerimientos del sistema que cumpla con todos los lineamientos establecidos en el estándar de la IEEE/ANSI 830-1998. (excepto puntos 3 y 4).
- Documentos de pruebas de requerimientos que cubra al menos un caso de prueba de tipo caja negra para cada uno de los requerimientos documentados en el inciso c.

2.9 Bibliografía.

1. Ian Sommerville (2008). Ingeniería del Software. Séptima Edición. Addison Wesley.
2. Prácticas recomendadas de la IEEE para la especificación de requerimientos de software :http://www.ctr.unican.es/asignaturas/is1/IEEE830_esp.pdf.
3. Senn, James A.(1992). "Análisis y Diseño de Sistemas de Información". Segunda Edición. McGraw Hill.
4. Freeman, P., (1980) "Requirements Analysis and Specification", Proc. Intl. Computer Technology Conf., ASME, San Francisco, August.
5. Balzer, R., and N. Goodman, (1979). "Principles of Good Software Specification", Proc, on Specifications of Reliable Software, IEEE

Práctica No.3

3.1 Nombre: Generación de un modelo de requisitos con la ayuda de una herramienta CASE.

3.2 Objetivo: Que el alumno pueda desarrollar un modelo de requisitos usando para lo mismo la herramienta CASE, cumpliendo con los entregables definidos dentro de la metodología RUP.

3.3 Introducción. El proceso unificado conocido como RUP, es un modelo de software que permite el desarrollo de software a gran escala, mediante un proceso continuo de pruebas y retroalimentación, garantizando el cumplimiento de ciertos estándares de calidad. Aunque con el inconveniente de generar mayor complejidad en los controles de administración del mismo. Sin embargo, los beneficios obtenidos recompensan el esfuerzo invertido en este aspecto.

El proceso de desarrollo RUP constituye un marco metodológico que define términos de metas estratégicas, objetivos, actividades y artefactos (documentación) requerido en cada fase de desarrollo. Esto permite enfocar esfuerzo de los recursos humanos en términos de habilidades, competencias y capacidades a asumir en roles específicos con responsabilidades bien definidas a lo largo del proceso.

El RUP se puede estructurar en la forma genérica que se muestra a continuación:

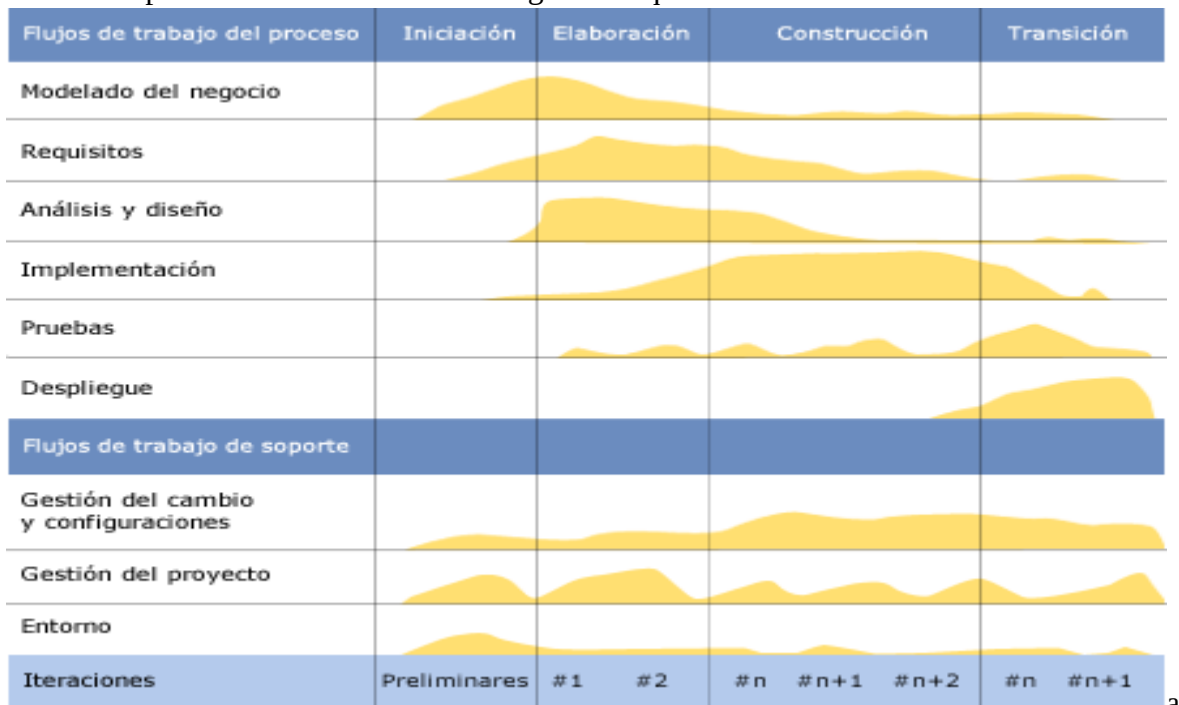


Figura 1. Estructura del ciclo de vida del proceso de desarrollo unificado

La descripción de la figura 1 es la siguiente en el eje horizontal se pueden observar las siguientes fases genéricas:

Fase de Iniciación. Esta fase tiene como propósito definir y acordar el alcance del proyecto con los patrocinadores, identificar los riesgos potenciales asociados al proyecto, proponer una visión muy general de la arquitectura de software y producir el plan de las fases y de iteraciones.

Fase de elaboración. En la fase de elaboración se seleccionan los casos de uso que permiten definir la arquitectura base del sistema y se desarrollaran en esta fase, se realiza la especificación de los casos de uso seleccionados y el primer análisis del dominio del problema, para diseñar una solución preliminar.

Fase de construcción. El propósito de esta fase es completar la funcionalidad del sistema, para ello se deben clarificar los requerimientos pendientes, administrar los cambios de acuerdo a las evaluaciones realizadas por los usuarios y se realizan las mejoras para el proyecto.

Fase de transición. El propósito de esta fase es asegurar que el software esté disponible para los usuarios finales, ajustar los errores y defectos encontrados en las pruebas de aceptación, capacitar a los usuarios y proveer el soporte técnico necesario. Se debe verificar que el producto cumpla con las especificaciones entregadas por las personas involucradas en el proyecto.

Para el eje vertical los flujos de trabajo que nos permiten la generación de los artefactos correspondientes pueden ser en forma general los siguientes:

Flujo de Trabajo de Requisitos. Es el flujo de trabajo junto con el de modelado de negocios que se desarrolla principalmente dentro de esta metodología y tiene como resultado el desarrollo de un primer modelo: **Modelo de Requisitos**, el cual sirve de base para la formación de todos los demás modelos en el desarrollo de software. En general, cualquier cambio en la funcionalidad del sistema es más fácil de hacer en esta etapa y con menores consecuencias. El modelo de requisitos que desarrollaremos se basa en la metodología Objectory (Jacobson et al. 1992), cimentado en el modelo de casos de uso el cuál irá evolucionando conforme pasan los modelos siguientes. Además de este modelo se complementa con otros dos que revisaremos más adelante en esta misma práctica.

Flujo de Trabajo de Análisis y Diseño: Este flujo permite la generación del **modelo de Análisis** el cual permite que la funcionalidad especificada en el modelo de casos de uso se estructure en un modelo que sea estable con respecto a cambios y extensiones del sistema, en otras palabras representa un modelo lógico independiente del ambiente de implementación en el cual se desarrolle el sistema.

Otro modelo desarrollado en este flujo de trabajo es el **Modelo de Diseño**, en donde la funcionalidad de los casos de uso ya estructurada por el análisis es realizada por el modelo de diseño mediante la adaptación del sistema al ambiente de implementación real.

Flujo de trabajo de la Implementación. Con el cual se desarrolla el **Modelo de Implementación** que nos permite que los casos de uso se implementen mediante el código fuente correspondiente y resolver el problema para el cual fue creado.

Flujo de Trabajo de las Pruebas: Que da como resultado un **Modelo de Pruebas** mediante el cual los casos de uso son probados para validar el cumplimiento de los requerimientos establecidos en el modelo de Requisitos.

Flujo de Trabajo del Despliegue: Que da por resultado un **Modelo de Despliegue** para considerar los detalles de la distribución física del sistema en el sitio de operación.

Como citamos anteriormente la primera etapa que revisaremos dentro de esta práctica tiene que ver con la elaboración del modelo de requisitos del sistema el cual describiremos a continuación.

Modelo de Requisitos: El propósito del modelo de requisitos es comprender completamente el problema y sus implicaciones. Todos los modelos se desarrollan directamente de él. El modelo de requisitos no es un proceso mecánico, ya que el analista debe interactuar constantemente con el cliente para completar la información faltante, y así clarificar ambigüedades e inconsistencias. El analista debe separar entre los requisitos verdaderos (funcionales) y las decisiones relacionadas con el diseño e implementación (no funcionales). Se debe indicar cuales aspectos son obligatorios y cuales son opcionales para evitar restringir la flexibilidad de la implementación.

Para brindar una visión más completa del sistema todo modelo de requisitos deberá de elaborarse desde las tres perspectivas más importantes para la comprensión de todo sistema, que son: **El comportamiento, la presentación y la información del sistema**, lo que da lugar a los siguientes modelos:

El comportamiento. Basado directamente en el *modelo de casos de uso* y que nos permita especificar la funcionalidad que ofrece el sistema desde el punto de vista del usuario.

La presentación. Que nos permita especificar cómo interactúa el sistema con actores externos al ejecutar los casos de uso, en otras palabras: cómo se verán visualmente las *interfaces gráficas*, y qué funcionalidad ofrecerá cada una de ellas.

La información. Que se puede basar en un *modelo de clases y objetos* que nos permitan identificar rápidamente cuáles objetos podrán ser guardados en una base de datos, si ése fuera el caso. Además de establecer la manera de guardar información temporal en la aplicación.

Modelo de Comportamiento. Como su nombre lo dice este modelo nos permite conocer a detalle el comportamiento dinámico del sistema. Para este modelado nos ayudamos principalmente de los casos de uso, que describimos a continuación.

Casos de uso. Los casos de uso son una técnica que se basa en escenarios para la obtención de requerimientos. Estos actualmente se han convertido en una herramienta fundamental de la notación de UML.

Los diagramas de caso de uso son uno de los cinco tipos de diagramas en UML para modelar aspectos dinámicos de sistemas (diagramas de actividad, diagramas de estados, diagramas de secuencia y diagramas de colaboración son otros cuatro tipos de diagramas en UML para modelar los aspectos dinámicos de un sistema).

Los diagramas de casos de uso no son importantes solamente en el modelado del comportamiento de un sistema, sino también para el de un subsistema o una clase permitiéndonos visualizar, especificar, y documentar su comportamiento. Con esto logramos tener sistemas, subsistemas, y clases entendibles para presentar una vista exterior de cómo estos elementos pueden ser usados dentro del contexto. Los diagramas de caso de uso son también importantes para probar sistemas ejecutables a través de ingeniería hacia adelante y para comprender sistemas ejecutables a través de ingeniería inversa. Revisemos algo de teoría de estos diagramas.

Términos y conceptos. Un diagrama de casos de uso es un diagrama que muestra un conjunto de casos de uso, actores y sus relaciones (dependencias, generalización y relación de asociación). Estos diagramas pueden contener también notas y restricciones.

Utilidad. Se usan para el modelado de la vista estática de los casos de uso de un sistema. Esta primera vista soporta el comportamiento de un sistema el alejamiento de los servicios visibles que el sistema provee en el contexto del medio ambiente. Cuando se modela la vista estática de un caso de uso de un sistema, se pueden aplicar diagramas de casos de uso de dos maneras diferentes:

- a) Modelar el contexto de un sistema. Modelar el contexto de un sistema implica dibujar una línea alrededor del sistema completo y los actores principales fuera de esta línea. En este caso se aplican los diagramas de casos de uso para especificar los actores y el manejo de sus roles.
- b) Modelar las necesidades de un sistema. Modelar los requerimientos de un sistema implica especificar qué es lo que hará el sistema (desde un punto de vista exterior), independientemente de cómo el sistema lo hará. Se pueden aplicar diagramas de casos de uso para especificar el comportamiento deseado del sistema. De esta manera, un diagrama de casos de uso nos da una vista del sistema completo como una caja negra y se puede ver que hay fuera del sistema y como el sistema reacciona al exterior, pero no se puede ver como el sistema trabaja en su interior.

Técnicas Comunes de Modelado.

a) Modelado del Contexto de un Sistema. Dado un sistema cualquiera, algunas entidades pueden vivir dentro de este, y otras vivir fuera de este. Por ejemplo, en un sistema de validación de tarjeta de crédito, puedes encontrar algunas entidades como cuentas, transacciones, y agentes de detección de fraude dentro del sistema. Similarmente, podrás encontrar entidades como clientes de tarjeta de crédito e instituciones de venta fuera del sistema. Las entidades que viven dentro del sistema son responsables de su comportamiento.

Todas estas entidades externas que interactúan con el sistema constituyen su contexto o medio ambiente. En UML, puedes modelar el contexto de un sistema con un diagrama de casos de uso, enfatizando los actores que rodean el sistema.

Para modelar el contexto de un sistema se recomienda:

- Identificar los actores que rodean el sistema considerando cuales grupos requieren ayuda del sistema para ejecutar sus tareas; cuales grupos son necesarios para ejecutar las funciones del sistema; cuales grupos interactúan con hardware externo u otros sistemas de software; y cuales grupos ejecutan funciones secundarias para administración y mantenimiento.
- Organizar actores que son similares a algún otro en una asociación de tipo herencia.
- Poblar un diagrama de casos de uso con estos actores y especificar el curso de la comunicación desde cada actor hacia los casos de uso del sistema.

Esta misma técnica se aplica para modelar el contexto de un subsistema. Un sistema en un nivel de abstracción es frecuentemente un subsistema de un gran sistema en un alto nivel de abstracción. Modelando el contexto de un subsistema es útil cuando estas construyendo sistemas de sistemas interconectados.

b) Modelado de las necesidades del sistema. Un requerimiento es una característica de diseño, atributo, o comportamiento de un sistema. Con las necesidades de un sistema, se acuerda un pacto establecido entre las cosas fuera del sistema y el sistema por sí mismo, cuando se declara que se espera que el sistema realice. En la mayor parte no se cuida como el sistema lo hace, solo se cuida que es lo que hace. Cuando se construye un sistema es importante iniciar con acuerdos acerca de que es lo que hará el sistema, aunque se podrá desarrollar el entendimiento de esos requerimientos conforme iterativa e incrementalmente se desarrolle el sistema.

Los requerimientos pueden ser expresados en varias formas, desde texto hasta expresiones en un lenguaje formal, y cualquier cosa entre ellos. Muchos si no todos, de los requerimientos funcionales de un sistema pueden ser expresados como casos de uso, y los diagramas de casos de uso UML son esenciales para manejar estos requerimientos.

Para modelar los requerimientos de un sistema se recomienda:

- Establecer el contexto de un sistema para identificando los actores que lo rodean.

- Para cada actor, considerar el comportamiento que se espera o necesita el sistema proporcione.
- Nombrar los comportamientos comunes como casos de uso.
- Modelar casos de uso, actores y sus relaciones en un diagrama de casos de uso.
- Adornar los casos de uso con notas.

Ingeniería hacia adelante e Inversa. La mayoría de los diagramas UML, incluyendo clases, componentes, y diagramas de estado, son claros candidatos para ingeniería hacia adelante e inversa porque tienen una analogía en código ejecutable. Los diagramas de casos de uso describen como un elemento se comporta, no como es implementado, por esto no se le aplica directamente la ingeniería hacia adelante o hacia atrás.

Ingeniería hacia adelante. Proceso de transformar un modelo a código a través de un mapeo hacia un lenguaje de implementación. Un diagrama de casos de uso puede aplicársele ingeniería hacia adelante para formas de prueba para el elemento para el cual aplica. Cada caso de uso en un diagrama de casos de uso especifica un flujo de eventos (y variantes de esos flujos), y estos flujos especifican como se espera se comporte el elemento. Un caso de uso bien estructurado podrá especificar pre y post condiciones que pueden ser usadas para definir un estado inicial para la prueba y las post-condiciones pueden ser usadas para definir un criterio exitoso. Para cada caso de uso en un diagrama, se pueden crear pruebas que puedes ejecutar cada vez que se libera una nueva versión del elemento, de este modo se confirma que el elemento trabaja como se necesita antes de confiar otros elementos a él.

Para aplicar ingeniería hacia adelante en un diagrama de casos de uso

- Para cada caso de uso en el diagrama, identificar sus flujos de eventos.
- Dependiendo de qué tan profunda se elija la prueba, generar un guion prueba para cada flujo, usando las precondiciones de flujo como un estado inicial de prueba y las posibles condiciones como criterio de éxito.
- Generar una prueba para representar cada actor que interactúa con el caso de uso.
- Usar herramientas para correr las pruebas cada vez que se libera el elemento hacia el cual el diagrama de casos de uso aplica.

Ingeniería Inversa. Es el proceso de transformar código hacia un modelo a través de un mapeo desde un lenguaje específico de implementación. Automáticamente la ingeniería inversa en un diagrama de caso de uso está mucho más allá del esto del arte. Simplemente porque hay una pérdida de información cuando se cambia de una especificación de cómo un elemento se comporta hacia como es implementado. Sin embargo, se puede estudiar el sistema existente y entender el comportamiento a mano; esto será lo que se colocará en forma de diagrama de casos de uso.

Para aplicar ingeniería inversa se recomienda:

- Identificar cada actor que interactúa con el sistema.

- Para cada actor, considerar la manera en la cual el actor interactúa con el sistema, cambia el estado del sistema o su medio ambiente, o responde a algún evento.
- Trazar el flujo de eventos en el sistema ejecutable relativo a cada actor. Iniciar con flujos primarios y después considerar rutas alternativas.
- Agrupar flujo relacionado declarando su correspondiente caso de uso.
- Proporcionar estos actores y casos de uso en un diagrama de casos de uso y establecer sus relaciones.

Sugerencias. Cuando se crea un diagrama de casos de uso en UML, recuerda que cada diagrama de caso de uso es solo una presentación gráfica de la vista estática de un caso de uso de un sistema. No un simple diagrama de casos de uso necesita capturar cada cosa en la vista de casos de uso de un sistema. Colectivamente, todos los diagramas de casos de uso de un sistema representan la vista estática del sistema completo; individualmente representan solo un aspecto.

Para una buena estructura de diagramas de casos de uso se recomienda:

- La estructura es enfocada sobre un aspecto de comunicación de la vista de un caso de uso de un sistema estático.
- Contiene solamente casos de uso y actores que son esenciales para entender el aspecto.
- Provee detalles de consistencia con los niveles de abstracción; exponiendo solo adornos que son esenciales para el entendimiento.

Cuando se dibuja un diagrama de casos de uso.

- Utilizar nombres representativos.
- Etiquetar los elementos con un mínimo de líneas cruzadas.
- Usar notas y colores para dibujar partes destacadas del diagrama.
- No demostrar demasiados tipos de relaciones. En general, si se incluyen complicadas y extensas relaciones, tomar esos elementos en otro diagrama.

Modelo de Presentación (Interfaces). El modelo de interfaces describe la presentación de información entre los actores y el sistema. Se especifica en detalle cómo se verán las interfaces de usuario al ejecutar cada uno de los casos de uso. Si se trata de Interfaz Humano Computadora (“HCI - Human Computer Interface”) se puede usar esquemas de cómo vería el usuario las pantallas cuando se ejecuta cada caso de uso. También se puede generar una simulación más sofisticada usando un Sistema Manejador de Interfaces de Usuario (“UIMS - User Interface Management System”). Normalmente, un prototipo funcional de requisitos mostrando las interfaces de usuario es una estrategia importante. Esto ayuda al usuario a visualizar los casos de uso según serán mostrados por el sistema a ser construido. Tal enfoque elimina muchas posibilidades de malos entendimientos. Cuando se diseñan las interfaces de usuario, es esencial tener a los usuarios involucrados, siendo esencial que las interfaces reflejen la visión lógica del sistema. Esto es realmente uno de los principios fundamentales del diseño de interfaces humanas, donde debe existir

consistencia entre la imagen conceptual del usuario y el comportamiento real del sistema. Si las interfaces son protocolos de hardware, se puede referir a los diferentes estándares, como protocolos de comunicación. Estas descripciones de interfaces son por lo tanto partes esenciales de las descripciones de los casos de uso y las deben acompañar. En estas etapas iniciales del desarrollo el diseño de las pantallas no es tan importante como el manejo de información que se ofrece el cual debe corresponder a las necesidades de cada caso de uso.

Modelo de Información (Dominio). El modelo del dominio del problema define un modelo de clases común para todos los involucrados en el modelo de requisitos, analistas al igual que clientes. Este modelo de clases consiste de los objetos del dominio del problema, o sea objetos que tienen una correspondencia directa en el área de la aplicación. Como los usuarios y clientes deberían reconocer todos los conceptos, se puede desarrollar una terminología común al razonar sobre los casos de uso, y por lo tanto disminuyendo la probabilidad de malos entendimientos entre el analista y el usuario.

El propósito principal del dominio del problema en el modelo de requisitos de nuestra metodología es formar una base común de entendimiento del desarrollo y no para definir el sistema completo. Por lo tanto, se pueden aprovechar algunas técnicas heurísticas para la identificación de los objetos en el dominio del problema, logrando un glosario o diccionario de clases que sirve como común denominador a todos los componentes del sistema, incluyendo a las diversas personas involucradas a lo largo del desarrollo. A diferencia de los métodos anteriores, el modelo del dominio del problema no debe ser demasiado extenso, ya que varios grados de refinamiento serán hechos posteriormente y aunque es suficiente con describir el dominio del problema en término de objetos o clases, es posible refinar aún más mediante la inclusión de asociaciones, atributos, herencia y operaciones, siempre y cuando esto ayude a comprender mejor el problema y no se vuelva un esfuerzo demasiado grande durante esta etapa. Se debe tener cuidado con hacer demasiado trabajo temprano ya que esto puede incluso dificultar su modificación posterior durante el modelo de análisis. La experiencia muestra que muchos (si no todos) de los objetos del dominio podrán aparecer durante el modelo de análisis. Sin embargo, puede haber cambios durante el modelo de análisis, incluyendo la eliminación de clase identificadas durante esta etapa, o incluso la incorporación de clases adicionales.

Enterprise Architect. (EA). Con la finalidad de poder elaborar más fácilmente un modelo en UML, nos auxiliamos con una herramienta CASE que en este caso es el Enterprise Architect, recordando lo que se planteó en la primera práctica.

El EA nos ofrece una serie de ventajas que pueden resumirse en las siguientes:

- Está Basado en estándares abiertos como UML, BPMN y SysML.
- Apoya a las empresas con marcos de arquitectura como TOGAF y UPDM.
- Posee herramientas personalizadas integradas para analizar y visualizar la ejecución de software.
- Posee simulación avanzada, herramientas de prueba, uso de repositorios, control de versiones y más.
- Es capaz de gestionar los requisitos, modelos estratégicos, análisis y modelos de procesos.

- Permite desarrollar y visualizar el software.
- Pone a prueba y comprueba el correcto comportamiento.
- Permite simular procesos. Establecer y administrar los puntos de prueba, depuración e inspección en la ejecución de software.
- Es capaz de simular dinámicamente modelos de comportamiento.
- Permite confirmar el diseño del proceso, especificar disparadores, eventos, limitaciones y más.
- Permite especificar los sistemas estructurales complejos y de gran tamaño.
- Ofrece soluciones de depuración incrustados.
- Permite construir soluciones específicas de dominio personalizado.
- Permite construir sus propias herramientas de modelado de dominio específico.

Conceptos importantes:

Proyecto. Un proyecto en EA es un mecanismo para almacenar y administrar los componentes de uno o más modelos UML.

Un proyecto (archivo. EAP) en una base de datos MS Access o (en la edición Corporativa de EA) una estructura de archivos en un sistema de administración de base de datos tales como MySQL u Oracle. Un proyecto puede contener un solo modelo, o un número de modelos, cada uno de los cuales define un aspecto particular del sistema o proceso. Usted puede crear el modelo de una plantilla específicamente estructurada para soportar los aspectos que el modelo representa, tal como un proceso de negocio, un modelo de requisitos o despliegue.

Modelo. Un modelo contiene los diagramas, elementos, relaciones y meta-información asociada que define la estructura y función de un sistema o proceso. Estos componentes están organizados en una jerarquía de paquetes, que ayuda a agrupar y administrar los componentes relacionados.

Los paquetes de más alto nivel en un modelo son las Vistas, que representan particiones del modelo que usted mismo define. Se puede comenzar con una vista estándar como clase o componente, o crear cualquier partición que sea apropiada para su modelo.

Un proyecto en EA típico podría tener una estructura parecida a la siguiente:

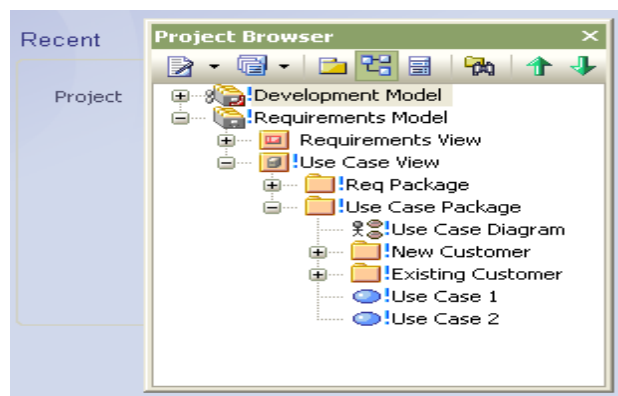


Figura 1 Estructura típica de un proyecto en EA.

El proyecto contiene dos modelos:

- Modelo de Desarrollo y
- Modelo de Requisitos.

Cada modelo contiene Vistas. El Modelo de Requisitos contiene:

- Vista de Requisitos y
- Vista de Casos de Uso

Cada Vista contiene paquetes. La Vista de Casos de Uso contiene:

- Paquete Requerimientos y
- Paquete Caso de Uso

Cada paquete contiene uno o más diagramas, uno o más paquetes, y varios elementos. El paquete Caso de Uso contiene:

- Diagrama Caso de Uso
- Paquete Usuario Nuevo
- Paquete Usuario Existente
- Elemento Caso de Uso 1
- Elemento Caso de Uso 2

Crear un Nuevo Proyecto. Los proyectos de EA, se pueden crear a través de la opción de menú que aparece al inicio: Crear un nuevo proyecto, la cual muestra la ventana: nuevo proyecto de EA, Seleccionar un directorio e ingrese un nombre de archivo para su proyecto, luego hacer clic en el botón Guardar. Una vez que el proyecto ha sido guardado, el Asistente del modelo aparecerá. Una selección de los modelos estará disponible. Verificar los modelos para incluir y hacer clic en el botón Aceptar.

Alternativamente nuevos proyectos se pueden crear desde la Página de inicio, seleccionar

Asistente del Modelo. El Asistente de Modelo está disponible cuando se crea un nuevo proyecto. Una vez que se creó un proyecto, puede acceder también al asistente de modelo desde la ventana: Explorador del proyecto. Haga clic con el botón derecho sobre un nodo raíz y seleccione la opción de menú Agregar modelo usando el asistente.

Campo/Botón	Descripción
Seleccionar desde	Hacer clic sobre la flecha del menú desplegable y seleccionar la categoría del modelo desde la cual desea seleccionar el tipo de modelo a crear.
Tecnología	Si posee Add-ins o Tecnologías MDG avanzadas este panel las lista. Seleccione la tecnología apropiada para listar las plantillas asociadas en el panel <i>Nombre</i> . Para listar las plantillas del modelo

	de EA estándar seleccione <predeterminado> .
Todo	Selecciona todos los modelos.
Ninguno	Elimine todos los modelos seleccionados.
Aceptar	Presionar este botón para crear el árbol de proyecto para su proyecto.
Cancela	Presionar este botón para crear un árbol de proyecto en blanco.
Ayuda	Muestra la ayuda.

Tabla 1. Opciones del asistente para la creación de un Proyecto

Administrar Vistas. Las vistas son los paquetes del nivel superior de EA. Esta terminología se utiliza simplemente para señalar que el paquete está en el nivel superior y se puede subdividir un proyecto en particiones tales como: procesos de negocio, modelo lógico o vista dinámica.

Hay 6 vistas principales:

- Vista de Casos de Uso: Diagrama de casos de uso, diagrama de análisis,
- Vista dinámica: Diagrama de actividades, diagrama de comunicaciones, diagrama de secuencias, diagrama de estados.
- Vista lógica: Modelo de clases, Ingeniería de código, Modelo de datos.
- Vista de componentes: Diagrama de componentes.
- Vista de despliegue - Diagrama de despliegue.
- Vista personalizada.

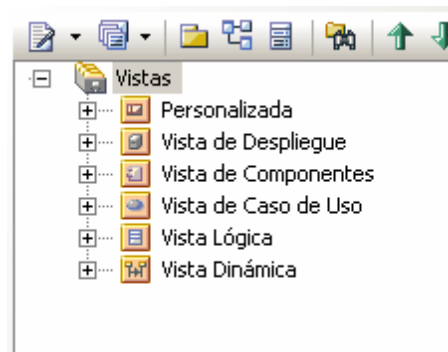


Figura 2. Vistas estándar de un proyecto en EA.

Crear una vista de usuario. Para crear una vista de usuario, proceda con los siguientes pasos:

1. Hacer clic con el botón derecho en el elemento raíz de la ventana del *Explorador del proyecto* (llamado *Vistas*). Se abre el menú contextual.
2. Seleccione la opción Nueva vista. Se abre la ventana *Crear una nueva vista*.
3. En el campo Nombre, ingrese el nombre de la vista.

4. En el panel Configurar estilo del ícono de vista, haga clic en el botón del icono de la vista que requiera.
5. Haga clic en el botón Aceptar.

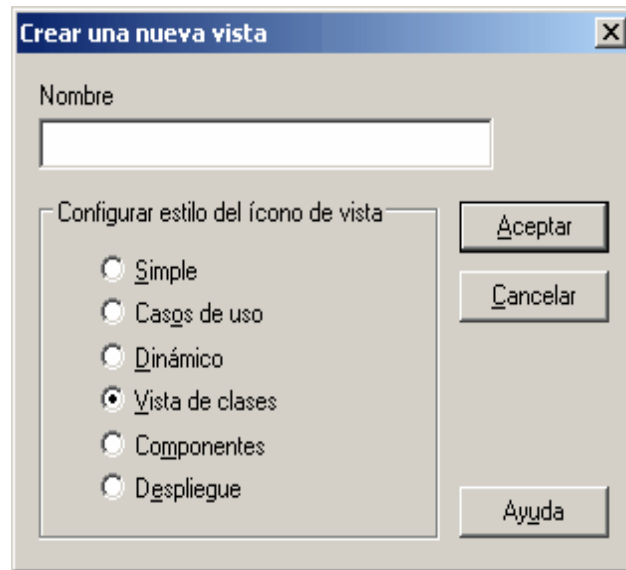


Figura 3. Creación de nuevas vistas en EA.

Diagramas UML. La figura debajo muestra la taxonomía de los 13 diagramas UML, como está definido por la especificación de UML 2.0 de Grupos de Desarrollo de Objetos. Como se puede ver, hay dos grupos mayoritarios de diagramas: diagramas Estructurales los cuales muestran una vista estática del modelo; y diagramas de Comportamiento los cuales muestran una vista dinámica del modelo. Como verás en teoría ambas representaciones se complementan y son muy útiles en toda arquitectura de un sistema.

Para más detalles acerca de la construcción de estos diagramas puede revisar estas opciones en tu programa de Enterprise.

- Diagramas Estructurales
- Diagrama de Clases
- Diagrama de Objetos
- Diagrama de Componentes
- Diagrama de Estructura Compuesta
- Diagrama de Despliegue
- Diagrama de Paquetes
- Diagrama de Comportamiento
- Diagrama de Secuencia
- Diagrama de Comunicaciones
- Diagrama de Descripción de la Interacción
- Diagrama de Tiempos
- Diagrama de Actividades
- Diagrama de Casos de Uso
- Diagrama de Máquina de Estados

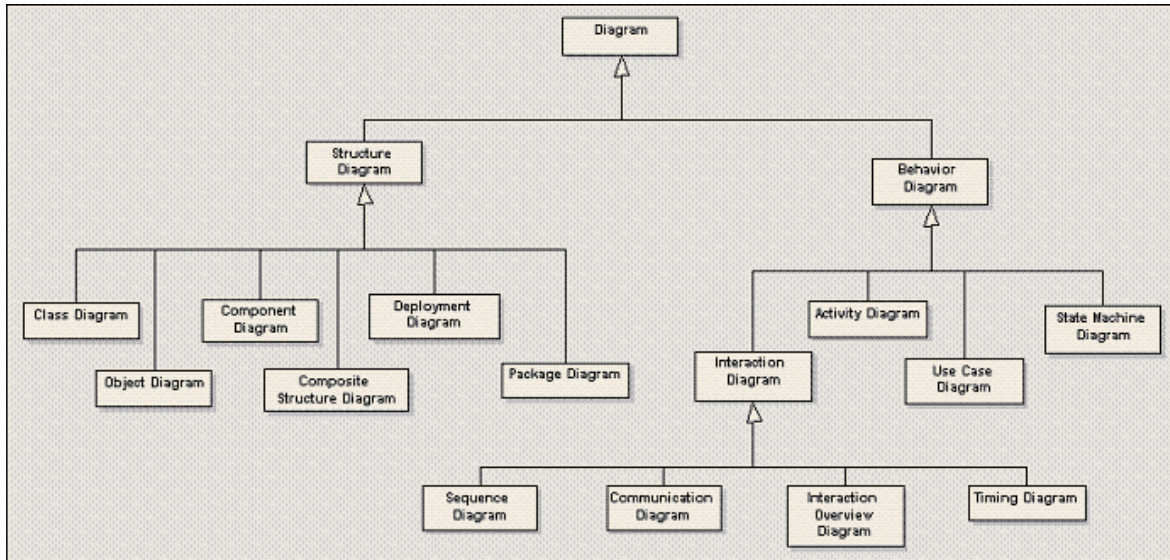


Figura 4 La jerarquía de los diagramas para EA.

Trabajando con Diagramas. Los diagramas se desarrollan en el espacio de trabajo principal en el que crea y conecta los elementos de modelado. Ellos se crean haciendo clic con el botón derecho sobre un paquete y seleccionando Nuevo Diagrama, o se cargan haciendo doble clic en los íconos de sus diagramas en el Explorador de Proyectos.

Diagramas Extendidos: Además de los 13 diagramas de UML, EA provee los siguientes diagramas extendidos:

- Diagramas de Análisis
- Diagramas Personalizados
- Diagramas de Requisitos
- Diagramas de Mantenimiento
- Diagrama de Interfaz de Usuario
- Diagramas de Base de Datos

Diagramas de Comportamiento. Los diagramas de comportamiento describen las características de comportamiento de un sistema o proceso de negocios. Los diagramas de comportamiento incluyen:

- Diagramas de Actividades
- Diagramas de Máquina de Estados
- Diagramas de Comunicaciones
- Diagramas de Descripción de la Interacción
- Diagramas de Secuencias
- Diagramas Tiempos
- Diagramas de Casos de Uso

Diagrama del Paquete. Los *Diagramas de Paquetes* se usan para reflejar la organización de los paquetes y sus elementos, y para proveer una visualización de sus correspondientes nombres de espacio.

Diagrama de ejemplo. El ejemplo siguiente muestra un diagrama de paquetes básico. El conector anidado entre *GeneradorAplicacion* y *Controlador* refleja lo que revela el contenido del paquete. El contenido del paquete se puede listar dando doble clic en el paquete para mostrar la ventana propiedades del diagrama, seleccionando la pestaña Elementos y seleccionando la Casilla de contenidos del paquete.

El conector «import» indica que los elementos en el paquete *Integrador*, que en este ejemplo es una única clase, *Integrador*, se importará en el paquete *Controlador*. El nombre de espacio del *Controlador* obtendrá acceso a la clase *Entero*; el nombre de espacio de *Entero* no se afecta. El conector «merge» indica que los elementos del paquete *Controlador* se importarán en *GeneradorAplicacion*, incluyendo los contenidos anidados e importados de *Controlador*.

Si un elemento ya existe en *GeneradorAplicacion*, tal como *Cargador* y *Tiempo*, las definiciones de estos elementos se expandirán por aquellas que se incluyen en el paquete *Controlador*. Todos los elementos agregados o actualizados por la mezcla son representados por una relación de generalización de vuelta a este paquete.

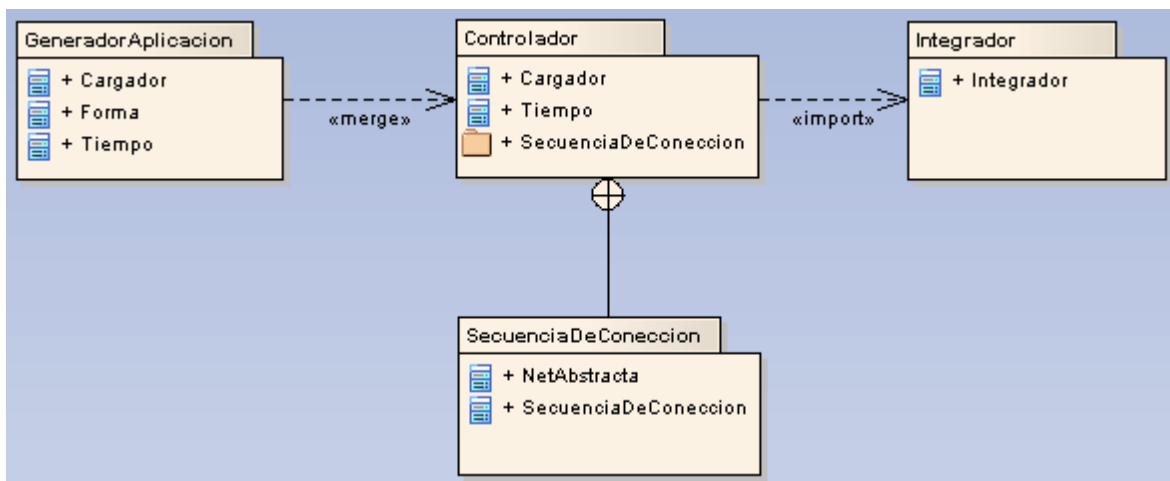


Figura 5. Diagrama básico de paquetes en EA.

Elementos y Conectores de la Caja de Herramientas. Seleccione los elementos y conectores del diagrama de Paquetes desde las páginas Clase de la caja de herramientas del EA que pueden ser las siguientes:

Elementos del diagrama de paquetes	Conectores del diagrama de paquetes
 Paquete	 Asociar
 Clase	 Generalización
 Interfaz	 Composición
 Tabla	 Agregación
 Enumeración	 Clase asociación
 Señal	 Ensamblar
 Asociación	 Realización
	 Anidamiento
	 Combinar paquete
	 Importar paquete

Tabla 2. Elementos y conectores para un diagrama de paquetes en EA:

Sysml y los Requerimientos. En la versión 2 de UML, mediante la extensión de Sysml, se adiciona el concepto de requerimiento para complementar los casos de uso mediante modelados gráficos que nos ayuden en la especificación de sistemas.

Dentro de EA se contemplan dos tipos de requerimientos a saber:

- Requerimiento externos - como una expectativa del sistema o proceso, lo que el sistema o proceso debe proveer, modelado como un *elemento*; por ejemplo, un requerimiento de negocio o un pedido de un propietario - Los requerimientos en este nivel tienen sus propias propiedades y se reportan por separado en los reportes RTF.

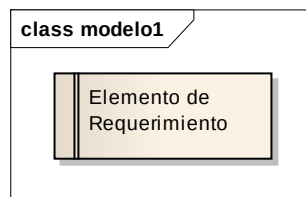


Figura 6. Símbolo de un elemento de requerimientos en EA.

- Requerimientos internos – una responsabilidad de un elemento existente, lo que el elemento debe hacer o ejecutar, definido como una propiedad del elemento.

Objeto : Objeto2

General | **Requerimientos** | Restricciones | Vinculos | Escenario | Archivos | Valores etiquetados

Requisito: Ingresar al sistema Tipo: [v]
Ultima actualización: 21/10/2014 [v]

Estado: [v] Dificultad: [v] Prioridad: [v] Estabilidad: [v]
Proposed Medium Medium Moderate

B I U A [v] [v] [v] [v] [v] [v]

Cada usuario podra ingresar al sistema solamente dando de alta su usuario y password

Mover a externo Nuevo Guardar Eliminar

Requisito	Tipo	Externo

Figura 7. Descripción de un Requerimiento interno en EA.

La administración de requerimientos en Enterprise Architect esta principalmente comprendida con elementos de requerimiento *externo* y los elementos que lo implementan y realizan.

Requerimientos en el modelo. Los elementos de requerimientos se pueden agrupar y organizar dentro de diagramas de requerimientos. Los elementos de requerimientos se conectan entre ellos por relaciones de agregación para formar una jerarquía:

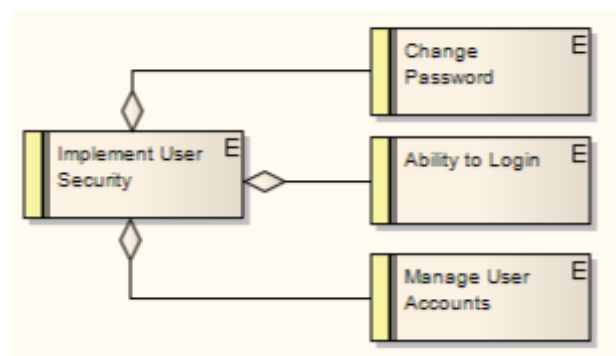


Figura 8. Modelo de Requerimientos en EA.

Es bastante usual desarrollar un paquete de cientos de elementos de requerimientos, ordenados individualmente y en jerarquías de complejidad variada. En el Explorador del proyecto puede usar la capacidad de Mostrar la numeración del nivel para resaltar el orden y arreglo de requerimientos rápida y fácilmente. La siguiente ilustración muestra un número de requerimientos en un paquete, donde la numeración del nivel hacen que el orden

y arreglo sea claro:

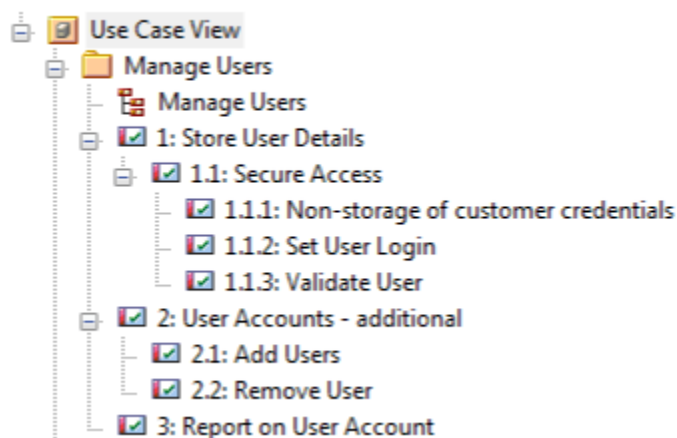


Figura 9. Organización del diagrama de requerimientos.

Si los elementos se agregan, mueven o eliminan desde el paquete, la numeración automáticamente se ajusta.

Casos de uso y Requerimientos. Los requerimientos se implementan (realizado) por los elementos del modelo como casos de uso, clases, interfaces y componentes. Hay muchas maneras para localizar el requerimiento para la característica o el servicio modelado por los elementos, o los elementos que desarrollan el requerimiento, más visible en los diagramas de Trazabilidad que describen los requerimientos y modelan los elementos conectados por las relaciones de Realización. El conector de Realización permite que los miembros del equipo del proyecto mantengan claro los objetivos del diseño y el desarrollo a la vez, y la ruta del desarrollo y el propósito.

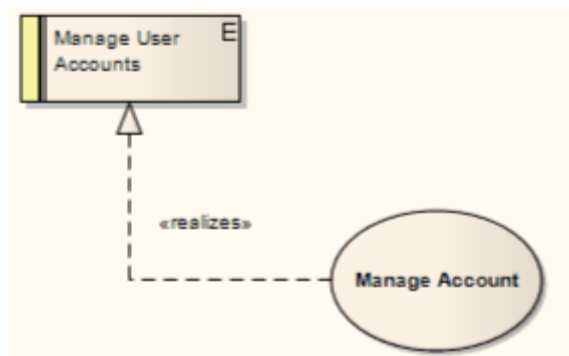


Figura 10. Relación entre casos de uso y requerimientos en EA.

La relación de la realización más usual es entre un requerimiento y un caso de uso. Un requerimiento se puede realizar por uno o más casos de uso, y un caso de uso puede realizar uno o más requerimientos.

Mientras un requerimiento define una condición que se debe alcanzar, el Caso de uso es la clave para definir y visualizar como se logra esa condición. Un diagrama de caso de uso describe el agrupamiento lógico de acciones, procesos y componentes para alcanzar un resultado requerido, y a través del uso de los elementos Actor también define el usuario y/o los roles del sistema que participan en el proceso.

Cada caso de uso (como un elemento compuesto) puede contener una combinación de diagramas hijos que se definen en más detalle cómo se puede implementar una actividad o capacidad particular - por ejemplo los diagramas incluyen diagramas de Secuencia, Comunicación, Actividad, Máquina de estado y Flujo de reglas de negocio. La implementación actual de cada caso de uso se realiza por los elementos de clase, componente e interfaz organizados en sus propios diagramas. Estas realizaciones también se pueden capturar y ver en los diagramas de trazabilidad, describiendo la ruta de desarrollo completo desde el requisito inicial a través de las pruebas y producción.

Diagrama de Casos de uso. Un diagrama de Casos de Uso captura las interacciones de los casos de uso y los actores, describe los requisitos funcionales del sistema, la forma en la que las entidades externas (actores) interactúan a través del límite del sistema y la respuesta del sistema.

Un caso de uso es un elemento de modelado UML que describe cómo un usuario del sistema propuesto interactuará con el sistema para llevar a cabo una unidad discreta de trabajo. Describe y significa una interacción simple a lo largo del tiempo que tiene un significado para el usuario final (persona, máquina u otro sistema), y se requiere que deje al sistema en un estado completo: se completó la interacción o se la volvió atrás hasta su estado inicial.

- Un caso de uso normalmente tiene requerimientos y restricciones que describen las características y reglas esenciales bajo las cuales opera.
- Un caso de uso puede tener asociado un diagrama de Secuencias ilustrando el comportamiento a lo largo del tiempo -quién hace qué y para quienes, cuándo.
- Un caso de uso típicamente tiene escenarios asociados a él que describen el flujo de trabajo a lo largo del tiempo que produce el resultado final. También está permitido alternar flujos de trabajo (para capturar excepciones, etc.)

Diagrama de ejemplo. El diagrama de ejemplo de abajo ilustra algunas características de los diagramas de Casos de Uso que concuerdan con lo descrito anteriormente respecto al tipo de asociaciones que pueden presentar entre los casos de uso y entre estos y los actores que interactúan con el sistema. Además es posible identificar el límite del sistema.

También se pueden observar las notas que incluyen información adicional al diagrama y algunas ligas que se pueden insertar en EA para integrar información como de tipo audiovisual que pueda servir de complemento al diagrama.

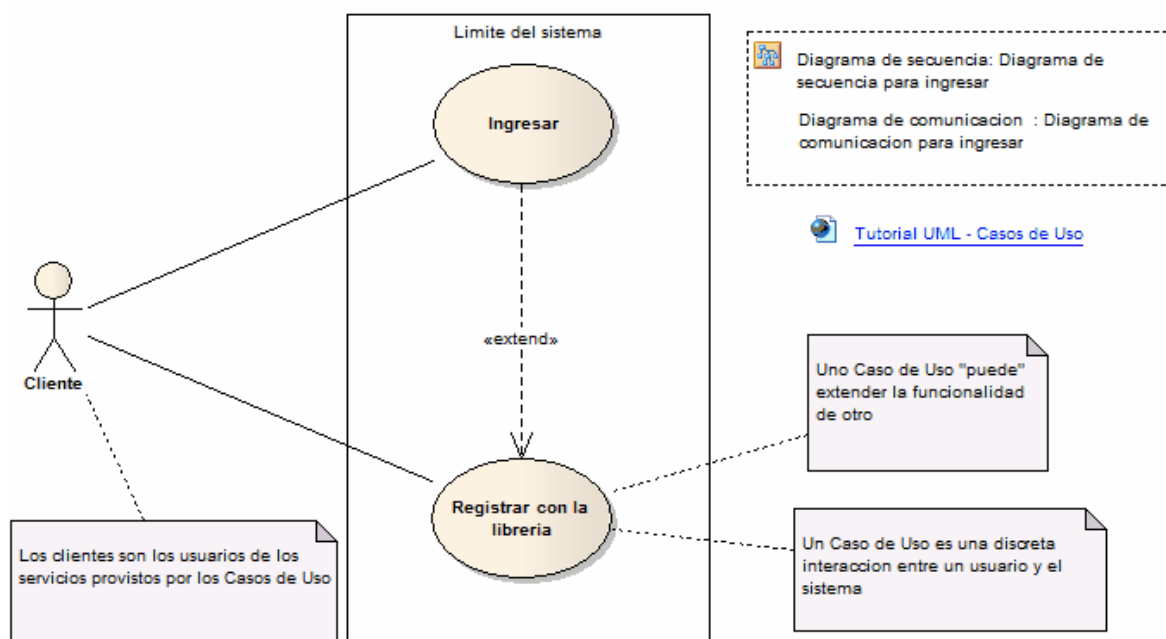


Figura 11. Componentes de un diagrama de Casos de Uso en EA.

Elementos de los Diagramas de Casos de Uso		Conectores de los Diagramas de Casos de Uso	
	Actor		Caso de Uso
	Casos de Uso		Asociar
	Colaboración		Generalización
	Límite		Inclusión
	Paquete		Extensión
			Realización
			Dependencia
			Trazado

Tabla 3. Elementos y conectores de los diagramas de Casos de Uso.

Diagrama de Clases El diagrama de Clases captura la estructura lógica del sistema. Las clases y entidades que constituyen el modelo. Es un modelo estático, describiendo lo que existe y qué atributos y comportamiento tiene, más que cómo se hace algo. Los diagramas de Clases son los más útiles para ilustrar las relaciones entre las clases e interfaces. Las generalizaciones, las agregaciones y las asociaciones son todas valiosas para reflejar la herencia, la composición o el uso y las conexiones respectivamente.

Diagrama de ejemplo. El diagrama de abajo ilustra las relaciones de agregación entre clases. La agregación con la punta de flecha en color claro indica que la clase Cuenta es usada por LibroDeDirecciones, pero que no está contenida necesariamente. La agregación con la punta de flecha en color oscuro indica la posesión o la contención de las clases destino (en el extremo del rombo) por las clases origen.

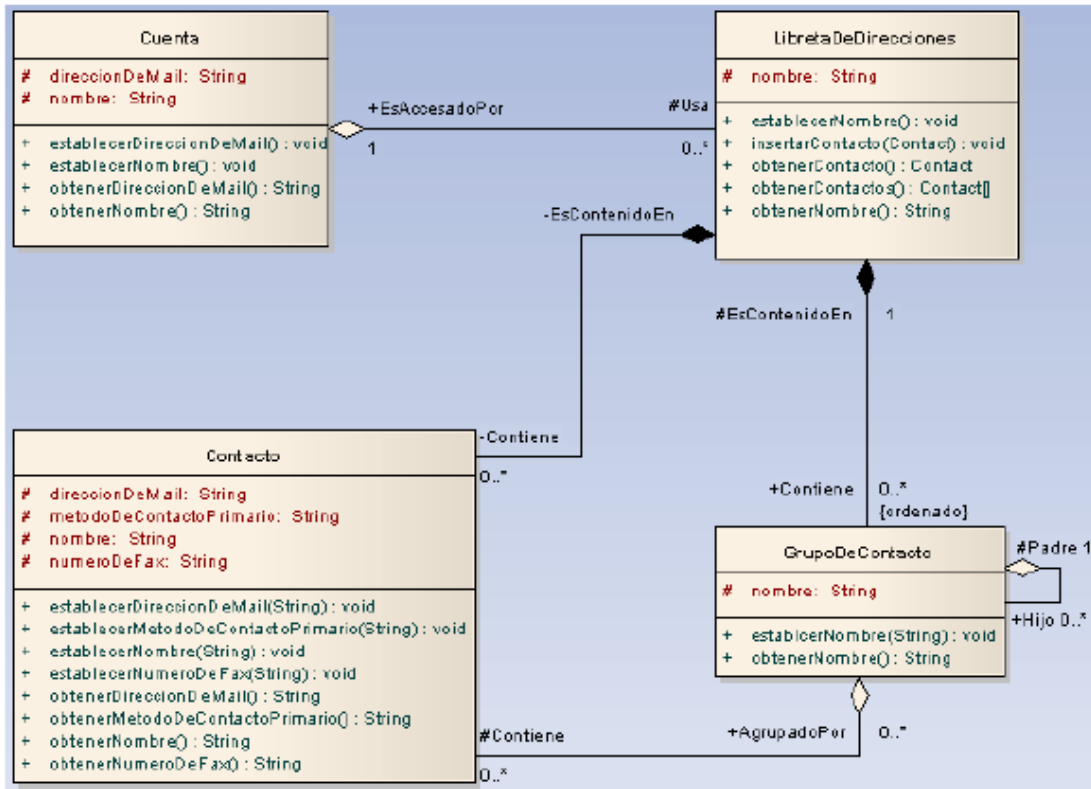


Figura 12. Ejemplo de un diagrama de clases en EA.

Elementos del diagrama de Clase	Conectores del diagrama de clase
 Paquete	 Asociar
 Clase	 Generalización
 Interfaz	 Composición
 Tabla	 Agregación
 Enumeración	 Clase asociación
 Señal	 Ensamblar
 Asociación	 Realización
	 Anidamiento
	 Combinar paquete
	 Importar paquete

Figura 4. Tabla de elementos y conectores de un diagrama de clases

Diagrama de Objetos. Un *diagrama de Objetos* está relacionado de cerca con un diagrama de Clases, con la diferencia de que éste describe las instancias de los objetos de clases en un punto en el tiempo. Esto podría parecer similar al diagrama de Estructura Compuesta, que modela el comportamiento en tiempo de ejecución; la diferencia es que el diagrama de Objetos ejemplifica al diagrama de Clases estático, mientras que los diagramas de Estructura Compuesta refleja las arquitecturas diferentes de sus contrapartes estáticas. Los diagramas de Objetos no presentan arquitecturas que varíen de sus correspondientes diagramas de Clases, pero reflejan la multiplicidad y los roles a los que las clases instanciadas podrían servir. Ellos son muy útiles en la comprensión de diagramas de Clases complejos, al crear diferentes casos en los que se aplican las relaciones y las clases. Un diagrama de Objetos puede ser también un tipo de diagrama de Comunicaciones, el cual modela también las conexiones entre pares de objetos y además de las secuencias de eventos a lo largo de cada camino.

Diagrama de ejemplo. El siguiente ejemplo primero muestra un diagrama de Clases simple, con dos elementos clase conectados.

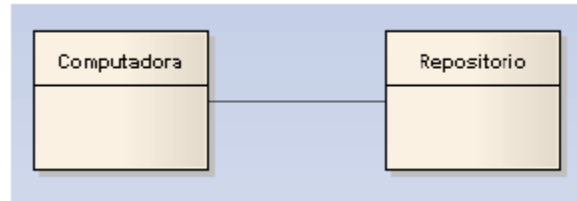


Figura 13. Ejemplo de un diagrama de clases simple en WA.

Las clases de arriba se instancian abajo como objetos en un diagrama de Objetos. Hay dos instancias de computadora en este modelo, lo que puede probar su utilidad por considerar las relaciones y las interacciones que las clases juegan en la práctica, como objetos.

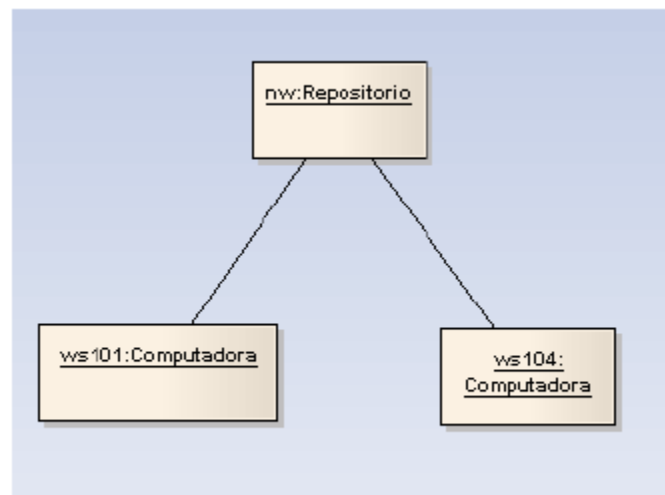


Figura 14. Ejemplo de un modelo de objetos en EA.

Object Diagram Elements	Object Diagram Connectors
 Actor	 Flujo de información
 Objeto	 Asociar
 Colaboración	 Dependencia
 Ítem de información	
 Limite	
 Control	
 Entidad	

Tabla 5. Elementos de la Caja de Herramientas y Conectores

Diagrama de la Interfaz del usuario. Los Diagramas de la Interfaz del Usuario usados para imitar visualmente la interfaz del usuario del sistema usando formas, controles y etiquetas.

El siguiente diagrama de la Interfaz del Usuario de ejemplo, las formas, controles y etiquetas se organizan en el diagrama para describir su apariencia. Los elementos UI también se pueden rastrear a otros elementos del modelo vinculando el UI con la implementación de base.

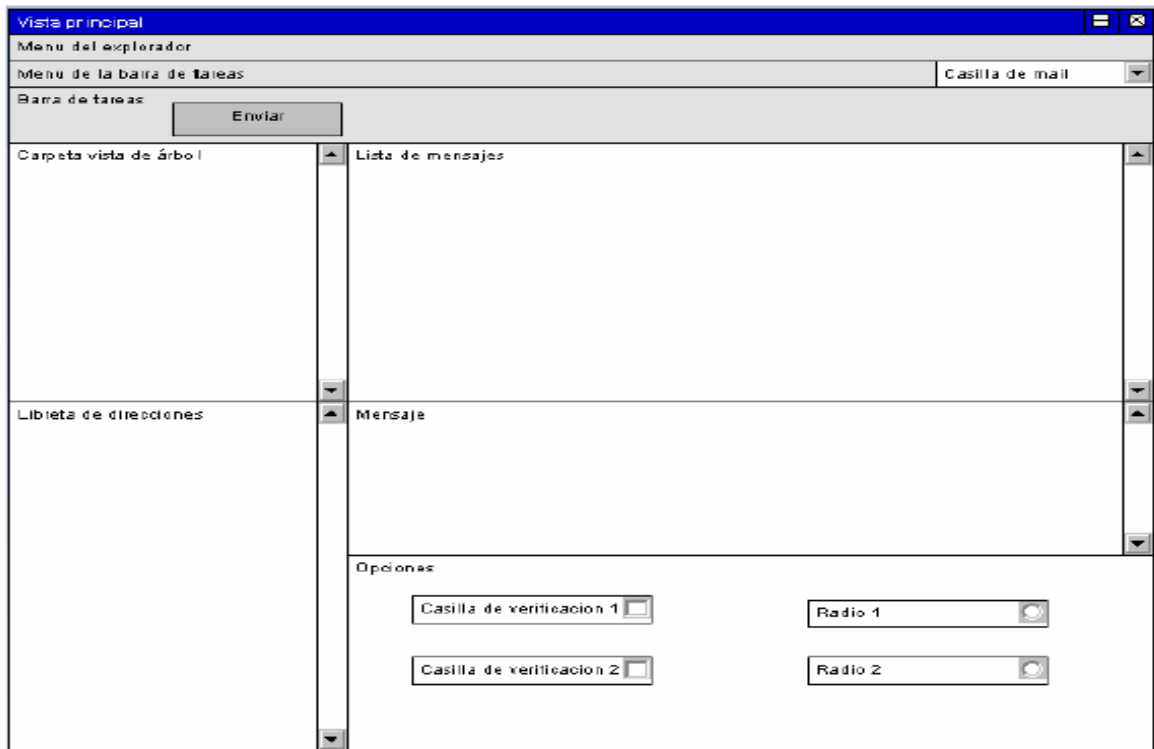


Figura 15. Ejemplo de diagrama de interface de usuario en EA.

Elementos del diagrama de requisitos	Conectores del diagrama de requisitos
 Paquete	 Asociar
 Pantalla	 Agregación
 Control UI	 Generalización
 Objeto	 Realización

Tabla 6. Conectores y Elementos de la Caja de Herramientas

3.4 Especificar la correlación con el o los temas y subtemas del programa. Es muy importante observar que de acuerdo al programa el alumno que cursa la unidad 2 del curso deberá saber elaborar un modelo de requisitos e implementarlo con ayuda de una herramienta CASE aplicando para ello una metodología de desarrollo (proceso unificado) como se ve en esta práctica.

3.5 Material y equipo necesario. A partir de esta práctica los diagramas de modelado se harán en EA ver. 11.0 (o la que el maestro decida), el cual deberá estar instalado en cada computadora de los alumnos.

3.6 Metodología. A continuación mencionaremos los pasos básicos para elaborar un modelo de requisitos utilizando EA.

Modelo de Casos de Uso.

- a) Si no cuenta con la herramienta de EA, le recomendamos bajar una prueba por 30 días de la dirección: <http://www.sparxsystems.com> en la opción de descargas encuentra la versión 11.1
- b) Una vez instalado el EA en su equipo abrir el programa y crear un proyecto nuevo con el nombre de su sistema, el cual lo puede guardar en la carpeta de mis documentos.
- c) Seleccione desde: Proceso Unificado las casillas siguientes: Casos de Uso, Modelo de Dominio, Modelo de Análisis, Modelo de Diseño y Modelo de Implementación.
- d) En el explorador del proyecto podrá observar la estructura que se muestra en la gráfica de la página siguiente. Si revisa los contenidos podrá entrar al modelo de casos de uso en donde se despliega un diagrama de ejemplo organizado en dos paquetes y unas notas respectivas para cada elemento del diagrama, de la misma manera se puede observar unas ligas en donde será posible leer sobre el uso y elaboración de diagramas de casos de uso, sobre los actores y finalmente ver algunos ejemplos que pueden ser ilustrativos. La recomendación es leer detenidamente esta información y borrar los diagramas de esta vista, mediante la selección correspondiente y el tachecito rojo que está en el extremo izquierdo de la barra de herramientas.
- e) Siguiendo esta secuencia le recomiendo desarrollar su modelo de casos de uso de su sistema organizado por un paquete de actores y uno o varios paquetes de casos de uso por cada subsistema, o sea que tendrá un número de paquetes y diagramas de casos de uso equivalentes al número de subsistemas con el que cuenta su sistema. En cada caso de uso quedarán claros los siguientes elementos: límites del diagrama, asociaciones entre los casos de uso, actores que participan en cada caso de uso.

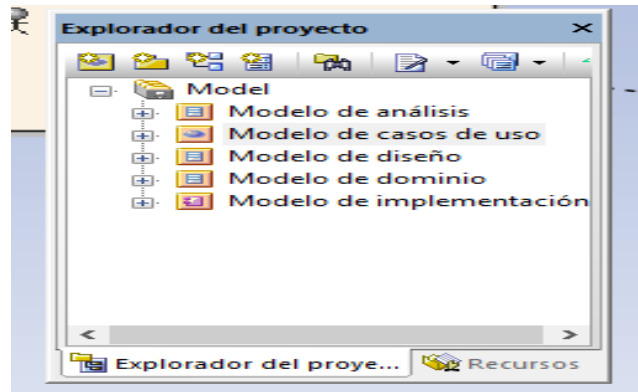


Figura 16. Ejemplo de ordenamiento de vistas.

- f) Una vez completado su diagrama dar doble clic en cada caso de uso y se despliega la ventana siguiente:

Figura 17. Caja de dialogo para la especificación de un caso de uso.

Para esta caja completar las opciones identificadas en la pestaña: General, Requerimientos, Restricciones y Escenarios por cada caso de uso.

Modelo de interfaces.

- a) Diseñar la pantalla(s) por cada caso de uso considerando los escenarios de cada uno de ellos y que se describieron en los mismos. Para esto se puede auxiliar de un IDE adecuado (Eclipse o NetBeans), o bien mediante las herramientas de

interface de usuarios de EA que describimos en la introducción, para ello se recomienda crear un paquete nuevo que se llame interfaces dentro del paquete de implementación. Se hará una vista por cada caso de uso con el nombre del mismo de acuerdo a los descritos en el punto anterior.

Modelo de Dominio

- a) Abrir el proyecto creado en el inicio de esta práctica
- b) Abra el diagrama que se encuentra dentro del paquete de modelo de dominio.
- c) Seleccionar y borrar todos los elementos del diagrama.
- d) Para definir el modelo del dominio recordar lo siguiente: El modelo de dominio es una vista de todos los objetos que conforman un área de interés, y sus relaciones. Se usa para capturar los objetos significativos dentro de un sistema, organización o cualquier dominio de destino.
- e) Igualmente generar un diagrama de clases en un diagrama adicional que sea coincidente con el diagrama anterior, recordando la diferencia entre ambos diagramas.
- f) Completar los diagramas con las asociaciones y multiplicidades entre clases.
- g) En cada clase dar doble clic, a fin de abrir la ventana que lleva el nombre de la clase y completar
- h) Se hace un diccionario de clases.
- i) Para la elaboración del modelo clases en la herramienta EA. Recomendamos ver el video de la bibliografía (6).

Diccionario de conceptos

Del inciso anterior generar en Word un archivo en el que describe cada uno de los elementos que conforman el diagrama anterior, citando el nombre del objeto/clase y en una descripción breve el significado de este concepto.

3.7 Sugerencias Didácticas. Para la elaboración del diagrama de casos de uso se recomienda que el alumno identifique primeramente el (los) actores principales que van a interactuar con el sistema y por cada actor tomar en cuenta la respuesta a las siguientes preguntas:

- ¿Cuáles son las tareas principales de cada actor?
- ¿Tendrá el actor que consultar y modificar información del sistema?
- ¿Tendrá el actor que informar al sistema sobre cambios externos?

¿Desea el actor ser informado sobre cambios inesperados?

En base a esto generar el escenario correspondiente mediante el establecimiento del flujo de eventos o acciones que elaborará llevará a cabo el actor y las respuestas propias del sistema. Es importante que cada caso de uso defina los requerimientos internos que permitirá su realización y que se contemplen todos los requerimientos tanto funcionales como no funcionales identificados en la práctica anterior.

Para la elaboración de interfaces se recomienda que el alumno se pueda apoyar de una herramienta IDE como puede ser NetBeans o Eclipse, para diseñar las diferentes pantallas por cada caso de uso e írselas mostrando al usuario a fin de que concuerde con la realidad.

Para la elaboración del Modelo del dominio se recomienda elaborar primero un diagrama de objetos tomando en cuenta los conceptos descritos en la descripción textual del problema, para después elaborar un diagrama de clases mediante un proceso de abstracción más general. En esta actividad se recomienda seguir el método heurístico descrito en la bibliografía 2 página 235.

3.8 Reporte de alumnos (Resultados). El alumno deberá de entregar al término de esta unidad un reporte completo denominado modelo de análisis que incluya lo siguiente:

Archivo con terminación EAP desarrollado en EA que contenga los siguientes paquetes:

- Modelo de Análisis (práctica 5).
- Modelo de Casos de Uso: Completo dividido por subsistemas con límites y relaciones entre ellos, por cada caso de uso completar el escenario correspondiente y sus requerimientos internos.
- Modelo de Diseño Paquete con diagramas de interfaces por cada caso de uso
- Modelo de dominio: Con dos sub-paquetes uno denominado objetos de dominio y otro modelo de clases con los diagramas correspondientes, de preferencia divididos por subsistemas.
- Modelo de Implementación (pendiente).
- Archivo en Word con el diccionario de datos.

3.9 Bibliografía.

1. Rational Unified Process. <http://www-306.ibm.com/software/awdtools/rup/>
2. Alfredo Weitzenfeld. (2005). Ingeniería de Software Orientada a Objetos con UML, Java e Internet. Thompson
3. Stephen R. Schach. (2005). Análisis y diseño orientado a objetos con UML y el proceso unificado. Mc.Graw Hill.
4. Consulta en youtube: <http://youtu.be/OrkwtIsF46E>

5. Consulta en youtube: <http://youtube/m2aFG7nQ0dU>
6. Consulta en youtube: <http://youtube.com/watch?v=jaqkOxW2Z78>

Práctica No.4

4.1 Nombre: Elaboración de un modelo de Análisis para un Sistema.

4.2 Objetivo: Que el alumno sea capaz de desarrollar un modelo de análisis para un sistema real tomando como base la arquitectura MVC y el uso de estereotipos de clase.

4.3 Introducción. El objetivo del modelo de análisis es comprender y generar una arquitectura de objetos para el sistema en base a lo especificado en el modelo de requisitos.

Durante esta etapa no se considera el ambiente de implementación, lo cual incluye al tipo de lenguaje de programación, el manejador de base de datos utilizado, la distribución del sistema o la configuración del hardware, etc.

En otras palabras el análisis pretende modelar el sistema bajo condiciones ideales, garantizando que la arquitectura de software resultante sea suficientemente robusta y extensible para servir de base a la estructura lógica de la aplicación pero sin consideraciones relativas al entorno de implementación que es posible que cambien incluso radicalmente.

El modelo de análisis genera una representación conceptual del sistema, consistiendo de clases de objetos. En el caso de los sistemas de información, uno de los tipos de arquitecturas más importantes es la arquitectura *MVC –Modelo, Vista, Control (Model, View, Control)*. Cada dimensión se implementa con la ayuda de clases con diferente estereotipo de acuerdo a la dimensión a la que pertenece.

Clases con Estereotipos. El tipo de funcionalidad o “la razón de ser” de un objeto dentro de una arquitectura se le conoce como su *estereotipo*. Para los sistemas de información la arquitectura del sistema según nuestro modelo de análisis se basa en tres estereotipos básicos de objetos:

El estereotipo *entidad* (“entity” en inglés) para objetos que guarden información sobre el estado interno del sistema, a corto y largo plazo, correspondiente al dominio del problema. Todo comportamiento naturalmente acoplado con esta información también se incluye en los objeto entidad. Un ejemplo de un objeto entidad es un registro de usuario con sus datos y comportamiento asociados.

El estereotipo *interface o borde* (“boundary” en inglés) para objetos que implementen la presentación o vista correspondiente a las bordes del sistema hacia el mundo externo, para todo tipo de actores, no sólo usuarios humanos. Un ejemplo de un objeto borde es la funcionalidad de interface de usuario para insertar o modificar información sobre el registro de usuario.

El estereotipo *control* (“control” en inglés) para objetos que implementen el comportamiento o control especificando cuando y como el sistema cambia de estado, de acuerdo a los casos de uso. Los objetos control modelan funcionalidad que no se liga

naturalmente con ningún otro tipo de objeto, como el comportamiento que opera en varios objetos entidad a la vez, por ejemplo, hacer alguna computación y luego devolver el resultado a un objeto borde. Un ejemplo típico de objeto control es analizar el uso del sistema por parte de algún usuario registrado y presentar tal información posteriormente. Este comportamiento no le pertenece a ningún objeto entidad u objeto borde específico.

Diagramas de Secuencias. Una vez identificadas las clases anteriores, proseguimos describiendo los casos de uso del modelo de requisitos según la lógica que deberán presentar estas clases para lograr la funcionalidad descrita en los diversos casos de uso.

Este es un paso muy importante ya que en base a la lógica propuesta para esta descripción, definiremos la arquitectura de análisis tanto estructural como funcional.

Dada la complejidad y la importancia de estas descripciones, es importante probar las secuencias funcionales de los casos de uso y revisar qué tan bien nuestra lógica y arquitectura de clase resuelve la funcionalidad establecida. Para eso introducimos el concepto de *diagramas de secuencias, interacción o eventos*, así como el diagrama de comunicación los cuales describen como los diferentes casos de uso son implementados mediante los objetos de nuestra arquitectura recién generados.

Diccionario de Clases. Como última etapa del modelo de análisis, se actualiza el diccionario de datos originalmente descrito para el dominio del problema para incluir todas las clases identificadas durante el modelo de análisis.

Aunque no es obligatorio, aprovechamos para separar estas clases en diferentes módulos para lograr una mejor organización y correspondencia entre clases y casos de uso. Aquellas clases que participan en varios casos de uso se pueden asignar a módulos adicionales

4.4 Correlación con el o los temas y subtemas del programa de estudio vigente. En la unidad 3 del programa de la materia, se estipula que el alumno comprenda los conceptos de arquitectura de clases, clases según estereotipos y diagramas de secuencias y mediante estos elementos pueda elaborar un modelo de análisis siguiendo la metodología utilizada hasta ahora: RUP lo que tiene una relación directa con lo que se revisa en la presente práctica.

4.5 Material y equipo necesario. En esta práctica el alumno requerirá utilizar la herramienta CASE EA en su versión libre de prueba o una versión licenciada propiedad de la escuela como se ha venido utilizando en las dos prácticas anteriores.

4.6 Metodología. Partiendo de los modelos de casos de uso elaborados en la práctica 3 se deberá de aplicar la siguiente metodología para aprovechar lo hecho a la fecha.

Análisis de cada caso de uso:

- a) Tomando cada uno de los casos de uso, por cada actor se definirá un objeto tipo borde para poderse comunicar con el sistema en muchos casos un actor puede necesitar de varios objetos borde. Al hacer esto hay que tener presente que

existen dos tipos diferentes de bordes a modelar; bordes a otros sistemas y bordes a los usuarios humanos. Para identificar qué parte del flujo de un caso de uso debe asignarse a los objetos borde, se debe analizar las interacciones entre los actores y los casos de uso. Esto significa buscar aspectos con una o más de las siguientes características:

- Presentación de información al actor que requiera información de regreso.
- Funcionalidad que cambie si cambia el comportamiento del actor.
- Flujo de acción que dependa de un tipo de borde particular.

- b) Tomando como base el modelo de dominio elaborado en la práctica anterior se tratará de identificar los objetos entidad por cada caso de uso, tomando en cuenta que estos se utilizan para modelar la información que el sistema debe manejar a corto y largo plazo. La información a corto plazo existe por lo general durante la ejecución del caso de uso, mientras que la información a largo plazo sobrevive a los casos de uso, por lo cual es necesario guardar esta información en alguna base de datos. En este punto se debe decidir si modelar la información como un objeto entidad o como un atributo, lo que depende de cómo se usará la información, o sea que si ésta se maneja de forma separada, debe modelarse como un objeto entidad, mientras que la información que esta acoplada fuertemente a alguna otra información y nunca se usa por si misma debe modelarse como un atributo de un objeto entidad. Todo depende de cómo los casos de uso manejen la información. Cierta información puede modelarse como objeto entidad en un sistema, mientras que en otro sistema puede ser un atributo. También es importante identificar qué operaciones y cuales atributos serán incluidos dentro de los objetos entidad.
- c) Los objetos control se identifican directamente de los casos de uso. Como primera aproximación, se asigna un objeto control a cada caso de uso, concreto y abstracto. Dado que se asigna inicialmente el comportamiento a los objetos borde y entidad para cada caso de uso, el comportamiento restante que no es posible asignarse en forma natural a estos objetos se asigna a los objetos control. Cuando tal modelo se ha desarrollado, puede darse la situación donde no queda comportamiento restante para modelar en el caso de uso. En tal caso no se necesita un objeto control. Otra situación es si el comportamiento que queda, después de distribuir el comportamiento relevante entre objetos borde y entidad, es demasiado complicado, la funcionalidad puede ser dividida en varios objetos control. Por otro lado, si un caso de uso se acopla a varios actores esto puede indicar que existen variados comportamientos en relación a los diferentes actores y por lo tanto deben asignarse varios objetos control. La meta debe ser ligar solo un actor con cada objeto control ya que los cambios en los sistemas a menudo son originados por los actores y de tal manera se logra modularizar los posibles cambios. La estrategia de asignación de control se debe decidir según cada aplicación. En la mayoría de los casos, sin embargo, se promueve la separación del control de un caso de uso en un objeto control que delega funcionalidad de manejo más local a los otros dos tipos de objetos.

- d) Con los datos anteriormente obtenidos se recomienda el llenado de la matriz de siguiente:

Caso de Uso	Clases Borde	Clases Entidad	Clases Control
-------------	--------------	----------------	----------------

Tabla 1. Matriz con especificación de clases por estereotipo por caso de uso

- e) Por cada caso de uso, genere los diagramas de secuencia y comunicación correspondiente utilizando EA. Nótese que estos dos son diagramas de comportamiento en donde se consideran los actores y el intercambio de información entre los objetos que participan en este considerando sus estereotipos correspondientes. Para el desarrollo de este diagrama se recomienda revisar el video 7 de la bibliografía.
- f) Finalmente actualizar o ampliar el glosario de términos elaborado en la práctica anterior, en donde se integre el significado de todas las clases con estereotipo que aparecen en el inciso d.

4.7 Sugerencias Didácticas. Se recomienda que una vez que por equipo los alumnos ya tengan claramente definidos los casos de uso del sistema, se continúe de la misma manera con el análisis de cada caso de uso definiendo una tabla para cada caso de uso, o sea que deberán de desarrollarse un número de tablas equivalente al total de casos de uso que contenga el sistema. Este trabajo se puede dividir por alumno de manera que cada uno de los alumnos pueda analizar un grupo de casos de uso y al final discutirlos por equipo (inciso d).

Para completar el inciso e se recomienda que cada alumno genere el diagrama de secuencia, el de comunicación o ambos según se decida en común acuerdo con el profesor. En este modelado es importante que el profesor haga algunas revisiones de los diagramas por equipo para verificar la correcta aplicación del UML.

4.8 Reporte de alumnos (Resultados).

Por equipos los alumnos entregarán un reporte con 3 componentes principales.

- Un archivo en Word con la matriz de clases por estereotipo por cada caso de uso determinado para el sistema en cuestión, se deberán de identificar los 3 estereotipos como mínimo por cada caso de uso: borde, entidad y control.
- El archivo .eap del proyecto elaborado en la práctica 3 en donde se complementa el modelo de casos de uso con los diagramas de secuencia y comunicación por cada caso de uso.
- El glosario de términos de la práctica 3 pero ampliado para el análisis y contemplando todas las clases identificadas en el primer inciso de esta sección.

4.9 Bibliografía:

1. Alfredo Weitzenfeld. 2005. Ingeniería de Software Orientada a Objetos con UML, Java e Internet. Thompson
2. Stephen R. Schach. 2005. Análisis y diseño orientado a objetos con UML y el proceso unificado. Mc.Graw Hill.
3. Spark Systems. Guía de Usuario de Enterprise Architect 7.1. 2009 Solus S.A.
4. Consulta en youtube <http://youtu.be/FzPET2TCpDQ>
5. Consulta en youtube http://youtu.be/sYUJq_JLkR0
6. Consulta en youtube <http://youtu.be/fbsG6Vwa4Tw>

Práctica No.5

5.1 Nombre: Generación de un modelo de análisis con la ayuda de una herramienta CASE

5.2 Objetivo: Que el alumno pueda desarrollar un modelo de análisis usando para lo mismo una herramienta CASE y cumpliendo con los entregables definidos dentro de la metodología RUP.

5.3 Introducción. Un modelo de análisis estaría incompleto sin la perspectiva del comportamiento del sistema que complemente los diagramas de casos de uso del modelo de requisitos elaborado en la práctica 3; esto se logra mediante los diagramas de secuencias o comunicaciones cuyos elementos principales consisten de los objetos con estereotipo identificados en la práctica 4.

Diagrama de Secuencia. Un *diagrama de Secuencia* es una representación estructurada de comportamiento como una serie de pasos secuenciales a lo largo del tiempo. Se usa para representar el flujo de trabajo, el paso de mensajes y cómo los elementos (objetos) en general cooperan a lo largo del tiempo para lograr un resultado.

Cada elemento de la secuencia está ordenado en una forma horizontal, con paso de mensajes hacia atrás y hacia adelante entre los elementos.

Un elemento actor se puede utilizar para representar al usuario iniciando el flujo de eventos.

Los elementos estereotipados, tales como límite, control y entidad, se puede utilizar para ilustrar pantallas, controladores e ítems de bases de datos, respectivamente.

Cada elemento tiene una línea de trazos llamada línea de vida, en donde este elemento existe y potencialmente toma parte en las interacciones.

Diagrama de Ejemplo

El siguiente es un diagrama de Secuencias de ejemplo mostrando varios elementos diferentes:

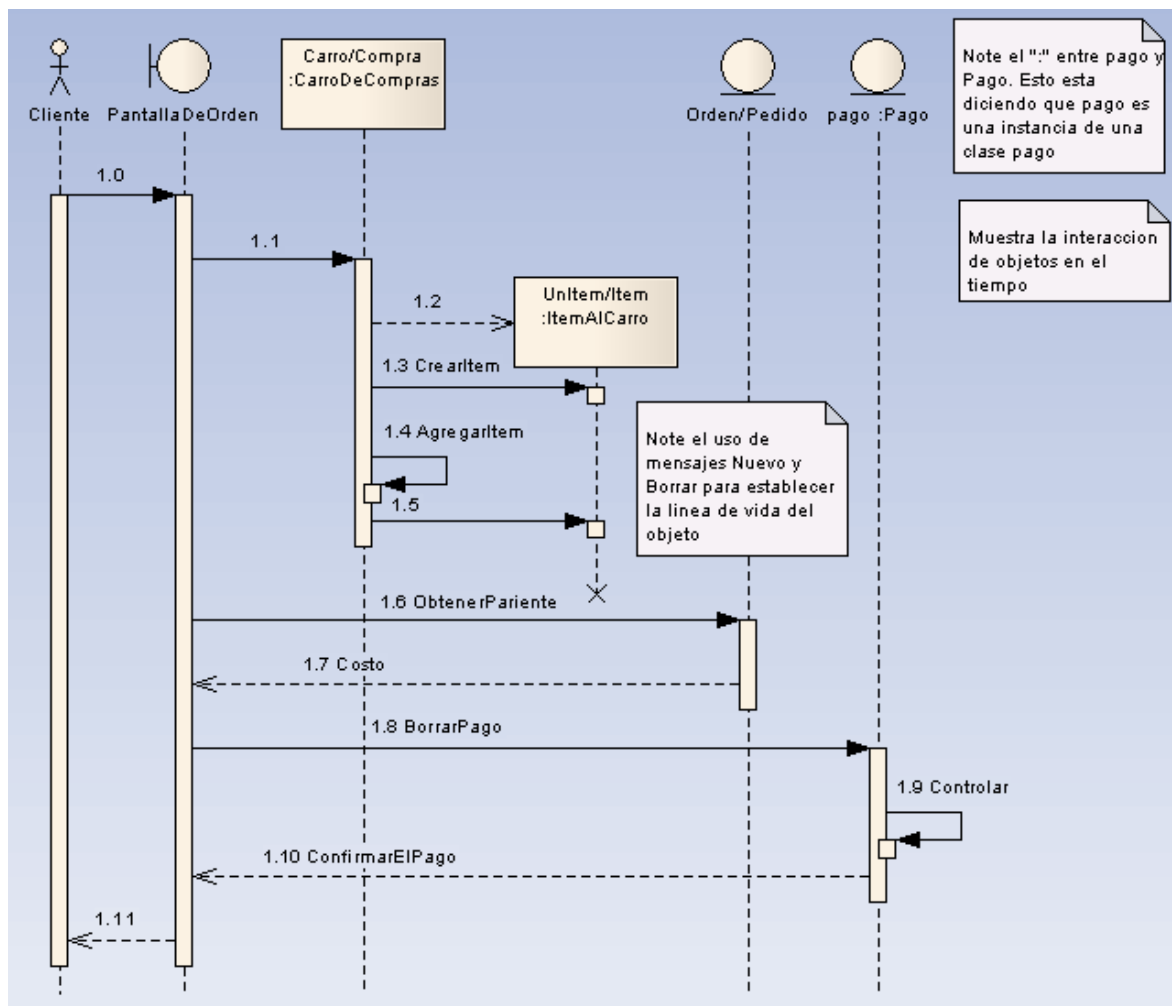


Figura 1. Ejemplo de un diagrama de secuencia en EA.

Elementos del Diagrama de Secuencias		Conectores del Diagrama de Secuencias	
	Actor		Mensaje
	Línea de Vida		Auto Mensaje
	Límite		Recursión
	Control		Llamada
	Entidad		
	Fragmento(s)		
	Punto Final		
	Puerta del Diagrama		
	Estado/Continuación		

Tabla 1. Elementos y Conectores de la Caja de Herramientas para diagrama de secuencia.

Un diagrama de secuencias puede existir en una forma genérica (describe todos los escenarios posibles) y en una forma de instancia (describe un escenario actual). Los diagramas de secuencias y los diagramas de comunicación expresan información similar, pero la muestran de diferentes formas.

El ejemplo de abajo muestra algunos posibles elementos de los diagramas de secuencias y su representación estereotipada.

- **Actor** - Una instancia de un actor en tiempo de ejecución.
- **Caso de Uso** - Una instancia de un caso de uso en tiempo de ejecución.
- **Objeto** - Un elemento estándar -no asignado a un tipo-.
- **Límite** - Representa una pantalla de interfaz de usuario o un dispositivo de entrada/salida
- **Entidad** - Un elemento persistente -normalmente se implementa como una tabla de una base de datos o un elemento.
- **Controlador** - El elemento activo que controla qué trabajo se realizó, cuándo y cómo.

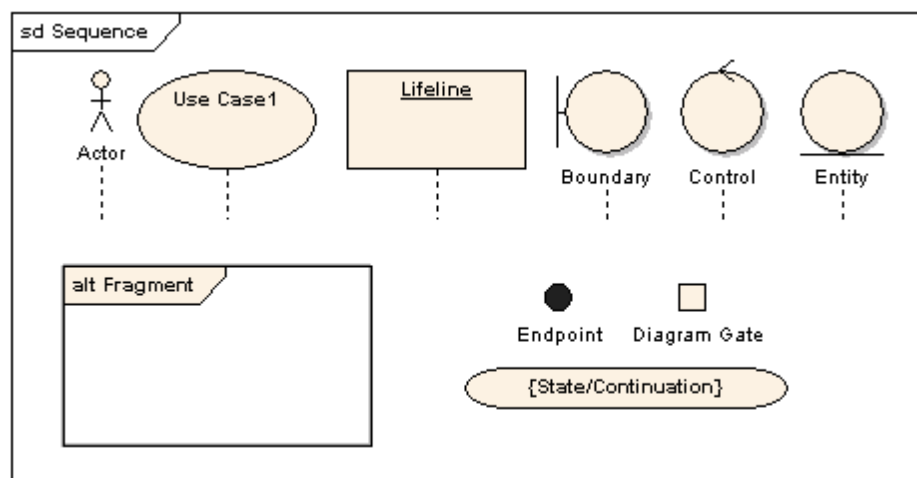


Figura 2. Elementos que conforman un diagrama de secuencias.

Un diagrama de Secuencia modela una vista dinámica de las interacciones entre los elementos del modelo en tiempo de ejecución.

Los diagramas de Secuencias se usan continuamente como modelos explicativos para los escenarios de los casos de uso. Mediante la creación de un diagrama de secuencias con un actor y los elementos involucrados en el caso de uso, puede modelar la secuencia de pasos que toman el usuario y el sistema para completar las tareas requeridas. Un elemento en un diagrama de Secuencia usualmente es un actor (el estímulo que inicia la interacción) o elementos colaborativos.

Un diagrama de Secuencia muchas veces se vincula directamente bajo el caso de uso al que se refiere. Esto ayuda a mantener juntos a los elementos, tanto en el modelo como en la

documentación que se produce. Para hacer esto, haga clic con el botón derecho sobre el caso de uso en el diagrama y seleccione: *Elemento Compuesto*. Alternativamente, desde el Explorador de Proyectos, haga clic con el botón derecho en el caso de uso y seleccione: *Nuevo Diagrama Hijo*, más adelante le explicaremos como elaborar un diagrama de secuencia.

Diagramas de Comunicaciones (formalmente Diagrama de Colaboraciones). Un diagrama de Comunicaciones muestra las interacciones entre los elementos en tiempo de ejecución en forma semejante a un diagrama de Secuencia. No obstante, los diagramas de Comunicaciones se usan para visualizar relaciones inter-objetos, mientras que los diagramas de Secuencia son más efectivos para visualizar procesamiento a lo largo del tiempo.

Los diagramas de Comunicaciones emplean asociaciones ordenadas y etiquetadas para ilustrar el procesamiento. Es importante la numeración para indicar el orden y el anidamiento del procesamiento. Un esquema de numeración podría ser 1, 1.1, 1.1.1, 1.1.2, 1.2, etc.

Comienza un nuevo segmento de números para una nueva capa de procesamiento, y sería equivalente a la invocación de un método.

Diagrama de Ejemplo

El ejemplo de abajo ilustra un diagrama de Comunicaciones entre instancias de objetos cooperantes. Tenga en cuenta el uso de niveles de mensajes para capturar flujos relacionados.

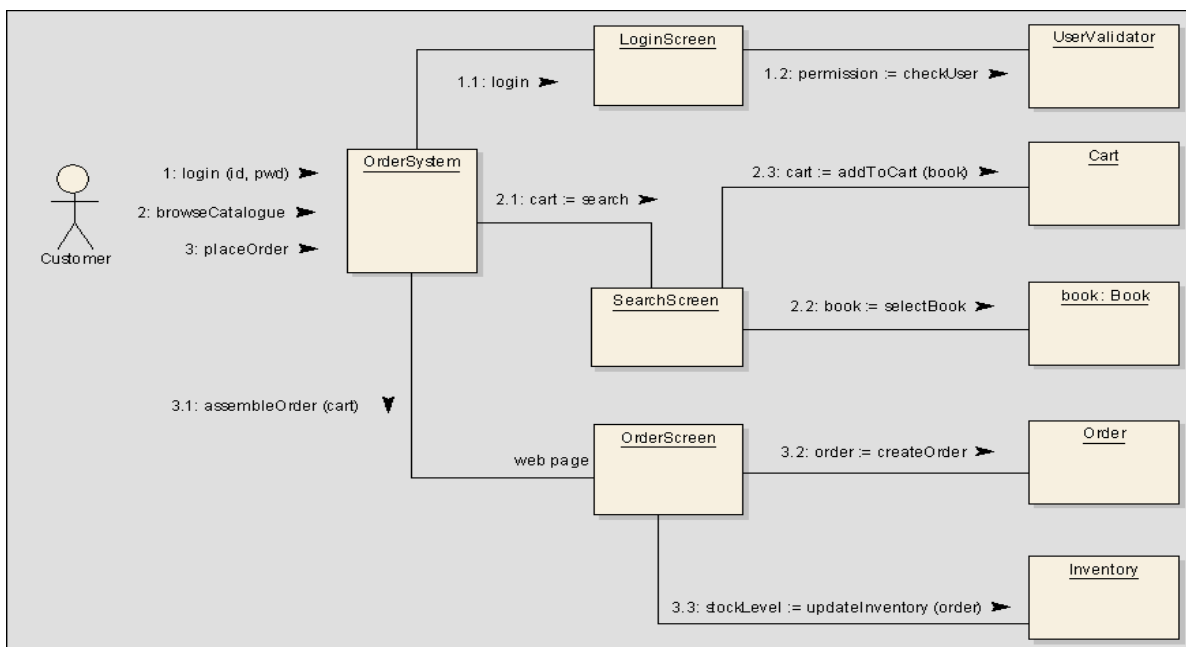


Figura 3. Ejemplo de un diagrama de comunicación para EA.

Elementos de los Diagramas de Comunicaciones		Conectores de los Diagramas de Comunicaciones	
 Actor		 Asociar	
 Objeto		 Dependencia	
 Límite		 Realización	
 Control		 Anidamiento	
 Entidad			
 Paquete			

Tabla 2. Elementos y conectores para un diagrama de comunicaciones.

En otras palabras en un diagrama de Comunicaciones se enfoca la interacción entre líneas de vida donde es central la arquitectura de la estructura interna y cómo ella se corresponde con el paso de mensajes. La secuencia de los mensajes se da a través de un esquema de numerado de la secuencia.

Los diagramas de Secuencias y de Comunicaciones expresan información similar, pero lo hacen de maneras diferentes.

Los diagramas de comunicaciones se conocían como diagramas de Colaboraciones en el UML 1.4. En EA, los flujos de mensajes se pueden ilustrar en diferente color lo que permite resaltar flujos de mensajes en particular en un diagrama de comunicación usando diferentes colores para cada conjunto de mensajes.

Para resaltar los colores en un diagrama de Comunicaciones, proceda con los siguientes pasos:

- Abra la ventana *Herramientas | Opciones* y vaya a la pestaña *Colores de Comunicaciones*.
- Asegúrese de que la casilla de verificación *Usar Colores en los Mensajes de Comunicaciones* esté marcada.
- Seleccione los colores que desee.
- En su diagrama de Comunicaciones, cada grupo de secuencia de mensajes ahora aparecerá en un color diferente como se muestra a continuación.

5.4 Correlación con el o los temas y subtemas del programa de estudio vigente. Al igual que en la práctica 4, en la unidad 3 del programa de la materia, se estipula que el alumno comprenda los conceptos de arquitectura de clases, clases según estereotipos y diagramas de secuencias y mediante estos elementos pueda elaborar un modelo de análisis siguiendo la metodología utilizada hasta ahora: RUP lo que tiene una relación directa con lo que se revisa en la presente práctica.

5.5 Material y equipo necesario. En esta práctica el alumno requerirá utilizar la herramienta CASE utilizada en las prácticas anteriores: EA, lo anterior para poder

desarrollar los diagramas de secuencias y comunicación utilizando estereotipos de manera de poder complementar el modelo de análisis que se comenzó por elaborar en la práctica 4.

5.6 Metodología. Para implementar el modelo de análisis con ayuda de EA, se recomienda seguir los siguientes pasos:

Del proyecto elaborado en la práctica 3 en EA se recomienda utilizar el archivo correspondiente y una vez abierto, por cada caso de uso del diagrama de casos de uso hacer lo siguiente:

- a) Dando clic izquierdo sobre el caso de uso aparece un menú contextual, seleccionar la opción *agregar* el cual despliega a su vez otro menú que en la opción más inferior tiene la opción *agregar diagrama*, la cual despliega una caja de dialogo en donde hay que nombrar el diagrama y seleccionar su tipo, en este caso: *diagrama de secuencia*.
- b) Seguir la secuencia correspondiente para trazar un diagrama de secuencia, hasta completar uno por cada caso de uso que contenga el diagrama obtenido en el modelo de requisitos de la práctica 3. Para hacer esto se recomienda consultar la bibliografía 4.
- c) De la misma manera que para el punto 1 elaborar un diagrama de comunicación para cada caso de uso, tomando en cuenta los estereotipos correspondientes establecidos en la práctica 4. Para hacer esto se recomienda consultar la bibliografía 5.
- d) Una vez completados los diagramas el alumno deberá de elaborar un glosario de términos detallando cada una de las clases que están presentes en los diagramas elaborados.

5.7 Sugerencias Didácticas. Se recomienda que el modelo de requisitos obtenido en la práctica 3 el alumno documente una matriz por cada caso de uso detallando las clases por estereotipo que ayuden a implementar dicho caso de uso.

Igualmente por cada caso de uso se deberán de elaborar dos diagramas: un diagrama de comunicación y un diagrama de secuencia de nueva cuenta esta labor podrá ser dividida entre los alumnos que conforman los equipos de acuerdo al subsistema que se le haya asignado a cada uno de ellos, lo que se sugiere es que mientras se vayan desarrollando los diagramas el profesor deberá de hacer varias revisiones de los mismos para que se aprenda a analizar un caso de uso mediante la elaboración de estos diagramas.

5.8 Reporte de alumnos (Resultados).

Por equipos los alumnos entregarán un reporte con 2 componentes principales.

- a) Un archivo EAP con el proyecto iniciado en la práctica anterior, conteniendo por cada caso de uso los diagramas de secuencias y comunicación correspondientes.
- b) Un glosario de términos elaborado en Word. Detallando todos elementos presentes en los diagramas anteriores.

5.9 Bibliografía.

1. Alfredo Weitzenfeld. 2005. Ingeniería de Software Orientada a Objetos con UML, Java e Internet. Thompson
2. Stephen R. Schach. 2005. Análisis y diseño orientado a objetos con UML y el proceso unificado. Mc.Graw Hill.
3. Spark Systems.. (2009). Guía de Usuario de Enterprise Architect 7.1. Solus S.A.
4. Consulta en youtube: <http://youtu.be/fbsG6Vwa4Tw>.
5. Consulta en youtube: http://youtu.be/gdf6_v3GX3g

Práctica No.6

6.1 Nombre: Elaboración de un modelo de diseño basado en responsabilidades para un sistema.

6.2 Objetivo: Que el alumno sea capaz de desarrollar un modelo de diseño para un sistema real bajo la metodología CRC.

6.3 Introducción. El modelo de diseño es un refinamiento y formalización adicional del modelo de análisis donde se toman en cuenta las consecuencias del ambiente de implementación. El resultado del modelo de diseño son especificaciones muy detalladas de todos los objetos, incluyendo sus operaciones y atributos.

Durante el diseño, se puede ver si los resultados del análisis son apropiados para su implementación. Si se descubren aspectos que no están claros en algunos de los modelos anteriores, estos deben ser clarificados, quizás regresando a etapas anteriores. La meta es refinarlo hasta que sea fácil escribir código fuente. Por lo tanto, es deseable tener un modelo de análisis ideal del sistema durante el ciclo de vida completo del sistema, dado que muchos de los cambios del sistema provienen de cambios en el ambiente de implementación. Tales cambios son entonces fácilmente incorporados, ya que el mismo modelo de análisis sirve de base para el nuevo modelo de diseño. De esta manera se puede ver el modelo de diseño como una especialización del modelo de análisis según un ambiente de implementación específico.

Se consideran por separado los dos aspectos principales del modelo de diseño:

- El *diseño de objetos* y
- El *diseño de sistema*.

Diseño de Objetos. En este diseño se refina y formaliza el modelo para generar especificaciones muy detalladas de todos los objetos, incluyendo sus operaciones y atributos. Se describe cómo interaccionan los objetos en cada caso de uso, especificando qué debe hacer cada operación en cada objeto. Este paso genera las interfaces de los objetos, las cuales deben ser luego implementadas mediante métodos.

Diseño de Sistema. Se adapta el modelo al ambiente de implementación. Este paso incluye identificar e investigar las consecuencias del ambiente de implementación sobre el diseño.

En el diseño de sistema, se recomienda tomar las decisiones de implementación estratégicas siguientes:

- El diseño de la base de datos del sistema,
- El diseño de las bibliotecas de componentes o su identificación para ser utilizadas en el nuevo sistema.
- La identificación del lenguaje de programación a utilizar.

- La definición de la forma en que se manejarán los procesos, incluyendo comunicación y requisitos de rendimiento,
- El diseño del manejo de excepciones y la recolección de basura.

En general, si el ambiente de implementación tiene pocas consecuencias en el sistema, el diseño se basará casi exclusivamente en el diseño de objetos, o sea, es una extensión directa y detallada del modelo de análisis describiendo los atributos y operaciones del sistema. Por el contrario, si el ambiente de implementación afecta de manera importante al sistema, el diseño se basará en una combinación de diseño de objetos y el diseño del sistema, o sea, en un modelo de análisis que dirige el resultado final del sistema aunque este será adaptado de manera importante al ambiente de implementación. En nuestro caso minimizaremos el efecto del ambiente de implementación sobre el sistema, razón por la cual comenzaremos con la descripción del diseño de objetos.

Diseño de Objetos. Para el diseño de objeto se tomará en cuenta el llamado: *diseño por responsabilidades (RDD - Responsibility-Driven Design)*. Este diseño está basado en un modelo *cliente-servidor* donde las clases son vistas como clientes cuando generan alguna petición hacia otra clase y como servidores cuando reciben peticiones de alguna otra clase.

La funcionalidad ofrecida por las clases servidores se define en término de sus *responsabilidades*, las cuales deben ser satisfechas por lograr sus *servicios* con las demás clases. Los servicios y responsabilidades corresponderán finalmente a los métodos de la clase. A su vez, las clases servidores pueden tener *colaboraciones* con otras clases para lograr la satisfacción de responsabilidades que por sí solas no pueden lograr. Como consecuencia de esto, se integran las responsabilidades y colaboraciones entre clases para definir *contratos* los cuales definen la naturaleza y alcance de las interacciones cliente-servidor. Los contratos y colaboraciones representan los requisitos de servicios entre las clases. En resumen, se buscará identificar los contratos entre los objetos cliente y servidor diseñando su distribución entre los distintos objetos del sistema.

La metodología del diseño basado en responsabilidades se compone de los siguientes pasos:

1. Especificación de las *tarjetas de clases*.
2. Identificación de las *responsabilidades* del sistema.
3. Identificación de las *colaboraciones* del sistema.
4. Identificación de las *jerarquías* de herencia del sistema.
5. Identificación de los *contratos* de servicio del sistema.
6. Identificación de los *subsistemas* del sistema.
7. Identificación de los *protocolos* del sistema.
8. Identificación de los *atributos* del sistema.
9. Especificación de los *algoritmos* del sistema.

1. **Especificación de las tarjetas de Clase.** Las *tarjetas de clases* (también conocidas como tarjetas *CRC: Clase-Responsabilidad-Colaboración*) permiten al diseñador visualizar las diferentes clases de manera independiente y detallada. El esquema para una tarjeta de clase se muestra en la tabla siguiente:

Clave:
Descripción:
Módulo:
Estereotipo:
Propiedades:
Superclases:
Subclases:
Atributos:

Figura 1. Formato de una tarjeta CRC.

La tarjeta CRC se divide en tres secciones:

Encabezado: consistiendo del nombre de la *clase*, una *descripción* de la clase (similar a la descrita en el diccionario de clases de análisis), el *módulo* al que pertenece la clase, el *estereotipo* de la clase (entidad, interface o control), las *propiedades* de la clase (abstracta o concreta), una lista de *superclases*, una lista de *subclases* y una lista de *atributos*.

Dos columnas debajo del encabezado: Correspondientes a las *responsabilidades* (a la izquierda) y *colaboraciones* (a la derecha) de la clase. Eventualmente en la columna izquierda se incluirá información sobre los *contratos*. El número de filas en estas dos columnas es extensible y no está limitado al número que aparece en el diagrama de la Tabla

2. Identificación de las responsabilidades. Uno de los esfuerzos más grandes del desarrollo y que involucra mayor complejidad es la especificación del comportamiento de cada una de las clases del sistema. A diferencia de la estructura interna de una clase, representada mediante sus atributos, los comportamientos corresponden a las operaciones y métodos de las clases. Dado que por lo general el número resultante de métodos en un sistema es mucho mayor que el de clases, el proceso de diseño involucra mayor complejidad que el de análisis. Si consideramos también la existencia de los atributos y las propias implementaciones de los métodos dentro de cada clase, la complejidad aumenta drásticamente. Sin embargo, esta complejidad no radica exclusivamente en el número de métodos, sino en el hecho que potencialmente todos los métodos de un objeto pueden ser llamados por todos los objetos del sistema. Esta es la verdadera fuente de la complejidad en la arquitectura del sistema, ya que cualquier cambio en uno de estos métodos afectará potencialmente a todo el resto del sistema. Por lo tanto, es necesario llevar a cabo un proceso muy cuidadoso para la identificación del comportamiento de las diversas clases.

En general, el comportamiento de las clases no debe descomponerse directamente en operaciones, sino seguir un procedimiento más natural comenzando con una descripción verbal de las *responsabilidades* o roles que tiene cada clase dentro del sistema.

Obviamente, cada objeto instanciado de una misma clase tendrá responsabilidades similares a las descritas por la clase. Las responsabilidades ofrecidas por un objeto corresponden a los *servicios* apoyados por éste. A partir de estas responsabilidades se podrá eventualmente determinar las operaciones que cada objeto debe tener e incluso el conocimiento que el objeto posee.

Las responsabilidades se identifican a partir de los casos de uso generados durante el modelo de análisis. Para ello, se revisa el modelo de casos de uso, haciendo una lista o subrayando todas las frases descritas para determinar cuáles de éstas representan acciones que algún objeto dentro del sistema deba ejecutar. Una de las decisiones más importantes durante la identificación de responsabilidades es a qué clase o clases se les debe asignar. Para ello se debe examinar el contexto en el cual se identificaron las responsabilidades.

Se debe asignar responsabilidades a las clases que almacenen la información más relacionada con la responsabilidad. En otras palabras, si un objeto requiere cierta información, es lógico asignarle la responsabilidad de mantener esa información. Si la información cambia, no será necesario enviar mensajes de actualización a otros objetos. Esto también significa que la responsabilidad de mantener la información no debe ser compartida. Compartir información implica una duplicación que puede dar lugar a inconsistencias. Si más de un objeto tiene responsabilidad sobre la misma información se debería reasignar la responsabilidad a un sólo objeto. También puede ocurrir que cierta responsabilidad agrupe varias responsabilidades juntas. En tal caso se puede dividir o compartir la responsabilidad entre dos o más objetos. Si se vuelve difícil decidir a quien se debe asignar una responsabilidad, se puede experimentar con diversas alternativas, con la ventaja de que durante las primeras etapas del diseño, este proceso no es muy costoso. Será mucho más difícil y tomará más tiempo hacerlo después de haber implementado el sistema completo.

En cuanto a la especificación de las responsabilidades, se debe describir las responsabilidades en términos generales, para poder luego encontrar más fácilmente responsabilidades compartidas entre clases. Por lo tanto, no es necesario considerar aspectos específicos de las responsabilidades, como nombres particulares o incluso parámetros. Otra consideración es asegurarse que los ciclos de responsabilidades y colaboraciones entre clases no sean interrumpidos y mantengan siempre una correspondencia con la arquitectura establecida durante el análisis.

3. Identificación de las colaboraciones. Es indispensable que los objetos dentro de un programa colaboren entre sí o de lo contrario el programa consistirá de múltiples “mini-programas” independientes. Las colaboraciones entre objetos se dan en base a las relaciones entre las responsabilidades de las distintas clases.

Dado la infinidad de posibles relaciones entre clase, las colaboraciones son la fuente principal de complejidad en el diseño de objetos.

Las colaboraciones representan solicitudes de un objeto *cliente* a un objeto *servidor*. Los objetos pueden jugar el papel de clientes o servidores dependiendo de su actividad en ese momento. Un objeto juega el papel de servidor cuando satisface una solicitud hecha por un

objeto cliente. Desde el punto de vista del cliente, cada una de sus solicitudes de servicio o colaboración se asocia con una responsabilidad implementada por algún servidor. A su vez, la clase que ofrece el servicio puede solicitar una colaboración con alguna otra clase servidor.

Esta cadena de solicitudes y servicios puede extenderse según sea necesario. El proceso de identificación de colaboraciones se basa en la funcionalidad de la aplicación y de la arquitectura de clases propuesta, algo que fue hecho durante la etapa de análisis. Para identificar colaboraciones se analiza las responsabilidades de cada clase, decidiendo si cada una de estas clases es capaz de satisfacer sus responsabilidades por si misma o si requiere de otra clase para lograrlo. Se analiza qué tanto sabe cada clase y qué otras clases necesitan su conocimiento o funcionalidad. El patrón de colaboraciones revela el flujo de control e información dentro del sistema. Analizando los patrones de comunicación entre objetos puede revelar cuando una responsabilidad se ha asignado de forma incorrecta., incluso puede revelar responsabilidades ausentes. Se analizan las colaboraciones estudiando qué objetos dentro del sistema tienen el conocimiento que se busca. Se busca aquellas clases que colaboran y las que no colaboran con las demás, qué grupos de clases parecen trabajar juntas y donde existen cuellos de botella. A través de las colaboraciones se analizan las dependencias entre clases. Si una clase no tiene colaboraciones con otras clases, en otras palabras, nadie colabora con ella y ella no colabora con nadie, esta clase debe descartarse. Sin embargo, antes de hacerlo es bueno revisar el diseño completo verificando todas las clases y colaboraciones. Es importante recordar que las colaboraciones son en una sola dirección, y que ciertos tipos de clases son típicamente clientes o servidores, ubicándose la clase en uno u otro extremo de la colaboración. En la mayoría de los casos, las clases correspondientes al estereotipo de *borde* y *entidad* no son clientes de otros objetos en el sistema, si no que existen para ser servidores, donde otras clases colaboran con ellas, típicamente las clases de *control*.

Durante el proceso de identificación de las colaboraciones se toma la tarjeta de clase que juega el papel de cliente, escribiendo en la columna derecha el nombre de la clase que juega el papel de servidor o sea la clase que colabora para la responsabilidad particular. Se escribe ese nombre directamente a la derecha de la responsabilidad a la cual la colaboración ayuda a satisfacer. Aunque una responsabilidad pudiera requerir de varias colaboraciones, se escribe el nombre de la clase principal como servidor de la colaboración. Las demás clases típicamente pasan a ser parámetros dentro de las llamadas de métodos al momento de implementarse, por lo cual se pueden mantener como parte de la responsabilidad general. Por otro lado, si varias responsabilidades requieren una misma clase para colaborar, se graban varias colaboraciones, una para cada responsabilidad. Se debe asegurar que la responsabilidad correspondiente exista para cada colaboración que se grabe. Se debe grabar la colaboración aunque ésta corresponda a otras instancias de la misma clase. Se debe omitir si la colaboración corresponde al mismo objeto ya que se busca identificar relaciones entre objetos o clases pero no llamadas dentro del mismo objeto.

El resultado de un ejemplo de diseño basado en responsabilidades se documenta en la tabla siguiente en donde se pueden ver las responsabilidades y colaboraciones para la clase borde de InterfaceUsuario, que es muy común en la mayoría de los sistemas.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
<i>despliega</i>	PantallaPrincipal
envía el evento "OK"	ManejadorPrincipal
envía el evento "Salir"	ManejadorPrincipal
envía el evento "Registrarse por Primera Vez"	ManejadorPrincipal
<i>despliega</i>	PantallaServicio
envía el evento "Obtener Registro"	ManejadorServicio
<i>despliega</i>	PantallaCrearRegUsuario
envía el evento "Registrar"	ManejadorRegistroUsuario
envía el evento "Salir"	ManejadorRegistroUsuario
<i>despliega</i>	PantallaObtenerRegUsuario
envía el evento "Actualizar"	ManejadorRegistroUsuario
envía el evento "Eliminar"	ManejadorRegistroUsuario
envía el evento "Registrar Tarjeta"	ManejadorRegistroUsuario
envía el evento "Servicios"	ManejadorRegistroUsuario
<i>despliega</i>	PantallaCrearRegTarjeta

Figura 2. Ejemplo de tarjeta CRC con responsabilidades y colaboraciones.

4. **Identificación de las jerarquías de herencia del sistema.** El diseño de las jerarquías de herencia es uno de los aspectos de programación más importantes de la orientación a objetos.

Mediante la herencia se puede lograr una buena reutilización del código del sistema, logrando arquitecturas de clases más compactas lo cual puede reducir radicalmente el tamaño del sistema final.

La herencia se identifica a partir de las responsabilidades y colaboraciones obtenidas anteriormente. De manera general, existen dos formas de aprovechar la herencia. La forma más común es la creación de superclases que guarden responsabilidades comunes a múltiples clases. La forma adicional y más avanzada está relacionada con el *polimorfismo* y busca aprovechar no sólo responsabilidades comunes sino también colaboraciones comunes entre clases. Estos dos enfoques también sirven de base para la extensibilidad de la arquitectura de clases. Por ejemplo, si varias clases definen una responsabilidad similar, se puede introducir una superclase de la cual estas clases hereden la responsabilidad común. Las nuevas superclases típicamente son clases abstractas. Cuanto mayor sea el número de clases concretas que puedan extenderse a partir de una funcionalidad abstracta, mayor será la probabilidad de que la abstracción sobreviva las pruebas del tiempo y mejoras del software.

Se necesita solo una responsabilidad para definir una superclase abstracta, pero se necesita por lo menos dos subclases antes de esperar diseñar una abstracción general útil. Si no se tiene o no se prevé por lo menos dos casos, no se debe perder tiempo en construir la abstracción. Probablemente no se pueda diseñar funcionalidad apropiada general sin varios ejemplos concretos que sirvan de guía.

En esta etapa se debe revisar las tarjetas de clases y clasificar cada clase según si es concreta o abstracta. Se debe además agregar tarjetas para las superclases (abstractas o concretas) según sea necesario y reasignar responsabilidades para corresponder al nuevo diseño. Mediante la reasignación de responsabilidades se busca producir jerarquías de clases que puedan reutilizarse y extenderse más fácilmente. Aunque llevaremos a cabo una sola etapa de diseño de jerarquías, la herencia debe aplicarse continuamente a lo largo del diseño de objetos. Se listan en la tarjeta de clase las responsabilidades propias no heredadas y aquellas responsabilidades que se sobrescriben. No es necesario volver a escribir las responsabilidades que se heredan de las superclases. Nótese en las siguientes secciones que estaremos agregando nuevas superclases, para lo cual agregaremos las correspondientes tarjetas de clases, y estaremos agregando información en las secciones de subclases y superclases, además de la especificación de si la clase es concreta o abstracta.

5. Identificación de los *contratos de servicio del sistema*. Un *contrato* es un mecanismo de diseño para agrupar las distintas responsabilidades de una clase que están relacionadas lógicamente entre sí. Los contratos sirven como indicadores de los diversos servicios provistos por cada clase. El objetivo final del contrato es ser un elemento de abstracción adicional en el manejo de la complejidad del sistema, dado que la funcionalidad completa del sistema dada a bajo nivel por las responsabilidades, puede ser vista a alto nivel como un grupo de servicios o contratos.

El contrato no es simplemente otro nombre para la responsabilidad, ya que la responsabilidad corresponde a una acción específica, mientras que un contrato define un conjunto de responsabilidades cercanas una de la otra.

Cada responsabilidad puede ser parte de un sólo contrato, aunque no tiene que ser necesariamente parte de algún contrato. Esto ocurre cuando las responsabilidades representan comportamiento que una clase debe tener, pero que son privadas a los propios objetos. En general, una clase puede apoyar uno o más contratos, aunque a menudo una clase con varias responsabilidades apoya un sólo contrato.

Un contrato entre dos clases representa una lista de servicios que una instancia de una clase puede solicitar de una instancia de otra clase. Todos los servicios especificados en un contrato particular son la responsabilidad del servidor para ese contrato. Las responsabilidades correspondientes al contrato deben ser ofrecidas públicamente.

En general, un contrato entre un cliente y un servidor no especifica cómo se hacen las cosas, solo qué se hace. Los contratos especifican quien colabora con quien, y qué se espera de la colaboración. El contrato cliente-servidor divide los objetos en dos categorías aquellos que proveen servicios (servidores), y aquellos que piden servicios (clientes). De tal manera,

un contrato es una lista de pedidos que le hace un cliente a un servidor. Ambos deben satisfacer el contrato, el cliente hace la solicitud y el servidor respondiendo de manera correspondiente.

Como se mencionó anteriormente, cliente y servidor son roles que juegan los objetos en cierto momento y no características inherentes de los propios objetos. Un objeto puede tomar el rol de cliente o servidor en relación a otros objetos. Por ejemplo, los *componentes* casi siempre juegan el rol de servidores, ya que satisfacen una responsabilidad específica.

Raramente los componentes necesitaran pedir servicios del propio sistema para poder satisfacer esas responsabilidades. Por el contrario, los *marcos (frameworks)* generalmente toman ambos roles, ya que se integran a las aplicaciones para solicitar y proveer funcionalidad.

Se puede determinar qué responsabilidades pertenecen a cuales contratos siguiendo ciertos criterios. Una forma de encontrar responsabilidades relacionadas es buscar responsabilidades que serán usadas por el mismo cliente. Aunque es posible que algunas clases den servicio a diferentes clientes mediante diversas responsabilidades, es más significativo cuando dos o más responsabilidades de una clase dan servicio a los mismos clientes. En tal caso sería innecesario definir distintos contratos. En su lugar, se puede abstraer de las responsabilidades específicas un solo contrato, satisfaciendo la responsabilidad general.

Si una clase define un contrato que tiene relativamente poco en común con el resto de los contratos definidos por esa clase, este contrato debe ser movido a una clase diferente, usualmente una superclase o subclase. De tal manera, se puede refinar las jerarquías de clases maximizando la cohesión de contratos para cada clase, lo cual minimizará el número de contratos apoyados por cada clase. Esto es algo deseable, ya que cuanto menos contrato exista, más fácil será comprender el sistema. En general, la mejor manera de reducir el número de contratos es buscar responsabilidades similares que puedan generalizarse. Esto también resultará en jerarquías de clases más extensibles.

Un buen diseño hace un balance entre clases pequeñas con pocos contratos, fáciles de comprender y reutilizar, junto a un número reducido de clases más complejas cuyas relaciones entre ellas se puede comprender más fácilmente.

Una técnica para definir contratos es comenzar definiendo primero los contratos de las clases más arriba en las jerarquías. Posteriormente, se definen nuevos contratos para las subclases que agregan nueva funcionalidad. Se debe examinar las responsabilidades para cada subclase y determinar si estas representan nueva funcionalidad o si son simplemente formas específicas de expresar responsabilidades heredadas, en cuyo caso serían parte del contrato heredado.

En cada tarjeta de clase se divide la sección de responsabilidades en dos. En la parte superior se especifica una sección para los contratos, mientras que en la parte inferior se especifica una sección para las responsabilidades privadas. Cada contrato debe incluir un nombre y un número de contrato. Se debe listar cada contrato para el cual esta clase es un

servidor. También se debe listar contratos heredados, e indicar la clase de la cual se heredan, aunque no hay necesidad de repetir los detalles del contrato. Para cada contrato, se debe listar las responsabilidades de la clase en la cual se basan, asignando cada responsabilidad al contrato correspondiente.

Si la responsabilidad requiere colaborar con una clase que define varios contratos, se debe indicar el número del contrato correspondiente entre paréntesis en la columna de la colaboración, para así simplificar el seguimiento de qué responsabilidad se relaciona con qué contrato.

6. Identificación de Subsistemas. Como se ha podido apreciar hasta el momento, la complejidad del sistema aumenta a medida que se incorporan nuevos detalles en el diseño, algo que por lo general es inevitable. Para lograr un mejor manejo de esta complejidad introducimos el concepto de *subsistemas*, el cual permite dividir el sistema completo en diversas partes, inspirado en la idea de “divide y conquista”. Los subsistemas permiten agrupar objetos relacionados para lograr cierta funcionalidad en “mini-sistemas”. El sistema completo se compone de estos “mini-sistemas” o subsistemas y cada subsistema puede ser subdividido en subsistemas adicionales o ser definido en términos de los objetos finales. Los subsistemas también permiten trabajar en diferentes partes del sistema en paralelo mediante su asignación a múltiples diseñadores.

La organización en subsistemas se logra a partir de los contratos identificados anteriormente entre los objetos.

Externamente, los subsistemas son mecanismos de encapsulamiento, vistos como “cajas negras”, donde sus objetos cooperan para proveer una unidad de funcionalidad claramente delimitada por el subsistema. Internamente, los subsistemas pueden tener estructuras complejas, con clases colaborando entre sí para satisfacer sus distintas responsabilidades contribuyendo al objetivo general del subsistema, o sea, satisfacer sus responsabilidades.

Se busca tener un fuerte acoplamiento funcional dentro de cada subsistema y un débil acoplamiento entre subsistemas, en otras palabras, se busca tener la mínima comunicación entre los diferentes subsistemas.

Un subsistema bien diseñado tiene pocas clases o subsistemas que directamente apoyan contratos, y un número mayor de colaboraciones entre clases y subsistemas internos.

Es importante distinguir entre el concepto de *módulo* y *subsistema*. Un módulo agrupa clases, correspondiente a *bibliotecas*, o sea, organizaciones estáticas definidas para almacenar y facilitar el acceso a las clases. Esto es similar al concepto de *directorios* en una computadora. Son estructuras puramente organizacionales sin ningún efecto sobre los propios procesos. Por otro lado, el subsistema agrupa objetos, siendo una abstracción dinámica correspondiente a la funcionalidad del proceso. En general, los objetos pertenecientes a un subsistema son instancias de clases que deben existir forzosamente en algún módulo. Mientras que las clases no deben duplicarse entre módulos, se permite

instanciar objetos de la misma clase en múltiples subsistemas, siendo esto parte de la reutilización de código.

Los subsistemas deben incluir completos o no ser incluidos, de manera que un sistema pueda ser ofrecido con o sin ciertos subsistemas. Dada la dependencia entre subsistemas, si se incluye un subsistema, se debe también entregar el subsistema del cual depende. Típicamente, los objetos de las clases entidad son reutilizados en los diversos subsistemas, mientras que los objetos de control y por lo general las de borde son propias a cierto subsistema.

Los subsistemas son una pieza fundamental en el manejo de la modularidad y extensibilidad del sistema. Por tal motivo, los subsistemas deben mantener una buena correspondencia con los casos de uso, de manera que cualquier cambio causado por modificaciones en la funcionalidad del sistema pueda ser rastreado a los subsistemas afectados. Cabe resaltar que los subsistemas no existen durante la ejecución de la aplicación, son puramente abstracciones del sistema.

Si consideramos que la mayor fuente de complejidad en el sistema radica en las relaciones entre objetos (a través de sus colaboraciones), la meta es reducir o al menos manejar de mejor manera estas relaciones. Un enfoque es definir los subsistemas de manera que, aunque el número total de relaciones sea el mismo, las relaciones estarán organizadas en grupos (*clusters*). Si se observan que cierto grupo de objetos están muy relacionados entre sí pero no tanto con otro grupo de objetos, esto debe asignarse a subsistemas comunes.

Como parte del proceso de identificación de subsistemas, se define un protocolo de comunicación para cada subsistema, el cual especifica las posibles interacciones entre subsistemas. Esto se logra exportando o haciendo públicos ciertos contratos pertenecientes a ciertos objetos del subsistema. La interface de estos objetos, sus servicios, definen la interface del subsistema completo. Para determinar los contratos apoyados por un subsistema, se tiene que encontrar todos los objetos o clases que proveen servicios a clientes fuera del subsistema.

Al igual que las clases, se debe describir los contratos ofrecidos por cada subsistema. Esto se hace introduciendo el concepto de tarjeta de subsistemas de manera análoga al de clases.

Dado que los subsistemas son solamente entidades conceptuales, estos no pueden directamente satisfacer ninguno de sus contratos. En su lugar, los subsistemas delegan cada contrato a una clase interna que lo satisface.

Los subsistemas se describen en *tarjetas de subsistemas*, un subsistema por tarjeta. Cada tarjeta incluye un nombre en la primera línea y dos columnas, como se puede ver en la tabla siguiente:

Subsistema: Nombre del Subsistema	
Descripción: Descripción del Subsistema	
Clases: Grupo de clases (instanciadas) que participan en este subsistema.	

Figura 3. Tarjeta de subsistemas.

En la columna izquierda se muestran los contratos ofrecidos de manera externa al subsistema, mientras que en la columna derecha, al lado de cada contrato se escribe la clase interna que ofrece el servicio, en otras palabras, a quien se le delega el contrato. Es importante notar que las clases abstractas no se asignan a ningún subsistema ya que no podrían ser instanciados directamente. En su lugar asignamos únicamente clases concretas.

7. Identificación de los protocolos del sistema. Una vez completadas las etapas anteriores, se debe detallar la especificación de cada clase hasta llegar a métodos y atributos finales. Aunque pudiéramos generar una especificación completa a nivel de “pseudo-código” o incluso código en un lenguaje particular, esto sería sobre especificar el diseño ya que el diseño debe definir los aspectos más relevantes de la solución pero sin ser el código final. El diseño debe dar cierta libertad al programador o implementador siempre y cuando se mantenga dentro del marco de la arquitectura de objetos generada hasta el momento. Entonces, ¿hasta dónde debe llegar el diseño? Nuevamente, la respuesta está relacionada con aspectos en el manejo de la complejidad. Si el diseño desarrollado ya resuelve los aspectos más importantes de la arquitectura, en particular todo lo relacionado con la asignación de responsabilidades, entonces ya hemos resuelto gran parte del problema. Sin embargo, existen otros aspectos que son fuente de complejidad más allá de las responsabilidades, estos incluyen los aspectos algorítmicos, estructuras de datos, entre otros junto con la adecuación a los demás aspectos consideramos dentro del “diseño de sistema”. Cuando ya no existen decisiones “importantes” que pudieran afectar de manera importante la arquitectura del sistema entonces ya se puede continuar con la implementación.

Al establecer los protocolos se incluye de manera importante la definición precisa de las responsabilidades públicas, ya que cualquier cambio en ellas afectará a todos los respectivos clientes. Para ello se define el concepto de protocolo, donde un protocolo corresponde al conjunto de firmas correspondientes a las distintas responsabilidades de una clase. El objetivo de los protocolos es refinar las responsabilidades hasta llegar a métodos precisos dando especial énfasis a las responsabilidades agrupadas en los contratos ofrecidos por las clases.

En general, las responsabilidades privadas representan notas de implementación para un programador y no deben sobre-especificarse. Sin embargo, en algunos casos se debe generar protocolos para responsabilidades privadas. Por ejemplo, si una superclase abstracta será implementada por un programador y sus subclasses implementadas por otro, las responsabilidades privadas usadas por sus subclasses deben especificarse completamente.

Se busca también incrementar la reutilización en el sistema si logramos que los protocolos contengan sólo uno pocos parámetros, simplificando su comprensión e incrementando la probabilidad de descubrir nuevas bases para la herencia. Se debe escoger los nombres con cuidado, siguiendo un patrón uniforme en el momento de asignar estos nombres.

Durante esta etapa es común descubrir sitios en el diseño donde el modelo fue impreciso, incorrecto, o pobremente comprendido. Por lo tanto, puede que sea necesario tener que repetir etapas anteriores del proceso de diseño antes de poder completar las definiciones de clases y contratos.

8. Identificación de los *atributos* del sistema. Lo que haremos a continuación, es detallar un poco más nuestro diseño especificando *atributos* correspondientes a estructuras de datos y *algoritmos* que especifiquen la manera de implementar las diversas responsabilidades correspondientes a los múltiples contratos de la aplicación.

En general, los *atributos* corresponden a los aspectos estructurales de las clases, como son los valores (números y textos), junto con las referencias a otros objetos. Estos conceptos varían de gran manera entre lenguajes. Los atributos correspondientes a referencias deben agregarse de acuerdo a las colaboraciones establecidas durante el diseño. Los atributos que guardan valores son lo último que se especifica en el diseño de objetos.

9. Especificación de los *algoritmos* del sistema. Los *algoritmos* definen la lógica utilizada por cada operación para resolver la responsabilidad a la cual corresponden. En general, muchas responsabilidades son suficientemente simples que no requieren de algoritmos, como son las responsabilidades de delegar entre los objetos para consultar o modificar los valores de los atributos. Sin embargo, es especialmente importante especificar los algoritmos para implementar funciones de lógica más compleja, como ordenar un conjunto de números o cadenas, o hacer cálculos de intereses sobre cuentas bancarias. Los algoritmos pueden especificarse de manera declarativa o procedural dependiendo de su complejidad. Cuando se especifica el algoritmo de manera procedural, esto típicamente se hace mediante un diagrama de flujo. Por el contrario, un algoritmo especificado de manera declarativa, típicamente se hace mediante una especificación textual en cuyo caso se debe usar algún conocimiento adicional para implementar el algoritmo. Vale la pena resaltar que los aspectos algorítmicos pueden llegar a ser una de las fuentes de mayor complejidad en un sistema.

6.4 Correlación con el o los temas y subtemas del programa de estudio vigente. La cuarta unidad del programa de la materia se refiere al modelado de diseño de la metodología RUP, en la cual se establece que el alumno aprenda a usar una estrategia de diseño como la que repasamos en la presente práctica la cual se dividió en el diseño de objetos y el diseño de sistema y que coincide con los puntos 4.2 y 4.3 del programa de la materia. En esta práctica nos concretamos a revisar el diseño de objetos, mientras que en la siguiente veremos el diseño de sistema.

6.5 Material y equipo necesario. Para esta práctica seguiremos con el uso del EA para completar la documentación de cada clase identificada en la práctica 5.

6.6 Metodología. La metodología para el diseño basado en Responsabilidades, se basa en los pasos descritos en la introducción que son los siguientes:

- a. Especificación de las *tarjetas de clases*.
- b. Identificación de las *responsabilidades* del sistema.
- c. Identificación de las *colaboraciones* del sistema.
- d. Identificación de las *jerarquías* de herencia del sistema.
- e. Identificación de los *contratos* de servicio del sistema.
- f. Identificación de los *subsistemas* del sistema.
- g. Identificación de los *protocolos* del sistema.
- h. Identificación de los *atributos* del sistema.
- i. Especificación de los *algoritmos* del sistema.

La metodología la describimos a continuación:

- a) Utilizar la siguiente tarjeta de clase para condensar el trabajo que se desarrollará en los siguientes pasos:

Nombre de la Clase	
Descripción de la Clase	
Tipo de Estereotipo	
Superclases	
Subclases	
Atributos	
Responsabilidades	Colaboraciones

Figura 4. Molde para completar la documentación de una clase

- b) Completar la columna: Responsabilidades, siguiendo lo descrito en la introducción.
- c) Completar la columna: Colaboraciones, siguiendo lo descrito en la introducción.
- d) De las colaboraciones descritas en cada tarjeta de clase generar el correspondiente diagrama de jerarquías de clase tomando como referencia lo establecido en la introducción y que mediante un diagrama como el siguiente:

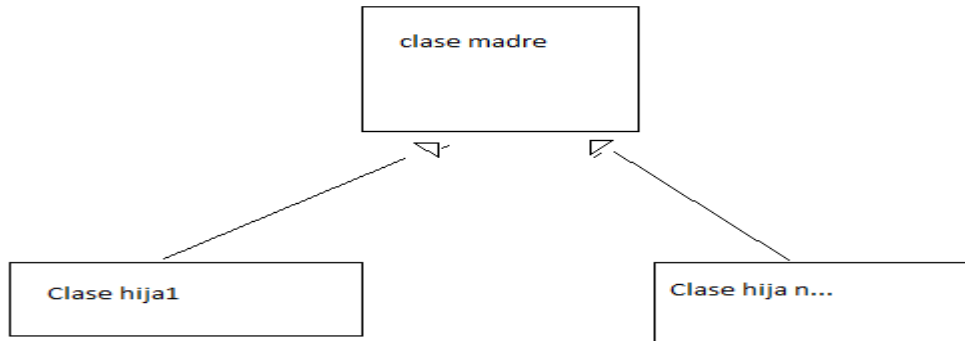


Figura 5. Diagrama de jerarquía de clase

- e) Para la identificación de los contratos se modifica la tabla del inciso a, definiendo a partir de las colaboraciones los contratos correspondientes por lo que la tabla quedará de la siguiente manera:

Nombre de la Clase	
Descripción de la Clase	
Tipo de Estereotipo	
Superclases	
Subclases	
Atributos	
Contratos	
Número y Nombre del contrato	

Figura 6. Tarjeta de clase con contratos.

- f) Identificación de los subsistemas. Para dividir el sistema en subsistemas, se recomienda llenar la siguiente tabla: Se deberán de elaborar un número de tablas equivalente a la cantidad de subsistemas identificados.

Subsistema: Nombre del Subsistema	
Descripción: Descripción del Subsistema	
Clases: Grupo de clases (instanciadas) que participan en este subsistema.	

Figura 7. Tarjetas de subsistemas

- g) Para los tres últimos incisos se recomienda utilizar la tabla del inciso a completando los 3 componentes que son: protocolos, atributos y métodos siguiendo lo establecido en la introducción de esta práctica, quedando finalmente nuestra tabla de la siguiente manera:

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos: Manejador manejador, Pantalla pantalla	
Contratos	
1. Desplegar Pantalla	
desplegarPantalla(Pantalla) devuelve void <i>Método encargado de desplegar las pantallas enviadas como parámetros. Se delega el despliegue particular a cada pantalla.</i>	Pantalla (1) : PantallaPrincipal (1), PantallaServicio (1), PantallaCrearRegUsuario (1), PantallaObtenerRegUsuario (1), PantallaCrearRegTarjeta (1), PantallaObtenerRegTarjeta (1)
2. Enviar Evento	
enviarEvento(Evento) devuelve void <i>Método encargado de recibir eventos del sistema de ventanas a través de las diversas pantallas. Se envía el evento recibido a los distintos manejadores.</i>	Manejador (1) : SubsistemaPrincipal (1), SubsistemaServicio (1), SubsistemaRegistro (1)

Figura 8. Ejemplo de una tabla CRC terminada, indicando protocolos, atributos y una breve descripción de los métodos a utilizar por cada contrato.

6.7 Sugerencias Didácticas. El desarrollo de esta práctica puede resultar extremadamente fatigoso máxime si consideramos que en un sistema de gran tamaño el documentar tantas clases puede requerir mucho tiempo de trabajo por lo que se sugiere que el trabajo sea modularizado en subsistemas los cuales sirvan para delegar el mismo en forma ordenada a los alumnos (un subsistema por cada alumno), de forma que cada alumno pueda hacerse cargo de documentar las clases más relevantes de cada subsistema (10 ejemplos de cada

clase por estereotipo), con la finalidad de aprender a hacer tal documentación. Deberá quedar claro que la documentación obtenida en este diseño se utilizará con el beneficio de que la reutilización de las clases en sistemas futuros será mucho más eficiente y fácil si se cuenta con las tarjetas correspondientes.

6.8 Reporte de alumnos (Resultados). El alumno deberá de entregar como evidencia en un reporte adecuado:

- a) Una tarjeta de subsistema según el número de subsistemas identificados (figura 7)
- b) Un conjunto de 10 tarjetas CRC por cada subsistema (Figura 1).
- c) Un diagrama de jerarquías de clases (Figura 1).

6.9 Bibliografía

1. Alfredo Weitzenfeld. 2005. Ingeniería de Software Orientada a Objetos con UML, Java e Internet. Thompson
2. Stephen R. Schach. 2005. Análisis y diseño orientado a objetos con UML y el proceso unificado. Mc.Graw Hill

Práctica No.7

7.1 Nombre: Generación de un modelo de diseño con la ayuda de una herramienta CASE.

7.2 Objetivo: Que el alumno pueda desarrollar un modelo de diseño usando para lo mismo una herramienta CASE y cumpliendo con los entregables definidos dentro de la metodología RUP.

7.3 Introducción. Como vimos en la unidad anterior en el desarrollo del modelo de diseño de objetos una parte importante es que las clases descritas y documentadas en las tarjetas CRC se vayan depurando mediante las técnicas de iteración e incrementación hasta que tengan un formato que los programadores puedan utilizar para la implementación correspondiente. Un aspecto importante de esta iteración e incremento es la identificación de los atributos, las operaciones y protocolos por cada clase especificada en las tarjetas.

Las tarjetas CRC servirán de soporte para la empresa en la generación de futuros sistemas y se podrán documentar en nuestra herramienta CASE que en este caso es el EA, recordando que en la parte del análisis se determinan las clases principales junto con los diagramas de interacción de donde se pueden identificar las operaciones a partir de los mensajes existentes entre los elementos que se modelan en estos diagramas.

A continuación describiremos brevemente como podemos documentar en EA los diagramas de clases identificados en el modelado de análisis.

Para completar un diagrama de clases hay dos componentes que son de importancia definir en el diseño que son los atributos, los métodos y sus protocolos, los cuales describiremos a continuación:

Atributos. Los atributos son características de una clase u otro elemento que representan las propiedades o elementos de información interna de ese elemento. Para una Clase Usuario, NombredeUsuario y DireccióndeUsuario pueden ser atributos. Los atributos tienen varias características importantes como el tipo, alcance (visibilidad), estereotipo y otras.

Para crear y modificar atributos de un elemento pertenecientes a una clase, se deberá de modelar primeramente el diagrama de clases respectivo tomando como referencia las tarjetas CRC diseñadas en la práctica anterior. Una vez depurado el diagrama de clases se entrará a la siguiente caja de dialogo mediante un doble click en el botón derecho sobre el elemento (clase) a editar, y desde el menú contextual seleccionar la opción: *atributos*, sobre cada una de las clases que conforman este diagrama. Después de esto, hacer clic en el elemento y presionar [F9] o arrastrar el atributo desde el explorador del proyecto en el elemento correspondiente.

Haciendo lo anterior se abre la ventana de <nombre del elemento> *atributos* y se despliega la caja de dialogo siguiente:

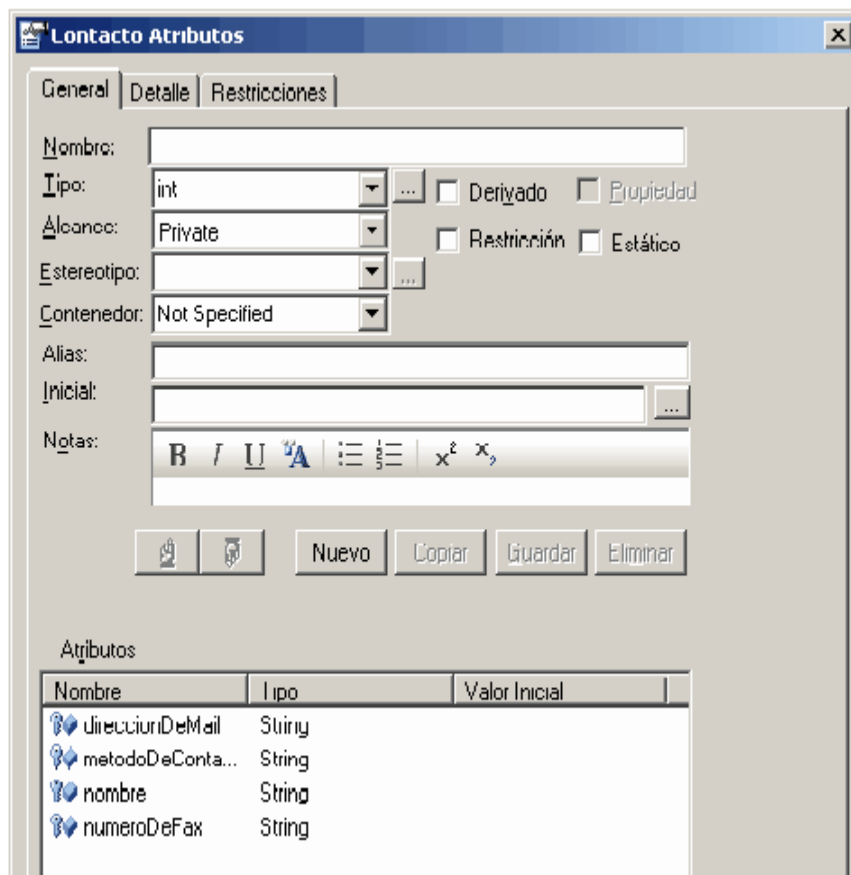


Figura 1. Definición de atributos por clases.

Operaciones. Las operaciones son características de una clase u otro elemento que representan el comportamiento o los servicios que soportan un elemento. Para una clase Cliente, *ActualizarNombreDelCliente* y *ObtenerDomicilioDelCliente* pueden ser Operaciones. Las Operaciones tienen varias características importantes, tales como tipo, alcance (visibilidad), estática, abstracta y notas. Si un elemento soporta operaciones (normalmente clases e interfaces) el menú contextual, que se obtiene haciendo clic con el botón derecho del mouse, contiene el ítem Operaciones. Seleccione esto para abrir la ventana Operaciones. Alternativamente, presione [F10].

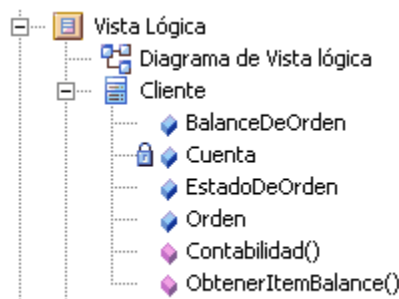
Los elementos con operaciones (normalmente clases) mostrarán estas características en los diagramas en la forma que se muestra en la figura 2, algunas características se mostrarán en forma reducida, por ejemplo *estático* se mostrará como '\$', *abstracto* como '*'



Figura 2. Composición de una clase

Las operaciones en el explorador del proyecto. Las Clases con operaciones tendrán sus características reunidas bajo ellas en el Explorador del Proyecto. Haga clic con el botón derecho sobre una operación y seleccione Propiedades para abrir la ventana Operaciones y editar los detalles de la característica.

Desde la ventana del Explorador del proyecto, puede arrastrar operaciones en los nuevos elementos para proporcionarle la misma operación.



Diseño del Sistema. Como vimos en la introducción de la práctica 6, en el diseño del sistema se toman decisiones importantes como la selección del lenguaje de programación para implementar el sistema, o cuantos de los subsistemas existentes se van a reutilizar en el nuevo sistema de información a desarrollar. Además de esto la portabilidad es otra decisión importante a considerar junto con la asignación de cada componente de software al componente de hardware correspondiente en el cual se va a ejecutar.

Una decisión importante lo constituye sin duda el diseño de la base de datos que se puede considerar como una de las partes del diseño que más relevancia tiene en los sistemas informáticos.

A continuación describiremos la forma de modelado de una base de datos en EA.

Modelado de una Base de Datos en EA. El EA, nos ofrece el soporte para lo siguiente:

- Ingeniería reversa de muchas DBMS populares, incluyendo Oracle 9i y 10g, SQL Server, My SQL, Access y PostgreSQL
- Modela tablas de bases de datos, columnas, claves, claves foráneas y relaciones complejas usando UML y un perfil de modelado de datos integrado.
- Enviar scripts de DDL generados para crear la estructura objetiva de las bases de datos.
- Habilidad para llevar a cabo el modelado de bases de datos, hacer ingeniería reversa desde un rango de DBMS, y generar directamente scripts DDL para crear estructuras de bases de datos.

El Modelado de base de datos y diseño de base de datos no son explícitamente cubiertos por las especificaciones del UML, pero se llevan a cabo en EA usando la opción de *Modelado de datos* de la *Caja de herramientas* del UML de EA y el *Perfil de modelado de datos UML*.

Este perfil provee extensiones fáciles de usar y entender para el UML estándar, mapeando los conceptos de base de datos de tablas y relaciones en los conceptos del UML de clases y asociaciones. Estas extensiones también le permiten modelar las claves de base de datos, disparadores, limitaciones, RI y otras características de base de datos relacionales.

Tablas y columnas. La *estructura* de modelado básica de una base de datos relacional es la *tabla*, que representa un conjunto de registros, o filas, con la misma estructura. El *elemento* de organización básico de una base de datos relacional es la *columna*. Cada artículo individual de datos ingresado en una base de datos relacional se representa por un valor en una columna de una fila en una tabla.

El perfil de modelado de datos UML representa:

- Tablas como *Clases* estereotipadas; es decir, los elementos clase con un *estereotipo* de tabla.
- Columnas como *atributos* estereotipados; es decir, atributos con un *estereotipo* de columna.

Claves de base de datos. Se usan dos tipos de claves para acceder a las tablas: *Claves primarias* y *claves foráneas*. Una clave primaria identifica únicamente un registro de una tabla, mientras una clave foránea accede los datos en alguna otra tabla relacionada a través de su clave primaria.

Una clave primaria consiste de una o más columnas; una clave primaria simple (columna simple) se identifica como el atributo de una operación estereotipada. Una clave primaria compleja (muchas columnas) se define como la operación estereotipada por sí misma.

Una clave foránea es una colección de columnas (atributos) que juntos tienen algún significado operacional (ellos refuerzan una relación a una clave primaria en otra tabla). Las claves foráneas se representan en EA como operaciones con el estereotipo FK; los parámetros de las operaciones se convierten en las columnas involucradas en la clave.

Base de datos soportados. EA soporta la importación de esquema de base de datos desde estas bases de datos:

- DB2
- Firebird/InterBase
- Informix
- Ingres
- InterBase
- MS Access
- MS SQL Server
- MySQL
- Oracle 9i y 10g
- PostgreSQL
- Sybase Adaptive Server Anywhere (Sybase ASA)
- Sybase Adaptive Server Enterprise (Sybase ASE).

Tareas Típicas. Las tareas típicas que puede querer realizar cuando modela o diseña bases de datos incluyen:

- Creando un Diagrama de Modelado de Datos
- Creando una tabla
- Configurando Propiedades de una tabla
- Creando columnas
- Creando Claves Primarias
- Creando Claves Foráneas
- Creando Procedimientos Almacenados
- Creando Vistas
- Creando Índices y Disparadores
- Generando DDL para una Tabla
- Generando DDL para un Paquete
- Convirtiendo Tipo de Dato para una Tabla
- Convirtiendo Tipo de Dato para un Paquete
- Personalizando Tipos de Datos para un DBMS
- Importando un Esquema de Base de Datos desde una Fuente de Dato ODBC

El Perfil de Modelado de Datos del UML no es actualmente un estándar ratificado; sin embargo tiene un soporte de industria amplio y es un método útil para conectar la brecha que hay entre el modelado de base de datos relacional convencional y UML.

Un diagrama de modelado de datos. El siguiente diagrama es un ejemplo de un diagrama de Modelo de Dato. Muestra 3 tablas que están vinculadas en pares de clave primaria a clave foránea con cardinalidad asociada. (FK). Las operaciones pueden agregarse para:

- Validación (check)
- Disparadores (triggers)
- Restricciones
- Índices

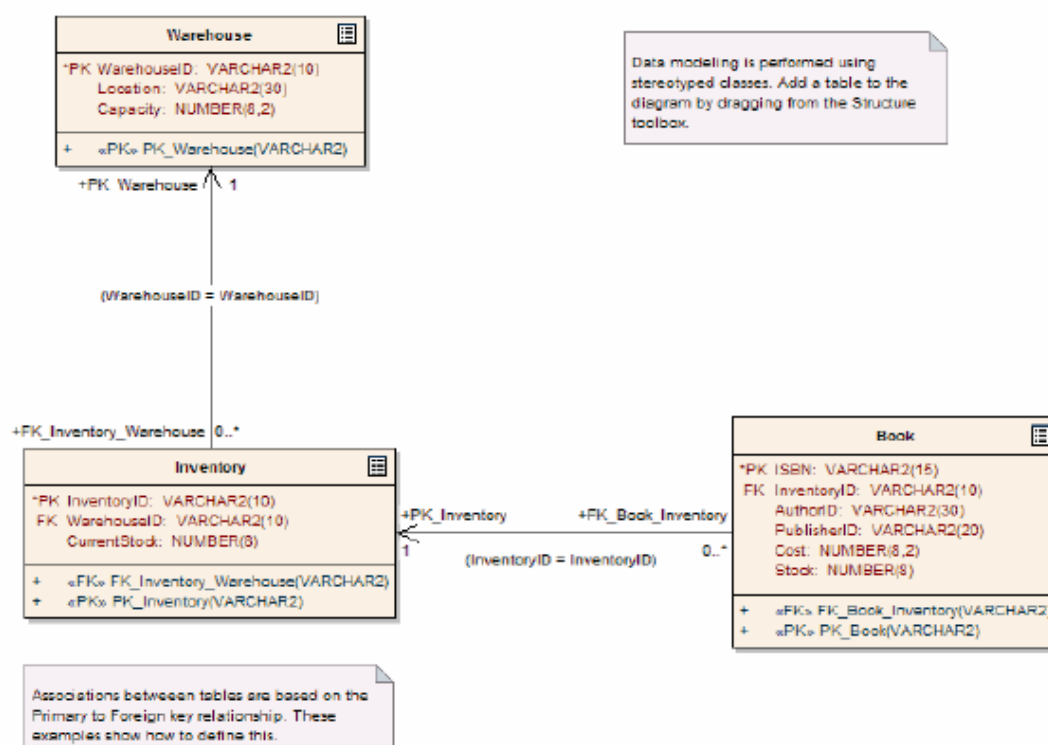


Figura 3. Diagrama de modelado de datos tipo.

Crear una tabla La estructura de modelado básico de una base de datos relacional es una *tabla*. Una tabla representa un conjunto de registros, o filas, con la misma estructura.

El *perfil de modelado de base datos UML* representa una tabla como una clase estereotipada; es decir, un elemento de clase con un estereotipo de tabla aplicada a este. Un ícono de tabla se muestra en la esquina derecha arriba de la imagen cuando esta se muestra en un diagrama de modelado de datos. Para crear una tabla, proceda con los siguientes pasos:

- Seleccione un diagrama.
- Seleccione la opción Más herramientas | extendido | Modelado de datos en la caja de herramientas.
- Haga clic en el elemento Tabla en la lista de elementos, y luego haga clic en un diagrama. El elemento Tabla se muestra en el diagrama.
- Si la ventana de propiedades Clase/Tabla no se abre, haga doble clic en la tabla para abrirla.
- En el campo Nombre, escriba un nombre para la tabla y configure cualquier otra propiedad como es requerido.
- De clic en aceptar.

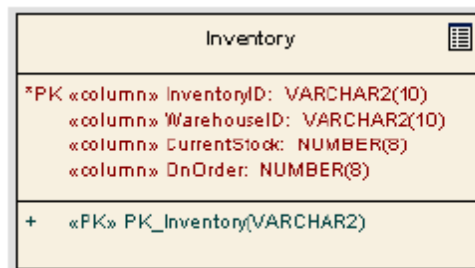


Figura 4. Ejemplo de una tabla en EA.

Configurar propiedades de la tabla. Una vez que creó su tabla puede configurar sus propiedades. La mayoría de las propiedades se pueden configurar desde la ventana de Propiedades, como se describe a continuación. Algunas propiedades necesitarán ser ingresadas como valores etiquetados, y estos son descritos en cualquier lugar, es decir configurando el valor del Propietario de la tabla, y para las configuraciones de bases de datos MySQL, configurando las Opciones de tabla.

Configurar el tipo de base de datos. La propiedad más importante para establecer para una tabla (después del nombre) es el tipo *Base de Datos*. Esto define la lista de Tipos de Datos que están disponibles para definir columnas, y también declara que dialecto del DDL se genera.

Para configurar el tipo de base de dato, proceda con los siguientes pasos:

- Haga doble clic en el elemento tabla en la pestaña del diagrama para abrir la ventana Propiedades.
- Seleccione la pestaña *General*.
- En el campo Base de datos, haga clic en la flecha desplegable y seleccione el tipo de base de dato.
- Haga clic en el botón Aceptar para guardar los cambios. Haciendo clic en la pestaña Detalle de la tabla en esta ventana, puede acceder a la ventana Columnas o la ventana Operaciones o puede Generar DDL para esta tabla.

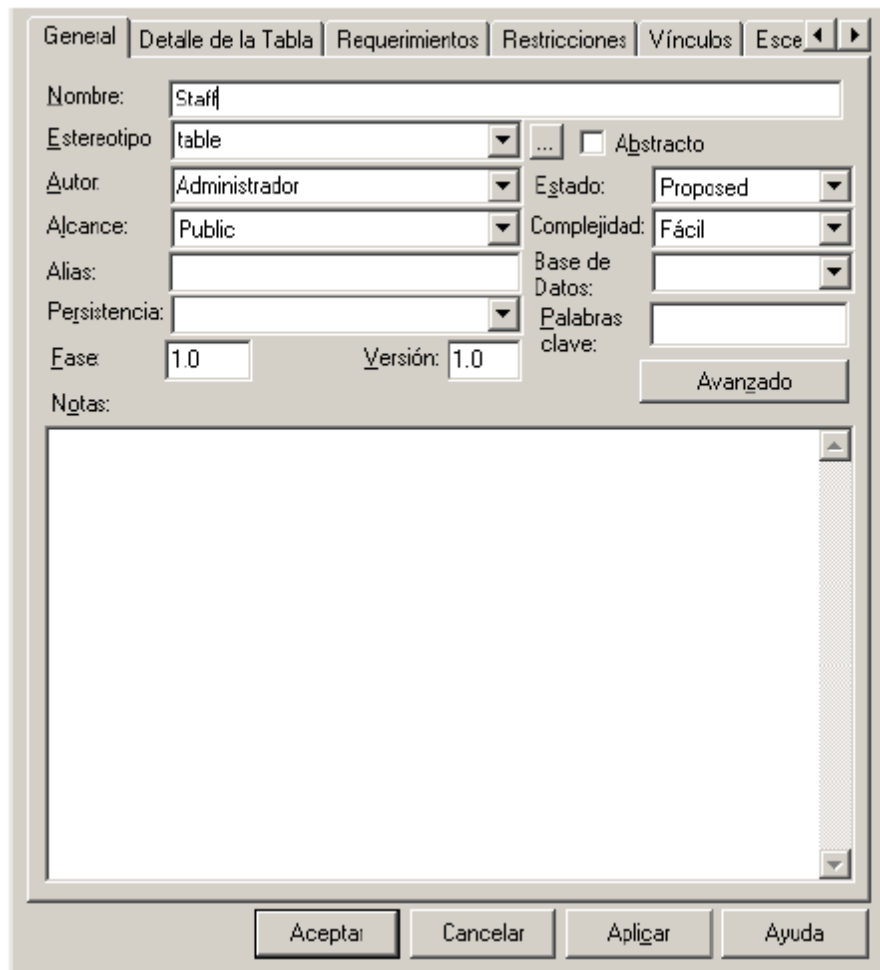


Figura 5. Caja de dialogo usada para configurar el tipo de base de datos en EA

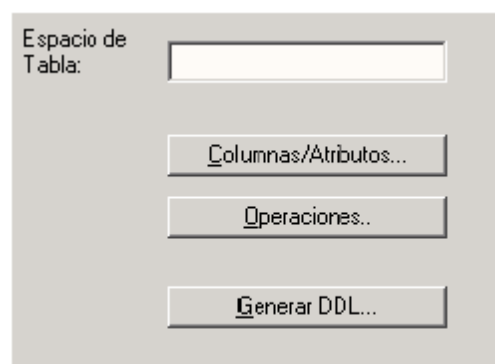


Figura 6. Caja de dialogo para detallar una tabla en EA.

Configurar propietario de la tabla. Para definir el propietario de una tabla, proceda con los siguientes pasos:

- Seleccione la opción de menú Ver|valores etiquetados, se abre la ventana correspondiente mostrada en la figura 7.
- Haga clic en la tabla en un diagrama o en la ventana de Explorador del proyecto. La ventana valores etiquetados ahora muestra las etiquetas para la tabla correspondiente.
- Haga clic en el botón nueva etiqueta, se abre la ventana de Valores etiquetados.
- En el campo **Etiqueta**, escriba el nombre de la etiqueta **Owner**. En el campo **Valor**, escriba un valor para la etiqueta *Owner*.
- Haga clic en el botón **Aceptar** para confirmar la operación. El DDL generado incluye el propietario de la tabla en el script SQL.

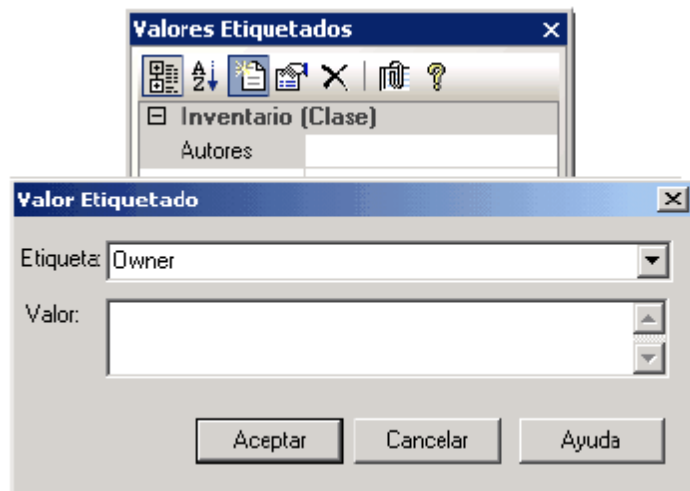
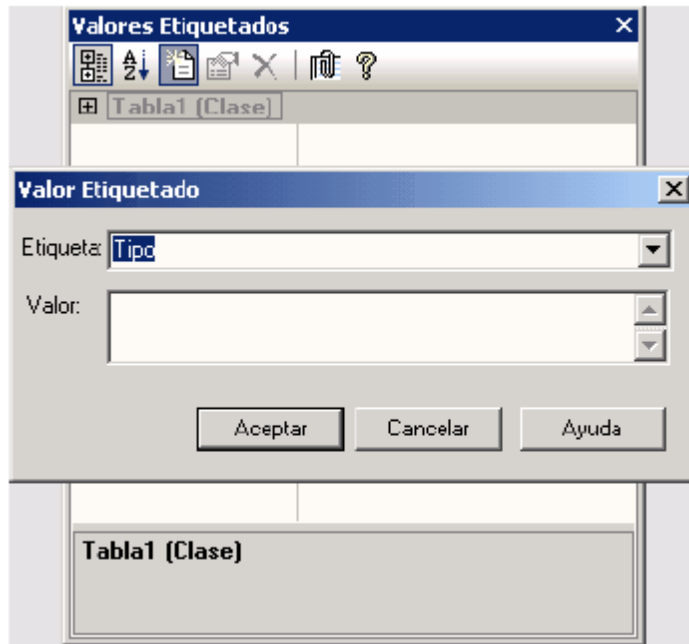


Figura 7. Caja de dialogo para definir valores etiquetados en EA.

Configurar opciones MySQL. Para hacer uso de las claves foráneas en MySQL, debe declarar el tipo de tabla como *InnoDB*. Para hacer esto, proceda con los siguientes pasos:

- Seleccione la opción de menú **Ver | valores etiquetados** o presione **[Ctrl]+[Shift]+[6]**. Se abre la ventana de *Valores etiquetado*.
- Haga clic en la tabla en un diagrama o en la ventana del *Explorador del proyecto*. La ventana *Valores etiquetados* muestra la tabla como seleccionada.
- Haga clic en el botón Nueva etiqueta. Se abre la ventana Valores etiquetados.
- En el campo Etiqueta, ingrese en nombre de la etiqueta Tipo. En el campo Valor, escriba InnoDB como el valor para la etiqueta Tipo.
- Haga clic en el botón Aceptar para confirmar la operación. El DDL generado incluye el tipo de tabla en el script SQL.
- Para permitir versiones posteriores de MySQL, las opciones de la tabla adicional que se pueden agregar de la misma manera incluyen lo siguiente.



Etiqueta	Valor (Ejemplo)
ENGINE	InnoDB
CHARACTER SET	latin1
CHARSET	latin1
COLLATE	latin1_german2_ci

Figura 8. Tipos de etiquetas en EA.

Configurar opciones de la tabla Oracle. Las opciones de la tabla de Oracle9i y 10g, puede configurar las propiedades de la tabla usando los valores etiquetados de la tabla. Proceda con los siguientes pasos:

- Seleccione la opción de menú **Ver | valores etiquetados** o presione **[Ctrl]+[Shift]+[6]**. Se abre la ventana de *Valores etiquetado*.
- Haga clic en la tabla en un diagrama o en la ventana del *Explorador del proyecto*. La ventana *Valores etiquetados* muestra el nombre de la tabla, seleccionada.
- Haga clic en el botón Nueva etiqueta.
- Definir las propiedades de la tabla como se muestra en los ejemplos de la siguiente hoja
- Haga clic en el botón Aceptar para guardar el valor etiquetado.

Todas las propiedades disponibles para la tabla de Oracle se listan a continuación.

The image shows two instances of the 'Valor Etiquetado' (Labeled Value) dialog box. The top dialog has 'Etiqueta: TABLESPACE' and 'Valor: "EA_DATA"PCTFREE 10PCTUSED 40 INTRANS'. The bottom dialog has 'Etiqueta: STORAGE' and 'Valor: (INICIAL 12K NEXT OK)'. Both dialogs have 'Aceptar', 'Cancelar', and 'Ayuda' buttons.

Propiedad	Valor
BUFFER_POOL	DEFAULT
CACHE	NOCACHE
DBVERSION	9.0.111
FREELISTS	1
GRANT OWNER1	SELECT
GRANT OWNER2	DELETE, INSERT, SELECT, UPDATE
INITIAL	65536
INITRANS	1
LOGGING	LOGGING
MAXEXTENTS	2147483645
MAXTRANS	255

Figura 9. Configuración para una tabla en Oracle

Las propiedades definidas para una tabla dada se lista en la ventana *Valores etiquetados*, como se muestra por la siguiente lista típica de valores etiquetados:

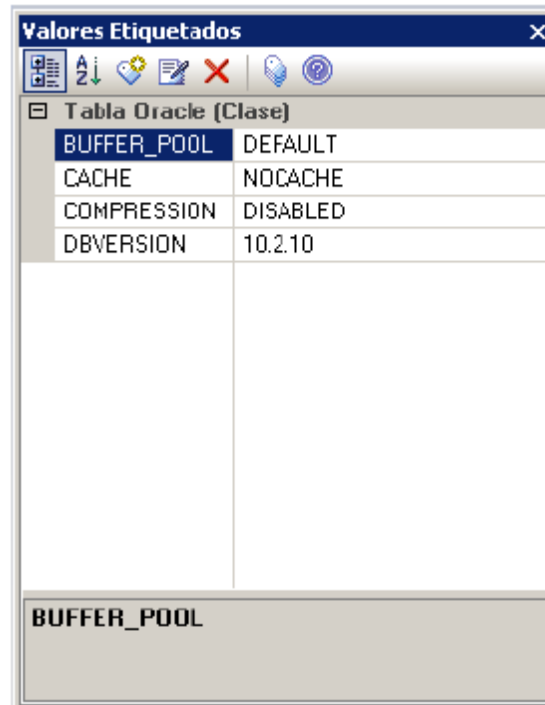


Figura 10. Valores etiquetados

Crear columnas. El elemento de organización básico de una base de datos relacional es la columna. Cada ítem individual de dato ingresado en una base de datos relacional se representa como un valor ingresado en la columna de una fila en la tabla. Las columnas son representadas en el Perfil de Modelado de Datos del UML como un atributo estereotipado, es decir, un atributo con el estereotipo de la *columna*.

Para MySQL, antes de crear columnas primero agregue ENUM y los tipos de datos SET. Seleccione la opción Configuraciones | Tipos de datos de la base de datos y, en la ventana Tipo de datos de base de datos, en el campo Nombre del producto seleccione MySQL. Agregue los tipos de datos ENUM y SET.

Para crear columnas de la tabla, proceda con los siguientes pasos:

- Puede: Hacer clic en el ícono columna en la página Modelado de datos de la caja de herramientas del UML de EA y arrástrelo en el nombre del elemento de la tabla; cuando se le informe de realizar esto, escriba el nombre de la columna y el tipo de dato, luego haga doble clic en el nombre de la columna en la tabla. Hacer clic sobre la tabla en un diagrama para abrir el menú contextual, y seleccione la opción Atributos.
- Se muestra la tabla con n atributos

General

Nombre: Staff

Tipo de datos: INTEGER

Esteriotipo: column

Inicial: 0

Acceso: Público

Notas:

Clave primaria ☒ No nulo ☒ Único ☒

Propiedades de la columna...

Columnas

PK	Nombre	Tipo	No N...	Único

Nuevo Guardar Eliminar

Cerrar Ayuda

Figura 11. Configuración para Mysql

Los siguientes campos para cada columna son opcionales:

Clave primaria - seleccione la casilla si la columna representa la clave primaria para esta tabla.

No Nulo - seleccione la casilla si los valores vacíos se prohíben para esta columna.

Único - seleccione la casilla si se prohíbe que alguno de los dos valores para esta columna sean idénticos.

Inicial - ingrese un valor que se pueda usar como un valor predeterminado para esta columna, si así se requiere.

Acceso - Haga clic en la flecha desplegable y seleccione alcance de Privado, Protegido o Público (este campo esta predeterminado como Público).

Alias. Ingrese un nombre alternativo para el campo (si tiene como objetivo mostrarlo), si hay alguno.

Notas. Ingrese cualquier otra información necesaria para documentar la columna.

Algunos tipos de datos, por ejemplo el tipo de NÚMERO de Oracle, requerirá una precisión y una escala. Estos campos aparecerán donde sea requerido y deberían completarse como es apropiado. Por ejemplo, para Oracle:

crea NÚMERO estableciendo Precisión = 0 y Escala = 0

crea NÚMERO(8) estableciendo Precisión = 8 y Escala = 0

Crea NÚMERO(8,2) estableciendo Precisión = 8 y Escala = 2.

Tenga en cuenta: Los tipos de datos Oracle VARCHAR2(15 CHAR) and VARCHAR2(50 BYTE) se pueden crear agregando la etiqueta LengthType con el valor CHAR o BYTE.

Los tipos de datos MySQL ENUM y SET, en el campo inicial escriba los valores como una lista separada por comas, en el formato: ("uno", "dos", "tres") como predeterminado "tres".

Crear paquetes Oracle. Para crear un paquete Oracle, proceda con los siguientes pasos:

1. Abra el proyecto en la ventana *Explorador del proyecto* y cree un paquete de EA (y, si es requerido, un diagrama de Clase).
2. Agrega un elemento Clase al paquete o al diagrama.
3. Abra la ventana *Propiedades* para el elemento y, en el campo **Estereotipo**, asigne el **Paquete**.
4. Para la especificación del paquete, cree una Operación con el nombre *Especificación* y con ningún tipo de retorno.
5. Abra la ventana *Propiedades* para la Operación *Especificación* y, en la pestaña *Comportamiento*, escriba la especificación del paquete completo en el campo **Código inicial**.
6. Para el cuerpo paquete, cree una Operación con el nombre *Cuerpo* y con ningún tipo de retorno.
7. Abra la ventana *Propiedades* para la operación *Cuerpo* y, en la pestaña *Comportamiento*, escriba el cuerpo del paquete completo en el campo **Código inicial**.

Clave Primaria. Las claves se usan para acceder a las tablas, y existen dos tipos: Claves primarias y Claves foráneas. Una Clave primaria identifica unívocamente a un registro en una tabla, mientras que una Clave foránea accede a la información de otra tabla relacionada por medio de una Clave primaria.

Definir una clave primaria simple. Si una clave primaria consiste de una sola columna, es muy fácil de definir.

1. Haga clic con el botón derecho sobre la tabla en un diagrama para abrir el menú contextual. Seleccione la ventana **Atributos**.
2. En la ventana **Atributos**, seleccione la columna que desea hacer clave primaria.
3. Seleccione la casilla **Clave Primaria** y presione el botón **Guardar**.

Una operación estereotipada se crea automáticamente. Esta es la operación que define la clave primaria para la tabla. Para quitar una clave primaria, simplemente elimine esta operación.

Definir una clave primaria compleja. A menudo, una clave primaria consiste de más de una columna. Por ejemplo, una columna *Apellido* puede no ser única dentro de una tabla, de

esta forma una clave primaria se crea desde las columnas *Apellido*, *Nombre* y *FechaDeNacimiento*.

Realice los siguientes pasos para crear una clave compleja:

1. Siga los pasos para crear una Clave Primaria Simple. No importa que columna elija.
2. Haga clic con el botón derecho sobre la tabla en un diagrama para abrir el menú contextual. Seleccione la opción **Operaciones**.
3. Seleccione la operación de la Clave Primaria (su nombre empieza con **PK_**) y luego seleccione la pestaña de *Columnas*.
4. Para agregar una columna a la clave primaria, haga clic en el botón **Nuevo** seleccione una columna desde la lista *Nombre de Columna*, y luego presione el botón **Guardar**.
5. Haga clic en los botones con manos (hacia abajo o arriba) para cambiar el orden de las columnas en la clave primaria, si es necesario.

Defina una plantilla del nombre de la clave primaria. Para definir una plantilla del nombre para una clave primaria, proceda con los siguientes pasos:

1. Seleccione la opción **Herramientas | Opciones | Ingeniería de código fuente | Editores de código**. Se muestra la página *DDL* de la ventana *Opciones*.

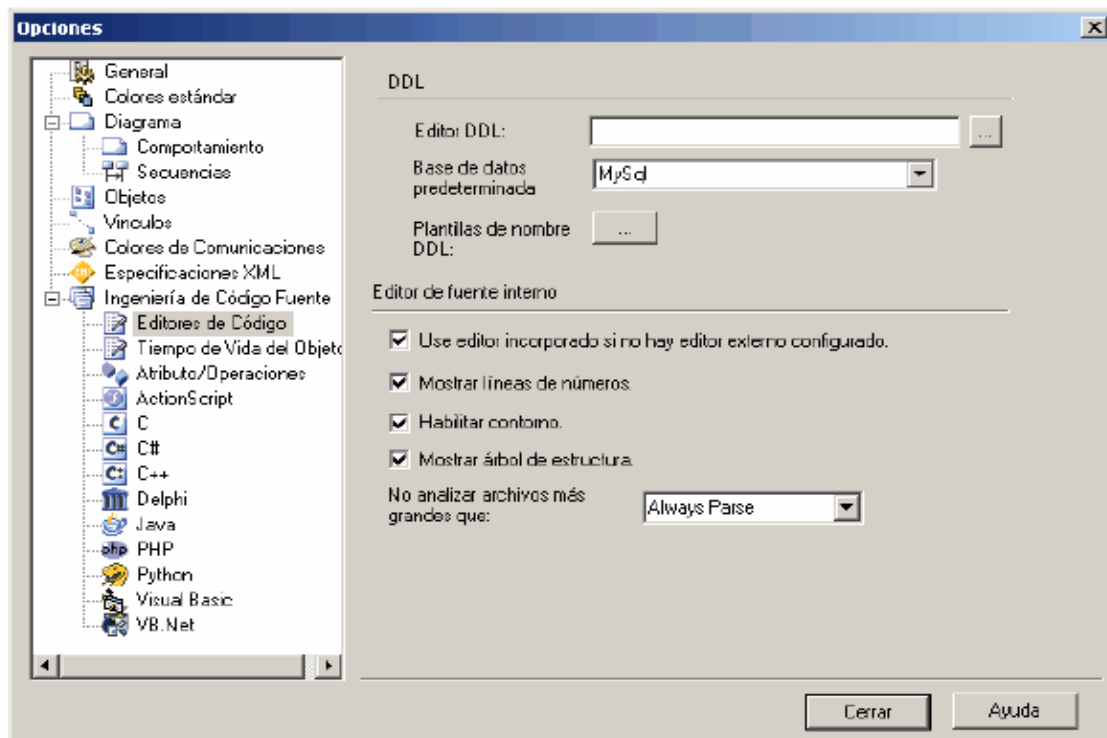


Figura 12. La definición de una plantilla del nombre de la clave primaria

2. Haga clic en el botón Plantilla de nombre DDL. Se muestra la ventana Plantilla de nombre DDL, mostrando las plantillas de nombre predeterminado.

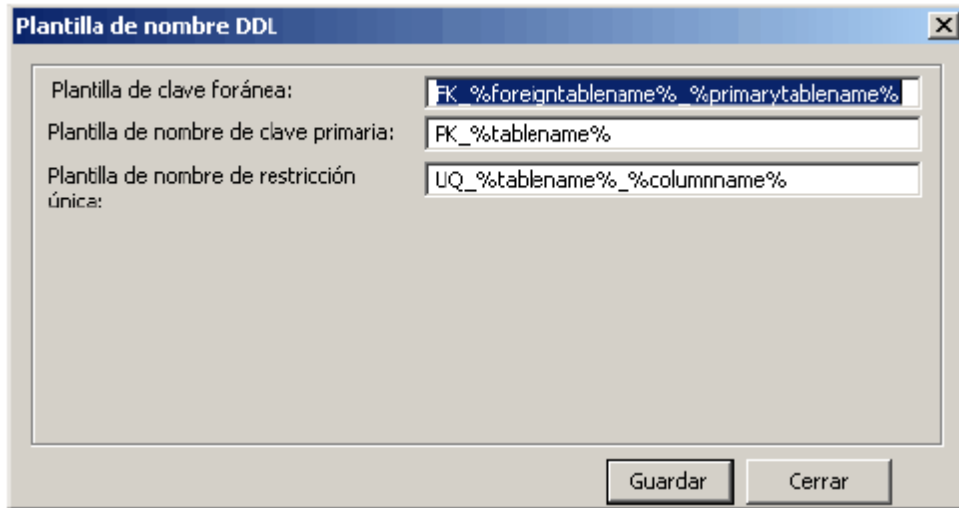


Figura 13. Plantillas del nombre DDL

Clave primaria no agrupada de SQL Server

Para modificar una columna de una tabla, proceda con los siguientes pasos:

1. Haga clic con el botón derecho sobre la tabla en un diagrama para abrir el menú contextual.
2. Seleccione el submenú **Operaciones**. Se abre la tabla *Operaciones de la tabla*.
3. Resalte la operación de la clave primaria y seleccione **Propiedades extendidas**.
4. Seleccionar la caja de herramientas **Clave primaria no agrupada de SQL Server**.
5. Haga clic en el botón **Guardar y cerrar**.

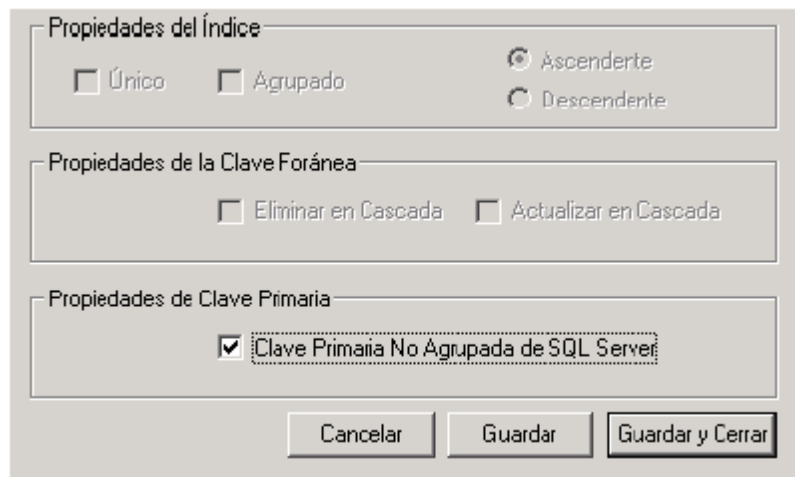


Figura 14. Propiedades de las tablas

Claves foráneas. Las claves se usan para acceder a las tablas: Claves primarias y Claves foráneas. Una Clave primaria identifica únicamente un registro en una tabla, mientras que una Clave foránea accede datos en alguna otra tabla relacionada por su Clave primaria. Las claves foráneas se representan en el UML de Enterprise Architect usando operaciones estereotipadas.

Una Clave foránea es una colección de columnas (atributos) que en conjunto tienen algún significado operacional (fuerzan una relación a una Clave primaria en otra tabla). Una clave foránea se modela como una operación estereotipada con el estereotipo *FK*; los parámetros de la operación pasan a ser las columnas involucradas en la clave.

Es necesario definir una Clave foránea para acceder otra tabla a través de su Clave primaria. Las Claves primarias son una característica de algunos sistemas de administración de base de datos, proporcionando "extras" como verificación de integridad referencial que previene la eliminación de un registro si su valor de clave primaria existe en alguna otra clave foránea de la tabla. Las mismas cosas se pueden obtener programáticamente.

Para crear una clave foránea, haga clic en el vínculo. También puede tener que definir una plantilla de nombre para las claves foráneas.

Crear una clave foránea. Para crear una clave foránea, proceda con los siguientes pasos:

1. Ubique las tablas requeridas en un diagrama o en la ventana del *Explorador del proyecto*.
2. Seleccione un vínculo *Asociación* en la sección *Clase* de la *Caja de herramientas* del UML de EA.
3. Haga clic en la tabla para contener la clave foránea (fuente) y arrastre un conector a la otra tabla (destino).
4. Use el menú contextual del vínculo para mostrar la ventana *Clave foránea*.
5. Si es necesario, edite el nombre predeterminado para la clave foránea.
6. Resalte las columnas involucradas en la relación de clave foránea.
7. Haga clic en el botón **Guardar** para generar automáticamente las operaciones de clave foránea. Ha creado la clave foránea.

Restricción de clave foránea

Nombre:

Origen: Inventario Destino: Almacén

Clave	Columna	Tipo
PK	InventoryID	VARCHAR2
	WarehouseID	VARCHAR2
	CurrentStock	NUMBER
	OnOrder	NUMBER

Clave	Columna	Tipo
PK	WarehouseID	VARCHAR2
	Location	VARCHAR2
	Capacity	NUMBER

Integridad referencial
☐ Eliminar cascada.

Columna	Tipo
WarehouseID	VARCHAR2

Columna	Tipo
WarehouseID	VARCHAR2

Figura 14. Definición de una restricción de clave foránea

Definir una plantilla del nombre de la clave foránea. Para definir la plantilla del nombre para una clave foránea, proceda con los siguientes pasos:

1. Seleccione la opción de menú **Herramientas | Opciones | Ingeniería de código fuente | Editores de código**. Se muestra la página *DDL* de la ventana *Opciones*.
2. Editar o reemplazar la plantilla del nombre en el campo **Plantilla de nombre de clave primaria**.
3. Haga clic en el botón **Guardar**.

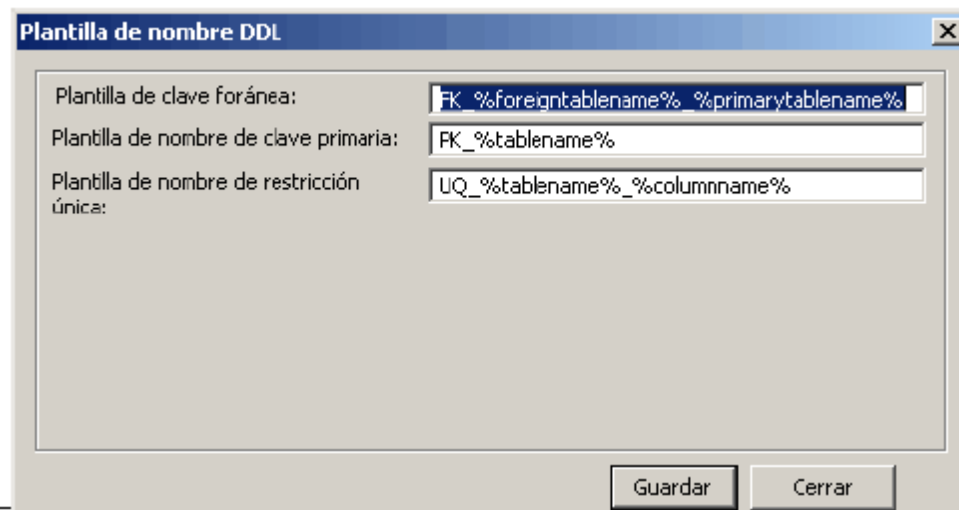
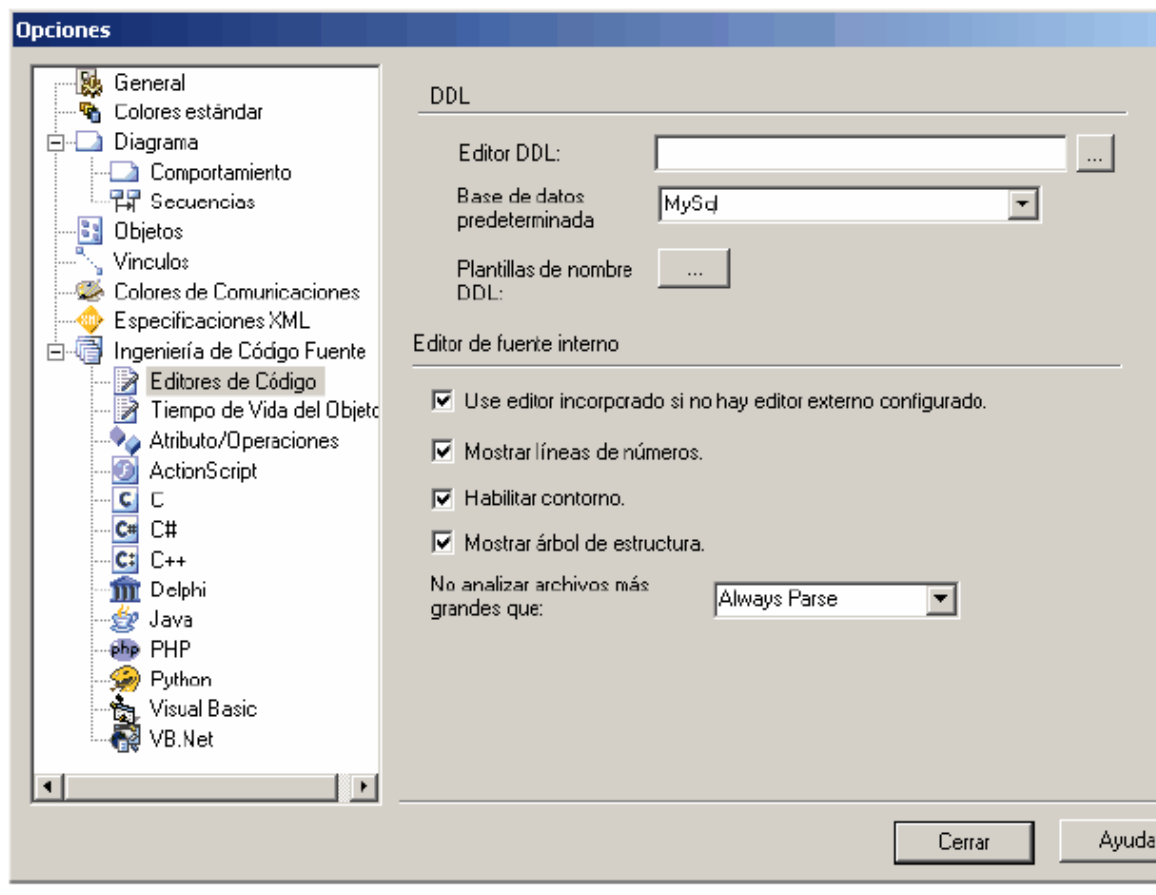


Figura 14. Definición de una Plantilla del nombre de la clave foránea

Procedimientos almacenados. Un procedimiento almacenado es un grupo de sentencias SQL que forman una unidad lógica y realizan una tarea particular. Los procedimientos almacenados son usados para encapsular un conjunto de operaciones o consultas para ejecutar en una base de datos. Los procedimientos almacenados pueden ser compilados y ejecutados con diferentes parámetros o resultados, y ellos pueden tener cualquier combinación de parámetros de entradas y salidas.

Enterprise Architect modela procedimientos almacenados como operaciones de una clase de acuerdo con el Perfil UML para el Modelado de datos. Alternativamente, puede modelar procedimientos almacenados como clases individuales.

Los procedimientos almacenados son actualmente soportados por DB2, SQL Server, Firebird/Interbase, Informix, Ingres, Oracle 9i and 10g, MySQL, PostgreSQL, Sybase Adaptive Server Enterprise (ASE) and Sybase Adaptive Server Anywhere (ASA).

Crear un procedimiento almacenado como una operación de una clase contenedor. Para crear una clase contenedor de Procedimiento almacenado, proceda con los siguientes pasos:

1. Seleccione la opción **Configuraciones | UML**. Se muestra la ventana *Tipos UML*, en la pestaña estereotipos.
2. En el campo **Estereotipo**, escriba los **procedimientos almacenados**.
3. En el campo **Clase base**, escriba **clase**.
4. Haga en el botón **Guardar**, y en el botón **Cerrar**.
5. Seleccione un diagrama apropiado.
6. Seleccione la opción **Más herramientas | UML | Clase** en la caja de herramientas del UML de EA.
7. Haga clic en el elemento **Clase** en la lista de elementos y luego haga clic en el diagrama. Si la ventana Propiedades de la clase no se abre automáticamente, haga doble clic en el elemento.
8. En el campo **Nombre**, escriba un nombre para la clase. Normalmente, este es el nombre de la base de datos.
9. En el campo **Estereotipo**, haga clic en la flecha desplegable y seleccione los **procedimientos almacenados**.
10. Haga clic en el botón **Aceptar** para cerrar la ventana. Ahora tiene un contenedor de procedimientos almacenados.
11. Abra la ventana Propiedades de la clase nuevamente y en el campo **Base de dato** haga clic en la flecha desplegable y seleccione el DBMS destino para modelar. (El campo muestra la base de datos predeterminada si esta se ha especificada con anterioridad).
12. En la ventana Propiedades, seleccione la pestaña Detalles de los procedimientos y haga clic en el botón: **Procedimientos almacenados**. (Alternativamente: Seleccione el contenedor de los procedimientos almacenados y presione **[F10]**, o Seleccione Características Operaciones desde el menú contextual.)
13. Se muestra la ventana Operación <nombre de la clase>.

14. En el campo **Nombre**, escriba el nombre del procedimiento almacenado.
15. En el campo **Tipo de retorno** haga clic en la flecha desplegable y seleccione el tipo de retorno (o use el valor predeterminado **resultset**).
16. En el campo **Esteriotipo**, asegúrese que el valor es **proc**.
17. Haga clic en el botón **Guardar**.
18. Para agregar los parámetros, haga clic en el nombre del procedimiento en el panel de *Operaciones* y haga clic en el botón **Editar parámetros**. Se muestra la ventana *Parámetros*.
19. En el campo **Nombre**, escriba el nombre del parámetro, y en el campo **Tipo** haga clic en la flecha desplegable y seleccione el tipo de parámetro.
20. Si el parámetro es un tipo de largo, agregue el largo después del tipo de parámetro. Por ejemplo, seleccione:

VARCHAR desde la lista desplegable y escriba (5) justo después de este, como el largo.

21. También puede escribir los valores del campo **Tipo** directamente en el campo.
22. Haga clic en el botón **Guardar**, y luego en el botón **Cerrar**. Se muestra la ventana *Operación <nombre de la clase>*.
23. Haga clic en la pestaña **Comportamiento**. En el campo **Código inicial**, escriba el texto del procedimiento.
24. Haga clic en el botón **Guardar**, y luego en el botón **Cerrar**.

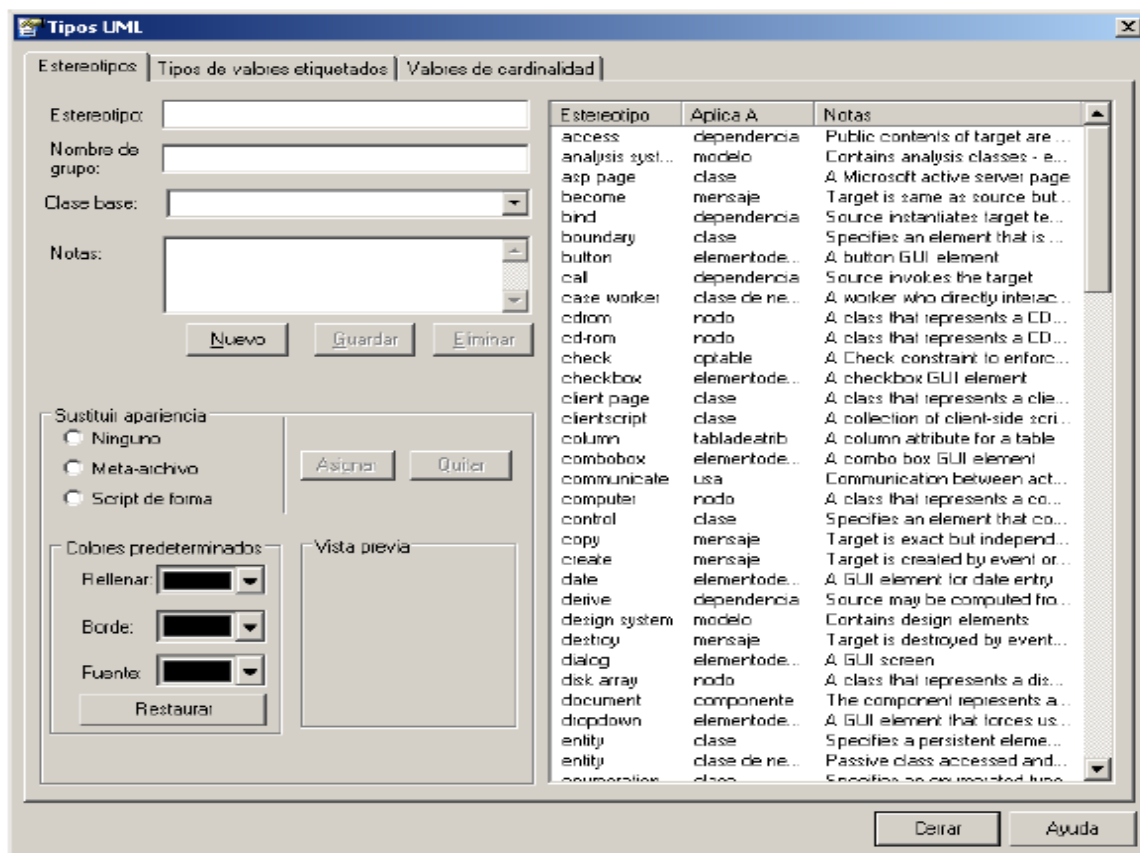


Figura 15. Ventana de tipos UML

Considere la siguiente observación:

Si usa la característica del parámetro como se describe anteriormente, sólo necesita agregar el estado de los procedimientos después cláusula AS.

Si usted prefiere no use la característica del parámetro como se describe anteriormente, inserte **todo** el texto del procedimiento almacenado en el campo **Código inicial**.

En cualquiera de los casos, el texto Crear procedimiento o crear o reemplazar el procedimiento debe ser la primer línea en el campo **Código inicial**.

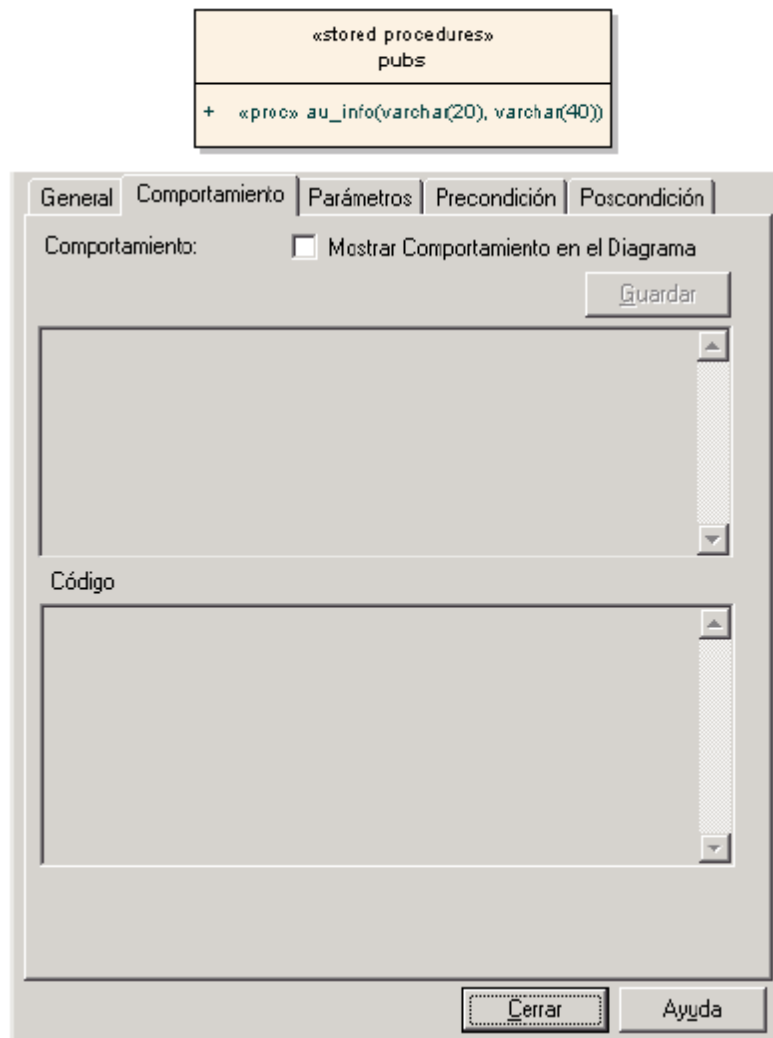


Figura 16. Definición de un procedimiento para una clase contenedor

Crear un procedimiento como una clase individual. Para crear un procedimiento almacenado como una clase individual, proceda con los siguientes pasos:

1. Seleccione la opción Configuraciones | UML. Se muestra la ventana *Tipos UML*, en la pestaña *Estereotipos*.

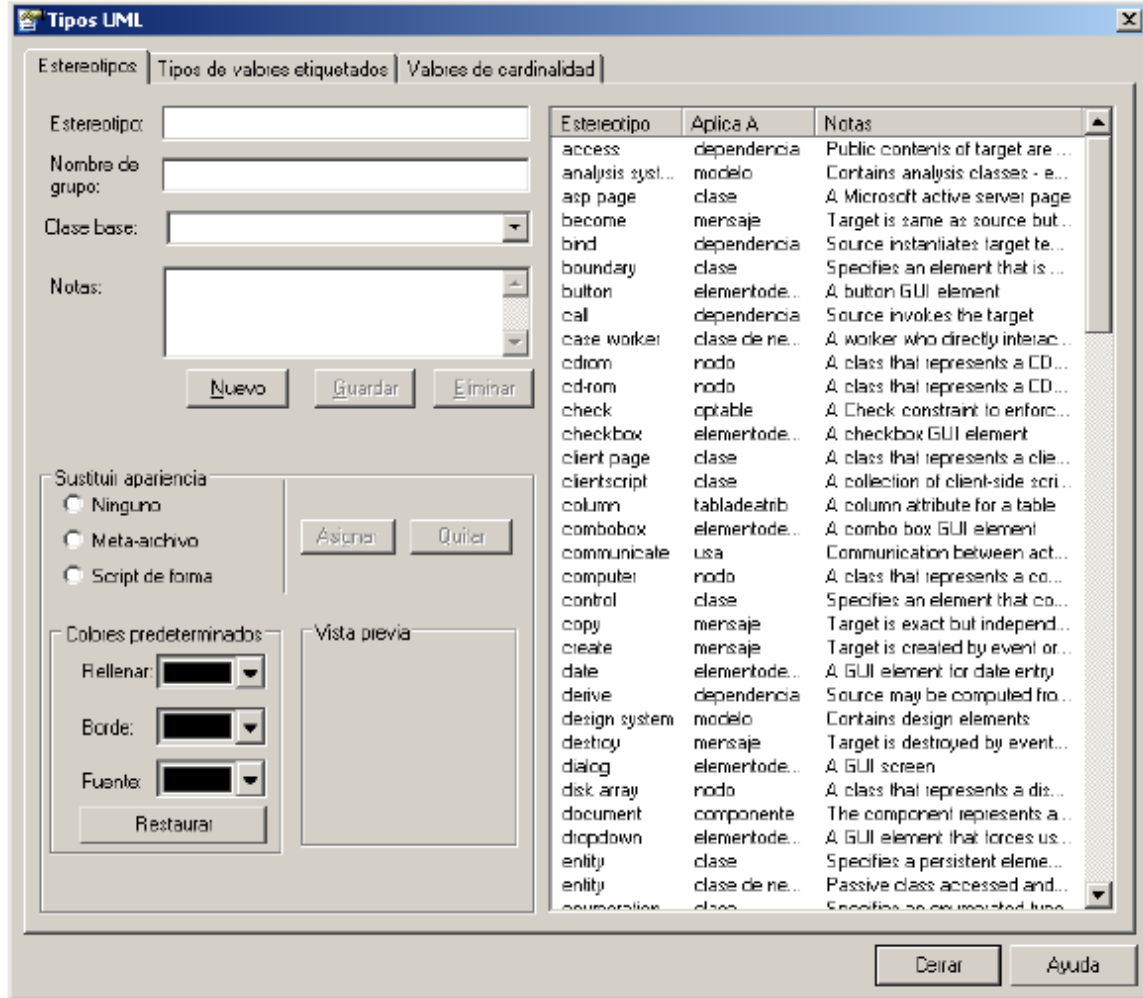


Figura 17. Definición de tipos para una clase individual

2. En el campo **Estereotipo**, escriba **procedimiento**.
3. En el campo **Clase base**, escriba **clase**.
4. Haga clic en el botón **Guardar**, y en el botón **Cerrar**.
5. Seleccione un diagrama apropiado.
6. Seleccione la opción **Más herramientas** | **UML** | **Clase** en la *caja de herramientas* del UML de EA.
7. Haga clic en el elemento **Clase** en la lista de elementos y luego haga clic en el diagrama. Si la ventana *Propiedades de la clase* no se abre automáticamente, haga doble clic en el elemento.
8. En el campo **Nombre**, escriba un nombre para el procedimiento.
9. En el campo **Estereotipo**, haga clic en la flecha desplegable y seleccione **procedimiento**.
10. Haga clic en el botón **Guardar** para cerrar la ventana. Se muestra el elemento nuevo *procedimiento*.

11. Haga doble clic en el elemento procedimiento. Se muestra la ventana *Procedimiento <name>*.

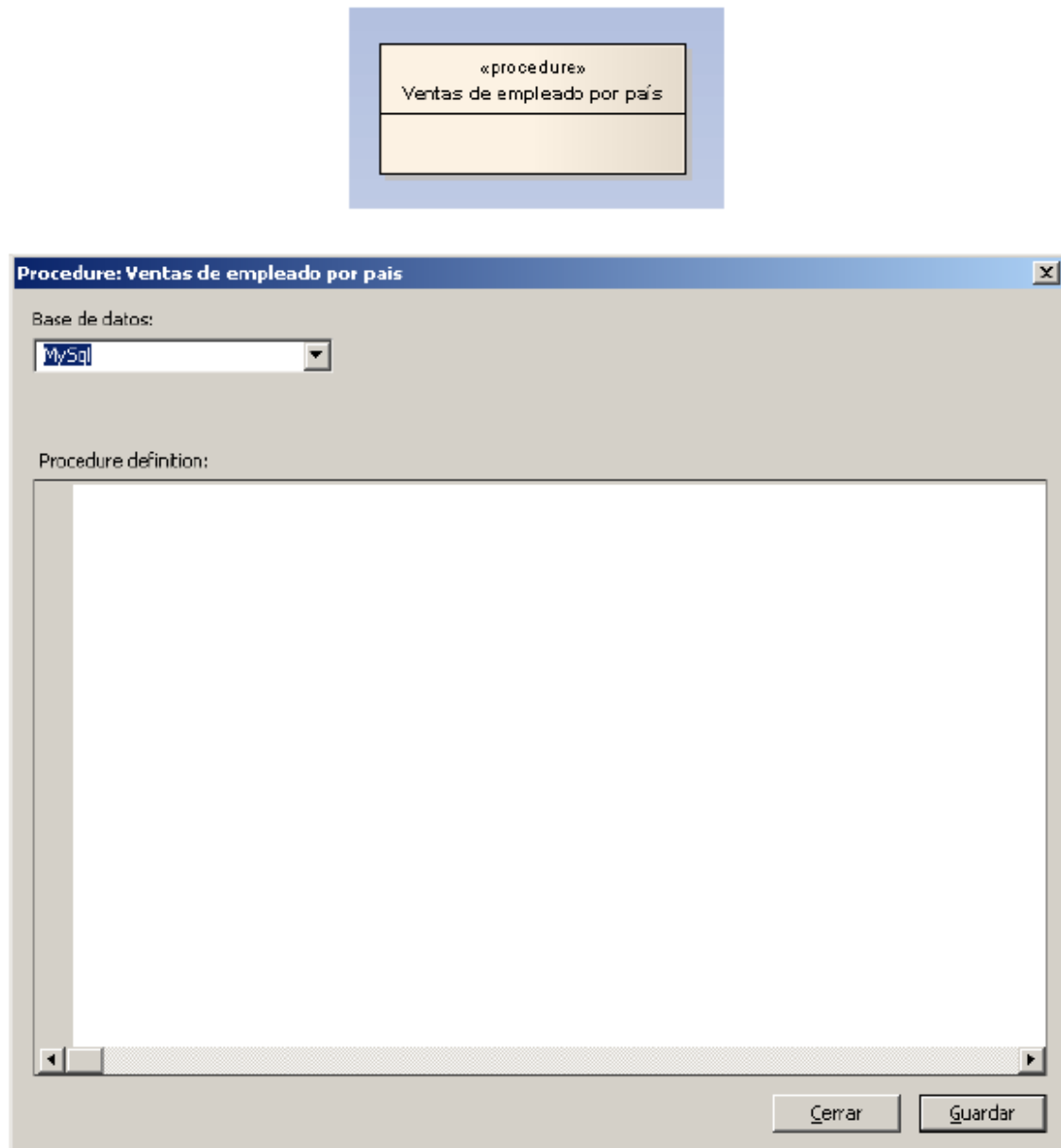


Figura 18. Procedimiento para una clase individual

12. En el campo **Definición del procedimiento**, escriba todo el texto del procedimiento.

13. Haga clic en el botón **Guardar**, y luego el botón **Cerrar**.

Crear un procedimiento de la clase individual. Para crear un procedimiento almacenado como una Clase individual, siga los pasos siguientes, en dos etapas:

1. Seleccione en el menú **Configuraciones | UML**. Se abrirá la ventana Tipos UML, en la pestaña de Estereotipos.
2. En el campo **Estereotipo**, escriba **procedure**.
3. En el campo **Clase Base**, escriba **clase**.
4. Haga clic en el botón **Guardar**, y luego en el botón **Cerrar**.

Crear un elemento procedimiento

1. Abra el diagrama requerido.
2. Seleccione en la Caja de herramientas de EA el menú **Más herramientas | UML | Clase**.
3. Haga clic en el elemento *Clase* de la lista de elementos y luego haga clic en el diagrama. Si la ventana de las Propiedades de la Clase no se abre automáticamente, haga doble clic en el elemento.
4. En el campo **Nombre**, escriba el nombre del procedimiento.
5. En el campo **Estereotipo**, escriba **procedure**.
6. Haga clic en el botón *Aceptar* para cerrar la ventana. Se muestra el nuevo elemento *procedimiento*.

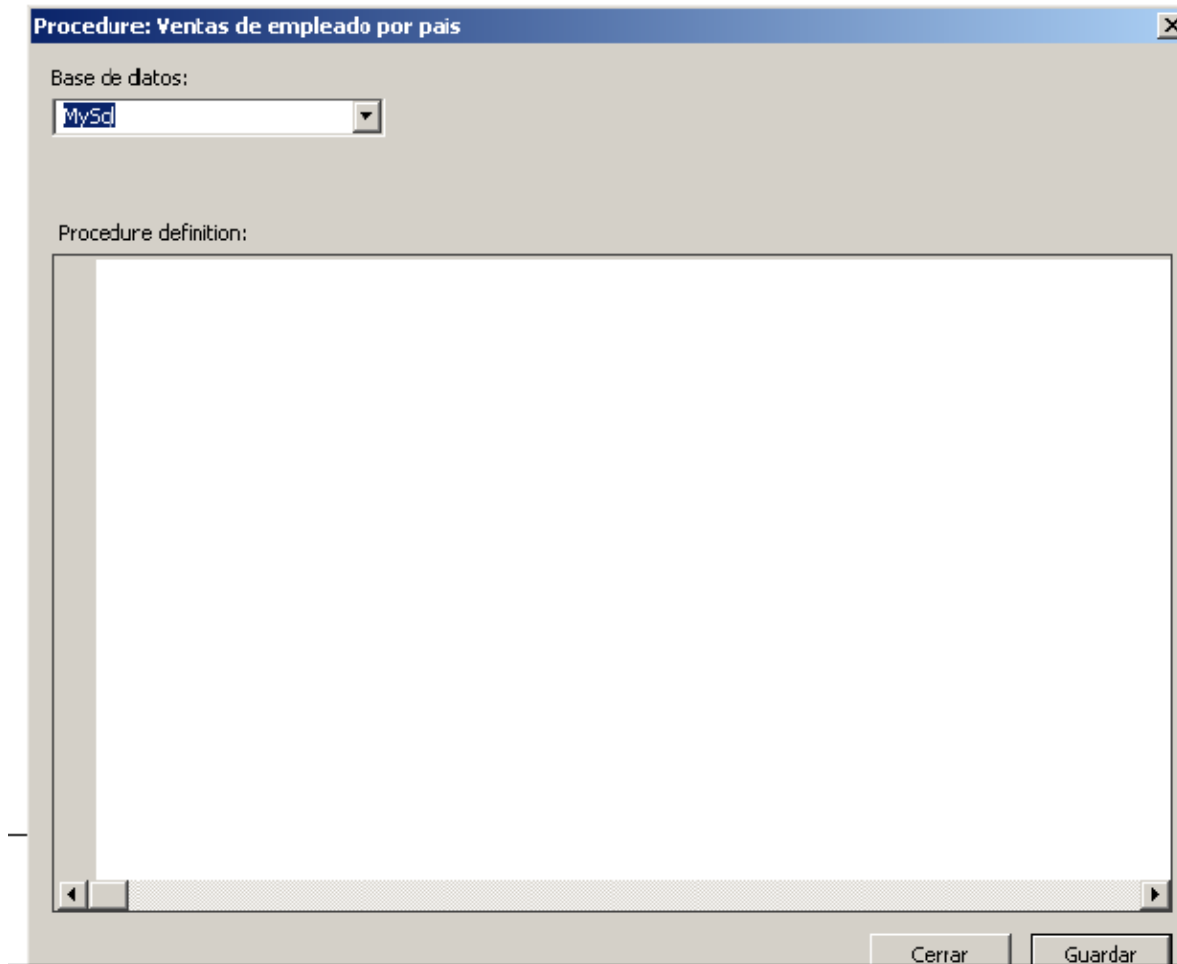


Figura 19. Creación de un elemento procedimiento

7.4 Correlación con el o los temas y subtemas del programa de estudio vigente. La cuarta unidad del programa de la materia se refiere al modelado de diseño de la metodología RUP, en la cual se establece que el alumno aprenda a usar una estrategia de diseño como la que repasamos en práctica anterior. Dentro de esta técnica de diseño basada en responsabilidades se pueden observar dos niveles de diseño como son: diseño de objetos y diseño de sistema que coincide con lo que se establece en el programa de la materia (puntos 4.2 y 4.3) y que igualmente se describe en esta práctica, el diseño de sistema tiene gran valor cuando el sistema en cuestión requiere adaptarse al ambiente de implementación en mayor medida que otros sistemas y un tema muy importante dentro de este diseño lo constituye la base de datos que resulta ser de gran importancia en los sistemas informáticos, de ahí la gran correlación que tiene este tema con el programa vigente.

7.5 Material y equipo necesario. En esta práctica como en las anteriores, seguiremos utilizando EA para completar el diseño de objetos con la elaboración de protocolos, algoritmos y métodos de las clases documentadas en la práctica anterior y el diseño de la base de datos para completar el diseño del sistema.

7.6 Metodología. Para el diseño de objetos se recomienda seguir los siguientes pasos.

- a) De las clases documentadas en la práctica anterior se deberá de completar cada tarjeta con la identificación de los protocolos del sistema, pensando que el protocolo pertenece al conjunto de firmas correspondientes a las distintas responsabilidades de una clase. El objetivo del protocolo deberá ser precisar las responsabilidades hasta obtener métodos que den especial énfasis a las responsabilidades agrupadas en los contratos ofrecidos por las clases.
En general se deberá buscar incrementar la reutilización en el sistema por medio de la simplificación de los protocolos que contengan pocos parámetros, incrementando la posibilidad de descubrir nuevas bases para la herencia. Se debe escoger nombres con cuidado, siguiendo un patrón uniforme en el momento de asignar éstos.
- b) En este inciso se busca detallar un poco más el diseño especificando atributos correspondientes a estructuras de datos y algoritmos que especifiquen la manera de implementar las diversas responsabilidades correspondientes a los múltiples contratos de la aplicación.
En general, los atributos corresponden a aspectos estructurales de las clases, como son los valores (números y textos), junto con las referencias a otros objetos. Los atributos correspondientes a referencias deben agregarse según colaboraciones establecidas durante la práctica anterior. Los atributos que guardan valores son lo último que se especifica en el diseño de objetos.
- c) En este punto deberemos de definir la lógica utilizada para cada operación mediante algoritmos que permitan resolver la responsabilidad correspondiente.

El desarrollo de esta lógica podrá ser tan compleja como la responsabilidad lo sea y tan sencilla como aquellas responsabilidades en donde no se requieran algoritmos, como son las responsabilidades de delegar entre los objetos para consultar o modificar valores de los atributos y dependerán completamente del lenguaje elegido para la implementación final ya que es posible que la lógica de la operación se pueda lograr con un algoritmo en este lenguaje. En este punto el video de la bibliografía 3, nos dará una idea completa de cómo es posible la generación de código java o php cuando el diagrama de clases este bien diseñado.

- d) Como parte del diseño del sistema se deberá de elaborar un diagrama de entidad relación del sistema mediante la creación de las tablas correspondientes, configurando propietarios de la tabla, opciones de la tabla ya sea en MySQL u Oracle según lo defina el catedrático. La definición de las tablas deberá de contener claves primarias simples y complejas así como claves foráneas y sus plantillas correspondientes, finalmente se deberán de definir procedimientos tanto para las clases contenedoras o individuales. Lo anterior de acuerdo a la introducción establecida en esta práctica.

7.7 Sugerencias Didácticas. La recomendación que se le hace al docente es que de las tarjetas elaboradas en la práctica anterior se formule un nuevo diagrama de clases en donde cada clase se complete hasta la generación de algoritmos y por equipos se pueda diseñar la base de datos en cualquiera de las dos plataformas más tradicionalmente utilizadas en los cursos del tecnológico (MySQL y Oracle).

7.8 Reporte de alumnos (Resultados). El alumno deberá de entregar como evidencia en un reporte.

- a) Un proyecto en archivo EAP con los diagramas completados y revisados de casos de uso, clases, interfaces, secuencias y comunicaciones.
- b) Los diagrama entidad relación con todos los detalles establecidos en la metodología.

7.9 Bibliografía

1. Alfredo Weitzenfeld. 2005. Ingeniería de Software Orientada a Objetos con UML, Java e Internet. Thompson
2. Stephen R. Schach. 2005. Análisis y diseño orientado a objetos con UML y el proceso unificado. Mc.Graw Hill.
3. Consulta en youtube: <http://youtu.be/m2aFG7nQ0dU>
4. Spark Systems. 2009. Guía de Usuario de Enterprise Architect 7.1. Solus S.A.

Práctica No.8

8.1 Nombre: Elaboración de un modelo de implementación para un sistema de aplicación real.

8.2 Objetivo: Que el alumno pueda desarrollar un modelo de implementación utilizando un lenguaje de programación orientado a objetos.

8.3 Introducción. El modelo de implementación toma las bases del modelo de diseño para generar una plantilla del código final en el lenguaje que nos permita obtener la herramienta CASE utilizada. Esta traducción debe ser relativamente sencilla y directa, ya que las decisiones mayores han sido tomadas durante las etapas previas.

Durante el modelo de implementación se hace una adaptación al lenguaje de programación y/o la base de datos de acuerdo a la especificación del diseño y según las propiedades del lenguaje de implementación y base de datos.

Aunque el diseño de objetos es bastante independiente del lenguaje actual, todos los lenguajes tendrán sus particularidades, las cuales deberán adecuarse durante la implementación final. La elección del lenguaje influye en el diseño, pero el diseño no debe depender de los detalles del lenguaje. Si se cambia de lenguaje de programación no debe requerirse el re-diseño del sistema.

En general, no se debe comenzar prematuramente a programar, es importante primero completar el proceso de planeación del sistema final desarrollado durante el diseño. Se debe usar guías de programación existentes en la organización. Si no existen, el equipo de software debe crear sus propias guías para decidir aspectos como la definición de los formatos para la asignación de nombres a las variables, el estilo de programación a usar, los métodos de documentación y la documentación en línea. Vale la pena resaltar que aunque existe cierta automatización en el proceso de generación del código final, en su gran mayoría los programadores hacen de forma “manual” la transición final a código fuente.

Como partimos del hecho de que a estas alturas el alumno ya cuente con los principales modelos de la metodología RUP, todos ellos elaborados en EA procederemos en esta introducción a revisar algunos aspectos de importancia en la generación del modelo de implementación como son: la generación de código a partir de estos modelos utilizando EA y las principales directrices de la implementación de código en el modelo vista controlador (MVC), que utilizaremos para este propósito.

Generación de código en el MVC. Como vimos en prácticas anteriores el MVC es un patrón de diseño que considera dividir una aplicación en tres módulos claramente identificables y con funcionalidad bien definida: El Modelo, la Vista y el Controlador, comenzaremos por identificar algunas ventajas y desventajas de este modelo para poder plantear su posible implementación utilizando un lenguaje de vanguardia como es el Java.

Ventajas del MVC.

- La aplicación esta implementada modularmente.
- Sus vistas muestran información actualizada siempre.
- El programador no debe preocuparse de solicitar que las vistas se actualicen, ya que este proceso es realizado automáticamente por el modelo de la aplicación.
- Si se desea hacer una modificación al modelo del dominio, como aumentar métodos o datos contenidos, solo debe modificarse el modelo y las interfaces del mismo con las vistas, no todo el mecanismo de comunicación y de actualización entre modelos.
- Las modificaciones a las vistas no afectan en absoluto a los otros módulos de la aplicación.
- MVC es bastante utilizado en la actualidad en marcos de aplicación orientados a objetos desarrollados para construir aplicaciones de gran tamaño: Java Swing, Apache Struts, Microsoft ASP.NET, las transformaciones XSL o incluso los documentos LATEX siguen este patrón de diseño.
- MVC está demostrando ser un patrón de diseño bien elaborado pues las aplicaciones que lo implementan presentan una extensibilidad y una mantenibilidad únicas comparadas con otras aplicaciones basadas en otros patrones.

Desventajas del MVC.

- El tiempo de desarrollo de una aplicación que implementa el patrón de diseño MVC es mayor, al menos en la primera etapa, que el tiempo de desarrollo de una aplicación que no lo implementa, ya que MVC requiere que el programador implemente una mayor cantidad de clases que en un entorno de desarrollo común.
- Sin embargo, esta desventaja es muy relativa ya que posteriormente, en la etapa de mantenimiento de la aplicación, una aplicación MVC es muchísimo más mantenible, extensible y modificable que una aplicación que no lo implementa.
- MVC requiere la existencia de una arquitectura inicial sobre la que se deben construir clases e interfaces para modificar y comunicar los módulos de una aplicación. Esta arquitectura inicial debe incluir, por lo menos: un mecanismo de eventos para poder proporcionar las notificaciones que genera el modelo de aplicación; una clase Modelo, otra clase Vista y una clase Controlador genéricas que realicen todas las tareas de comunicación, notificación y actualización que serán luego transparentes para el desarrollo de la aplicación.
- MVC es un patrón de diseño orientado a objetos por lo que su implementación es sumamente costosa y difícil en lenguajes que no siguen este paradigma.

La librería de interfaces gráficas de usuario que viene en el SDK5 de Java se llama Java Swing y es una magnífica opción para la implementación de nuestro sistema por lo cual enumeraremos algunas de sus características sobresalientes:

- Implementa bastantes componentes visuales (botones, campos de texto, tablas, barras de menús, arboles, etc.).
- Los componentes provistos por Java Swing son independientes de la plataforma donde se ejecuta la aplicación (contrario a la idea de AWT , otra librería de interfaces de usuario de Java, que utiliza componentes nativos de la interfaz de usuario del sistema operativo donde se ejecuta la aplicación) y por tanto, la portabilidad de las aplicaciones desarrolladas con Swing está garantizada.
- Permite “conectar” y “desconectar” estilos de interfaz de usuario (llamados “look and feels”) que modifican la forma en que se muestra y se comporta toda la interfaz de usuario, así, la misma aplicación puede verse como una aplicación Windows o como una aplicación Motif simplemente conmutando el look and feel en tiempo de ejecución.
- Cumple con el diseño de JavaBeans que hace que los componentes de esta librería puedan ser utilizados en entornos de desarrollo integrados (IDEs) como JBuilder o Eclipse.
- Su arquitectura está profundamente basada en MVC, lo que proporciona un alto grado de extensibilidad y de personalización de los componentes de la librería.

Si bien Swing se basa en el patrón de diseño MVC, presenta algunas diferencias en su implementación: La diferencia más notoria es que el controlador y la vista están implementados como un único elemento denominado “delegado de interfaz de usuario” (“UI delegate”). Cada componente está asociado a un conjunto de datos (modelo del dominio) a través de un modelo de aplicación y del delegado de interfaz. Este delegado ha sido diseñado debido a la incapacidad de programar un controlador genérico con conocimiento de la vista a la que hubiera podido controlar, por tanto, cada componente tiene asociado un delegado de interfaz que a su vez, está asociado a un modelo de aplicación que proporciona toda la información que el delegado requiere y que notifica al componente si algún cambio se ha dado en el modelo del dominio; por tanto, Java Swing implementa el modelo MVC en una implementación por componente

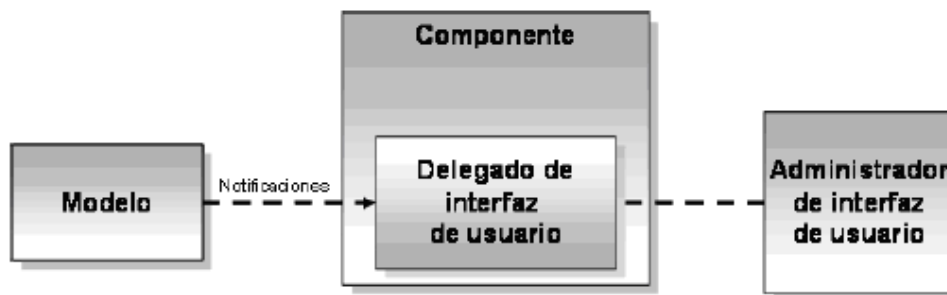


Figura 1. Modelo de implementación e java swing de MVC.

En la figura anterior se observa también la existencia de un elemento denominado “administrador de interfaz de usuario” (“UI manager”); este elemento permite conectar un look and feel específico a cada componente visualizado y almacena también información general a la apariencia de los componentes en su visualización (color de fondo por defecto, tipo de letra por defecto, tipos de bordes, etc.). Todos los componentes de Swing están manejados por el mismo administrador de interfaz de usuario.

Los modelos implementados en Swing están clasificados en dos categorías: Modelos de estado de interfaz de usuario y modelos de datos. Los modelos de estado de interfaz definen el estado visual de un componente, como por ejemplo, el estado de un botón que está presionado o información de los “ítems” seleccionados en determinada lista. Los modelos de datos representan información que está contextualizada en la aplicación. Estos modelos de datos sirven como puente de comunicación entre el modelo del dominio y el delegado de interfaz de un componente.

Modelos y componentes. Como se pudo ver hasta el momento, cada tipo de componente tiene asociado un tipo particular de modelo de aplicación. Swing provee estos tipos de modelo como interfaces (declaraciones de métodos sin implementaciones, por ejemplo el `TableModel`), además proporciona modelos por defecto (`DefaultTableModel`), que proporcionan funcionalidad básica y lista para usarse. La ventaja de que los modelos de aplicación sean implementados como interfaces, es que el desarrollador puede implementar un modelo de aplicación adecuado al modelo del dominio que tiene, sin tener que reestructurar sus datos según requerimientos de Swing.

Veamos todo esto con un ejemplo: Se desea hacer un programa que muestre en una tabla una lista de personas, para lo cual se implementa primeramente una clase `Persona` que tenga nombre, apellido, carnet de identidad y teléfono como atributos:

```
class Persona
{
    protected String nombre;
    protected String apellido;
    protected String carnet;
    protected String telefono;

    public Persona(String aNombre, String aApellido, String aCarnet, String aTelefono)
    {
        nombre = aNombre;
        apellido = aApellido;
        carnet = aCarnet;
        telefono = aTelefono;
    }

    public String getNombre()
    {
        return nombre;
    }

    public String getApellido()
    {
        return apellido;
    }
}
```

```

public String getCarnet()
{
    return carnet;
}
public String getTelefono()
{
    return telefono;
}
}

```

Figura 2. Implementación de la clase Persona.

Luego se construye una clase denominada ListaDePersonas que implementa los métodos para agregar y acceder a las personas a la lista:

```

class ListaDePersonas
{
    protected Vector vectorDePersonas;
    public ListaDePersonas()
    {
        vectorDePersonas = new Vector();
    }
    public void agregarPersona(Persona aPersona)
    {
        vectorDePersonas.add(aPersona);
    }
    public int getNumeroDePersonas()
    {
        return vectorDePersonas.size();
    }
    public Persona getPersona(int aIndex)
    {
        return (Persona) vectorDePersonas.elementAt(aIndex);
    }
}

```

Figura 3. Implementación de la clase ListaDePersona.

Las clases Persona y ListaDePersonas se constituyen en el modelo del dominio de la aplicación de ejemplo. Para que este modelo pueda ser mostrado por un elemento de interfaz, debe construirse un modelo de aplicación, en este caso, la clase PersonasTableModel que se basa en la clase AbstractTableModel que define la funcionalidad básica del modelo de la tabla como se muestra a continuación.

```

class PersonasTableModel extends AbstractTableModel
{
    protected ListaDePersonas listaDePersonas;
    PersonasTableModel(ListaDePersonas aListaDePersonas)
    {
        listaDePersonas = aListaDePersonas;
    }
    public int getRowCount()
    {
        return listaDePersonas.getNumeroDePersonas();
    }
}

```

```

    }
    public int getColumnCount()
    {
        return 4; //hay cuatro atributos en la clase Persona
    }
    public Object getValueAt(int aRow, int aCol)
    {
        Persona persona = listaDePersonas.getPersona(aRow);
        switch (aCol)
        {
            case 0: return persona.getNombre();
            case 1: return persona.getApellido();
            case 2: return persona.getCarnet();
            case 3: return persona.getTelefono();
        }
        return "";
    }
    public String getColumnName(int aCol)
    {
        switch (aCol)
        {
            case 0: return "Nombre";
            case 1: return "Apellido";
            case 2: return "C.I.";
            case 3: return "Telefono";
        }
        return "";
    }
}

```

Figura 4. Implementación de la clase PersonasTableModel.

Los métodos implementados en la clase PersonasTableModel, son métodos definidos en la clase AbstractTableModel que se encargan de proporcionar toda la información que el delegado de la tabla necesita para mostrar esa información. Como se puede ver en el ejemplo, la clase PersonasTableModel sirve como un puente entre el modelo del dominio y el delegado de interfaz de usuario y permite que el componente se adapte a la estructura de datos manejada.

Una vez definidas estas clases, se procede a colocar el componente JTable (asociado al modelo definido anteriormente) en una ventana.

```

import javax.swing.*;
public class PersonasTableFrame extends JFrame
{
    protected JTable table;
    protected ListaDePersonas listaDePersonas;
    public MVCTestFrame()
    {
        setSize(400, 300);
        setTitle("PersonasTableModel");
        initData();
        initUI();
        setVisible(true);
    }
}

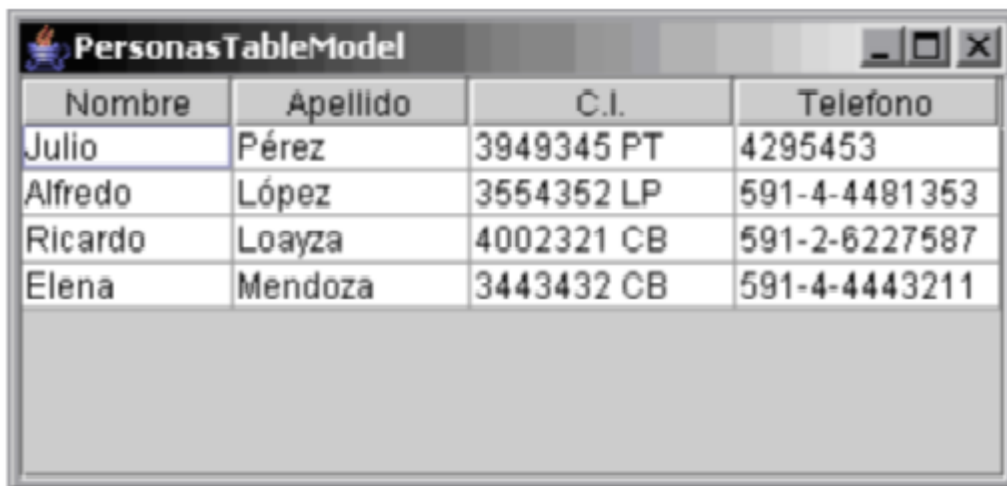
```

```

    }
    protected void initData()
    {
        listaDePersonas = new ListaDePers//Datos ejemplo
        listaDePersonas.agregarPersona(new Persona("Julio", "Pérez",
"3949345 PT", "4295453"));
        listaDePersonas.agregarPersona(new Persona("Alfredo", "López",
"3554352 LP", "591-4-4481353"));
        listaDePersonas.agregarPersona(new Persona("Ricardo", "Loayza",
"4002321 CB", "591-2-6227587"));
        listaDePersonas.agregarPersona(new Persona("Elena",
"Mendoza", "3443432 CB", "591-4-4443211"));
    }
    protected void initUI()
    {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        getContentPane().setLayout(new BorderLayout(5, 5));
        PersonasTableModel tableModel =
        new PersonasTableModel(listaDePersonas);
        table = new JTable(tableModel);
        JScrollPane tableScrollPane =
        new JScrollPane(table);
        getContentPane().add(tableScrollPane, BorderLayout.CENTER);
    }
}

```

Figura 4. Implementación de la ventana principal.



Nombre	Apellido	C.I.	Telefono
Julio	Pérez	3949345 PT	4295453
Alfredo	López	3554352 LP	591-4-4481353
Ricardo	Loayza	4002321 CB	591-2-6227587
Elena	Mendoza	3443432 CB	591-4-4443211

Figura 5. Tabla de PersonasTableModel

Notificaciones. Java Swing maneja el paso de notificaciones por medio de un mecanismo de eventos y de “escuchadores” (“listeners”) de eventos.

Para que una vista se actualice debido al cambio de su modelo asociado, Swing provee dos tipos de notificaciones: Las notificaciones ligeras (“lightweight”) y las notificaciones de estado (“stateful”). Las notificaciones ligeras envían a las vistas un mensaje de “modelo

modificado”. Es la vista la que debe solicitar al modelo más información sobre los cambios realizados. Este tipo de notificaciones es bastante útil cuando la frecuencia de modificaciones al modelo es alta (como en el modelo del JScrollPane que se modifica cada que se arrastra el control del scroll) y es bastante usado en modelos que almacenan estado de interfaz.

El cuadro siguiente muestra los modelos, los listeners y los eventos que funcionan con esta clase de notificaciones:

<i>Modelo</i>	<i>Listener</i>	<i>Evento</i>
BoundedRangeModel	ChangeListener	ChangeEvent
ButtonModel	ChangeListener	ChangeEvent
SingleSelectionModel	ChangeListener	ChangeEvent

Figura 6. Modelos con sus respectivos Listeners y eventos para notificación ligera.

El segmento de código de la figura 7, muestra como el método stateChanged procesa notificaciones cuando el modelo de un botón es modificado:

```
protected void initUI()
{
    boton = new JButton("Evento");
    boton.setBounds(10, 10, 75, 25);
    ButtonModel modelo = boton.getModel();
    modelo.addChangeListener(this); //Listener
    getContentPane().add(boton);
}

//método encargado de escuchar la notificación
public void stateChanged(ChangeEvent aEvent)
{
    contador++;
    mensaje.setText("Modelo cambiado:" + contador + "veces");
}
```

Figura .7 Ejemplo de procesamiento de cambio de estado de modelo.

Las notificaciones de estado proporcionan información mucho más precisa sobre el cambio que se ha dado en el modelo. El cuadro de la figura 8 muestra los modelos, los listeners y los eventos que funcionan con esta clase de notificaciones:

En el segmento de código 9 se ve como el método insertUpdate se encarga de procesar las notificaciones de estado cuando se inserta texto en el documento de un campo de texto:

<i>Modelo</i>	<i>Listener</i>	<i>Evento</i>
ListModel	ListDataListener	ListDataEvent
ListSelectionModel	ListSelectionListener	ListSelectionEvent
ComboBoxModel	ListDataListener	ListDataEvent
TreeModel	TreeModelListener	TreeModelEvent
TreeSelectionModel	TreeSelectionListener	TreeSelectionEvent
TableModel	TableModelListener	TableModelEvent
Document	DocumentListener	DocumentEvent

Figura 8. Modelos con sus respectivos Listeners y eventos para notificaciones de estado

```
protected void initUI()
{
    textField = new JTextField();
    textField.setBounds(10, 10, 175, 22);
    textField.getDocument().addDocumentListener(
        new DocumentListener() //listener
        {
            public void insertUpdate(DocumentEvent aEvent) //evento
            {
                contador++;
                mensaje.setText("Modelo cambiado: " + contador + " veces");
            }
            public void changedUpdate(DocumentEvent aEvent) { }
            public void removeUpdate(DocumentEvent aEvent) { }
        });
    getContentPane().add(textField);
}
```

Figura 9: Ejemplo de procesamiento de notificación de cambio de estado.

Ventajas. La implementación de Swing del patrón de diseño MVC presenta ventajas y desventajas como las que citamos a continuación:

- Permite la creación de interfaces de usuario de una manera sencilla y rápida, permitiendo el manejo del patrón MVC pero ocultando los detalles de su implementación.
- El mecanismo de eventos de Java se adapta perfectamente al mecanismo de notificaciones de MVC.
- Al estar los modelos separados de la vista, las posibilidades de extensión de la librería y de personalización de componentes ya existentes son enormes.
- Permite al usuario crear sus propias estructuras de datos y adaptar la interfaz de usuario a ellas y no a la inversa, como sucede con librerías ya implementadas.

Desventajas.

- La principal desventaja de esta implementación es que, aunque la arquitectura de Java Swing lo permite, la personalización y extensión de los modelos de algunos componentes (especialmente tablas y arboles), es bastante difícil.

- Los modelos de estado y los modelos de datos de Java Swing no están claramente diferenciados en la implementación y, si bien el modelo de datos es la implementación del modelo de la aplicación, el modelo de estado está muy alejado de la definición de modelo que MVC introdujo.
- El controlador definido por MVC es casi inexistente en Java Swing y su trabajo puede confundirse con el del administrador de interfaz de usuario.

Ingeniería de Código en EA. La ventaja de trabajar en ambientes como el de EA, es que podemos desarrollar tanto Ingeniería Inversa (generación de código a partir de modelos) o Ingeniería Directa (generación de código y de ahí los modelos correspondientes), a continuación haremos un breve repaso de estos temas importantes.

En EA, el modelado depende y soporta ingeniería de código - genera y actualiza código desde el modelo, y crea y actualiza modelos desde el código, las funciones del EA en este sentido se resumen como sigue:

- Ingeniería directa, ingeniería reversa, entorno y sincronización de código en un rango de lenguajes de programación.
- Depurar y perfilar código.
- Modelado y generación de código para Tecnologías XML.
- Llevar a cabo modelado de base de datos y diseño de base de datos para un rango de sistemas de administración de base de datos.
- Convertir componentes de modelos desde un dominio a otro con transformaciones
- Arquitectura Dirigida por Modelo (MDA).

Generar Código Fuente. Generando Código Fuente (Ingeniería directa) toma en cuenta el UML y los elementos del modelo de interfaz y crea un código fuente equivalente para elaboraciones y compilaciones futuras, al hacer esto lo que la herramienta case simplifica es el trabajo común de codificar clases, atributos y métodos, y se logra una mejor simetría entre el modelo, y el código generado.

El Código es generado desde los elementos del modelo de interfaz o clase, de modo que debe crear la clase requerida y elementos de interfaz desde el cual los generará. Agregar atributos (los que se convierten en variables) y operaciones (los que se convierten en métodos).

Antes de que genere código, debe asegurar las configuraciones predeterminadas para que la generación de código iguale sus requerimientos. Las configuraciones de generación predeterminada se ubican en la sección de *Ingeniería de Código Fuente* en la ventana de *Opciones* (seleccione la opción de menú Herramientas | Opciones | Ingeniería de código fuente), con esto se logra igualar su lenguaje con las preferencias requeridas.

Las Preferencias que pueden ser definidas incluyen constructores y destructores predeterminados, métodos para interfaces y las opciones de Unicode para lenguajes creados.

Además de las configuraciones predeterminadas para generar código, EA soporta los siguientes lenguajes de código con sus propias opciones de generación de código específicas:

- Actionscript
- C++
- C# (para versiones .NET 1.1 y .NET 2.0)
- Delphi (estándar, mas extensiones C++ administradas por .NET)
- Java (incluyendo Java 1.5, aspectos y genéricos)
- PHP
- Python
- Visual Basic
- Visual Basic .NET

El Esquema de Plantilla de Código (EPC) permite la personalización de la manera en la que EA genera código fuente y también permite la generación de lenguajes que no son soportados específicamente por EA.

Además de poder generar código para una clase individual, también puede seleccionar un grupo de clases para una generación de código por lote. Cuando haga esto, aceptara todas las opciones de generación de código por defecto para cada clase en el conjunto.

Para generar múltiples clases. Se recomienda seguir los siguientes pasos:

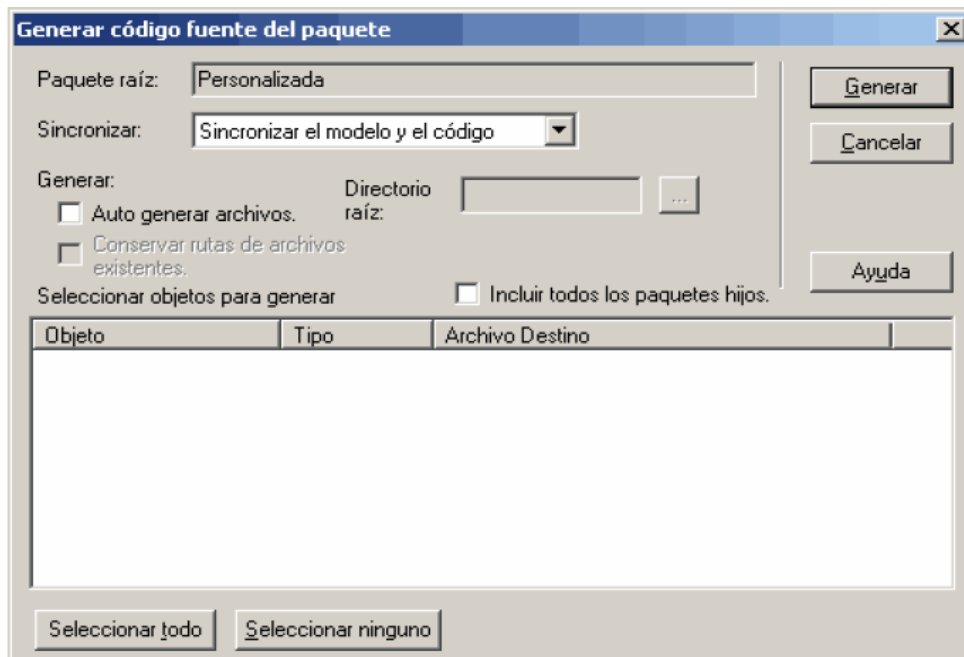
1. Seleccione un grupo de clases o interfaces en un diagrama.
2. Haga clic con el botón derecho sobre un elemento del grupo para abrir el menú contextual.
3. Seleccione la opción Generación de código | Generar los elementos seleccionados. Se abre la ventana *Generación de Código*, mostrando el estado del proceso mientras se ejecuta.
4. Si no se especificó aún un nombre de salida para la clase/interfaz, EA pedirá un nombre adecuado a medida que proceda la generación.

Generar un Paquete Además de generar código fuente de clases únicas y de grupos de clases, también puede generar código de un paquete. Esta característica provee las opciones para generar paquetes hijos en forma recursiva y generar automáticamente estructuras de directorio basadas en la jerarquía de paquetes. Esto permite generar una rama completa de su modelo del proyecto de una sola vez. Esto se hace con los pasos siguientes:

1. En el *Explorador de Proyectos*, haga clic con el botón derecho sobre el paquete para el que quiera generar código.

2. Seleccione: Ingeniería de Código | Generar Código Fuente desde las opciones del menú. El diálogo *Generar Paquete de Código Fuente* se mostrará.
3. Seleccione la opción de Sincronizar apropiada:
 - Sincronizar el modelo y el código: Las clases con archivos existentes se sincronizarán hacia adelante con ese archivo. Las clases se generarán en el archivo que se muestra, si existe.
 - Sobrescribir código: Todos los archivos destino seleccionados se sobrescribirán (se generan hacia adelante).
 - No generar: Se generarán sólo las clases seleccionadas que no tengan un archivo existente. Todas las otras clases se ignorarán.
4. Resalte las clases que desea generar. No seleccione todas las que no desee generar.
5. Si le requiere que EA genere los nombres de los directorios y de los archivos automáticamente basados en la jerarquía de paquetes, marque la opción Auto Generar Archivos. Se le pedirá que seleccione un directorio raíz bajo el que se generarán los directorios de código fuente. La característica de AutoGenerar ignora por defecto todas las rutas que ya existen asociadas con una clase. Puede cambiar este comportamiento marcando la opción Conservar Rutas de Archivos Existentes.
6. Si desea incluir todos los subpaquetes en la salida, marque la opción Incluir todos los Paquetes Hijos. Cuando haya hecho su selección, presione Generar para iniciar el proceso.
7. Haga clic en el botón Generar para empezar a generar código.

A medida que proceda la generación de código se le informará del progreso. Si una clase requiere un nombre de archivo de salida se le pedirá que ingrese uno en el momento apropiado.



Campo	Descripción
Paquete Raíz	El nombre del paquete a generarse
Sincronizar	Estas opciones especifican cómo se deberían generar los archivos existentes
Auto Generar Archivos	Especifica si EA debería generar automáticamente los nombres de archivos y de directorios, basados en la jerarquía de paquetes
Directorio Raíz	Si se Auto Generar Archivos , especifica la ruta bajo el que se creará la estructura de directorios generada
Conservar Rutas de Archivos Existentes	Si se Auto Generar Archivos , especifica si hay que usar los caminos de archivos existentes asociados con las clases. Si no está seleccionado, EA genera las clases a los caminos determinados automáticamente, sin considerar si ya hay archivos de código fuente asociados con las clases.
Incluir todos los Paquetes Hijos	Si está marcado, se incluyen en la lista todas las clases de todos los subpaquetes del paquete destino. Esta opción facilita la generación recursiva de un paquete dado y sus subpaquetes.
Seleccionar Objetos para Generar	Lista todas las clases que están disponibles para la generación bajo los paquetes destino. Sólo se generarán las clases seleccionadas (resaltadas). Las clases se listan con su archivo de código fuente destino.

Botón	Descripción
Seleccionar Todo	Marca todas las clases de la lista como seleccionadas
Seleccionar Ninguno	Marca todas las clases de la lista como no seleccionadas
Generar	Comienza la generación de todas las clases seleccionadas.
Cancelar	Sale de la ventana del Editor de Plantillas de Código . No se generará ninguna clase

Figura 10. La generación de código de un paquete y las opciones principales.

8.4 Correlación con el o los temas y subtemas del programa de estudio vigente. La quinta unidad del programa de la materia se refiere al modelado de implementación de la metodología RUP, en la cual se establece que el alumno aprenda a hacer la implementación del diseño elaborado anteriormente por medio de la misma herramienta case utilizada en las practicas anteriores o bien por medio del uso de un lenguaje de programación que le permita completar el trabajo de implementación, de ahí que en esta unidad hicimos alusión

a estos dos métodos de implementación de nuestro modelo de diseño por lo que se tiene una gran correlación de lo visto en esta práctica con el programa de la materia.

8.5 Material y equipo necesario. En esta práctica como en las anteriores, utilizaremos el EA para completar la elaboración del código a partir de los modelos elaborados anteriormente, aunque también es importante contar con un IDE tipo NetBeans y un compilador de Java para la elaboración del código final.

8.6 Metodología. Para la implementación del diseño se recomienda seguir los siguientes pasos.

1. Tomar cada paquete de clases generado en la parte del diseño correspondiente a cada uno de los subsistemas que conformarán mi sistema principal y seguir los pasos establecidos en la introducción correspondiente para generar las plantillas de código en java, guardarlos en la carpeta: código subsistema1, 2, etc. Seleccionar la opción de sincronizar modelo y código.
2. Abrir el archivo texto de la carpeta y revisar el código para completarlo con el IDE que haya elegido el profesor.
3. Completar la implementación del sistema utilizando el IDE seleccionado para obtener el código del sistema.

8.7 Sugerencias Didácticas. La recomendación es aprovechar la herramienta del EA para que a partir de los modelos generados en la misma se pueda generar una plantilla de código en Java, la cual servirá para la implementación final del código.

Nuevamente el trabajo puede dividirse por subsistemas de forma que cada alumno aprenda lo respectivo a la generación de código a partir de los modelos que generó en las prácticas anteriores (ingeniería directa).

En virtud del tamaño de los sistemas que pueden tenerse como proyectos de clase y que bien sabemos que el desarrollo de todo sistema es un proceso que puede tomar mucho más tiempo que un semestre, se recomienda que el alumno llegue hasta la generación de código automática para todas sus clases y complete este trabajo para el semestre próximo con la finalidad de que mediante la utilización de un IDE adecuado pueda completar un prototipo funcional dentro de la materia de gestión de proyectos de software como es el objetivo de esta cadena de materias que son parte de la retícula de la carrera.

8.8 Reporte de alumnos (Resultados). El reporte de los alumnos en esta práctica consistirá en lo siguiente:

- a) Archivos texto con plantillas de código en Java por cada paquete del sistema.
- b) Listado impreso de estos archivos texto.

8.9 Bibliografía

1. Alfredo Weitzenfeld. 2005. Ingeniería de Software Orientada a Objetos con UML, Java e Internet. Thompson
2. Consulta en YouTube: <http://youtu.be/m2aFG7nQ0dU>.
3. Spark Systems. 2009. Guía de Usuario de Enterprise Architect 7.1. Solus S.A.