Ambientes de Desarrollo de Software Basado en Componentes

José Alberto Vela Dávila

Dra. Perla Velasco Elizondo

Ambientes de Desarrollo de Software Basado en Componentes

Reporte Técnico de Investigación
que para obtener el grado de Maestro en Ingeniería de Software presenta

José Alberto Vela Dávila

Director: Dra. Perla Velasco Elizondo

Centro de Investigación en Matemáticas

27 de Julio de 2009

Versión 2.0

Control de Versiones de la Plantilla

Nombre del Archivo	Versión	Fecha	Autor	Comentarios
RT_AmbientesdeDesarrolloCBD_v0.1_JAVD	0.1	2008/07/10	JAVD	Versión Inicial
RT_AmbientesdeDesarrolloCBD_v0.2_JAVD	0.2	2008/07/15	JAVD	Actualización del documento con base en los comentarios de Perla Velasco
RT_AmbientesdeDesarrolloCBD_v0.3_JAVD	0.3	2008/07/24	JAVD	Actualización del documento con base en los comentarios de Perla Velasco
RT_AmbientesdeDesarrolloCBD_v0.4_JAVD	0.4	2008/07/30	JAVD	Actualización del documento con base en los comentarios de Perla Velasco
RT_AmbientesdeDesarrolloCBD_v0.4_JAVD	0.5	2008/07/31	JAVD	Actualización del documento con base en los comentarios de Perla Velasco
RT_AmbientesdeDesarrolloCBD_v1.0_JAVD	1.0	2008/08/01	JAVD	Versión final de Estudio Guiado 1
RT_AmbientesdeDesarrolloCBD_v1.0_JAVD	2.0	2009/07/23	JAVD	Versión final Estudio Guiado 2

Índice

INDICE DE FIGURAS

Índice de Tablas

José Alberto Vela Dávila

Centro de Investigación en Matemáticas CIMAT, 2009

Resumen

En este documento hablaremos acerca de los ambientes de desarrollo de software, IDE por sus siglas en inglés, los cuales son herramientas que ayudan a los programadores a desarrollar software; abordaremos específicamente aquellos ambientes de desarrollo que se enfocan u orientan al paradigma de desarrollo de software basado en componentes, CBSE por sus siglas en inglés, o CBD como también se le conoce.

Analizaremos estos ambientes, de acuerdo a algunas características básicas que de acuerdo a ciertos autores deberían contener en la etapa de composición, es decir al momento de ensamblar una aplicación utilizando diversos componentes.

Al final se mostrará un cuadro comparativo que muestra cuales de las características que seleccionamos, están contenidas en los ambientes de desarrollo estudiados.

Palabras Clave

Componente, Ambiente de Desarrollo, Repositorio, Adaptación, Composición, Desarrollo, Reutilización. Paradigma.

Capítulo 1. Introducción

Los ambientes de desarrollo de software, IDE por sus siglas en inglés, son herramientas que ayudan a los programadores a desarrollar software sobre entornos más amigables, Es decir aquellos en los que el programador puede acceder con el menor esfuerzo a diferentes recursos como, editores, compiladores, etc.

En este documento hablaremos específicamente de aquellos ambientes de desarrollo que se enfocan u orientan al paradigma de desarrollo de software basado en componentes, CBSE por sus siglas en inglés, o CBD como también se le conoce.

Analizaremos estos ambientes, conoceremos sus principales características, así como sus ventajas y desventajas.

Antes de comenzar con el análisis, debemos conocer algunos conceptos, por lo tanto se ha decidido organizar este capítulo de la siguiente manera:

Primero que nada en la sección 1.1 comenzaremos definiendo el paradigma CBD, además se muestran varias definiciones de lo que es un componente, desde el punto de vista de diferentes autores.

En la sección 1.2 se describe el desarrollo de componentes, al cual también se le llama desarrollo para reutilización, además se muestra el ciclo de vida o proceso de desarrollo de este. Además se habla acerca del desarrollo de software con componentes llamado también desarrollo con reutilización y se describe el proceso para lograr este tipo de desarrollo.

En la sección 1.3 se describe que son los ambientes de desarrollo de forma general y se hace un especial énfasis en aquellos basados en componentes.

En el Capítulo dos se describen una serie de características que sería importante implementarán los IDEs con el propósito de agilizar y facilitar el ensamble de un sistema a través de componentes.

Finalmente en el Capítulo 3 se Analizan algunos ambientes, de acuerdo a las características básicas mencionadas en el capítulo 2.

También se elabora un cuadro comparativo que muestra cuales de las características que seleccionamos, están contenidas en los ambientes de desarrollo estudiados.

1.1 Ingeniería de software basada en componentes

La ingeniería de Software basada en componentes, es un paradigma de desarrollo de software que reutiliza artefactos pre-existentes de software llamados componentes para la construcción de sistemas finales.

El uso de componentes tiene el objetivo de reducir el tiempo de elaboración de software, mejorar la calidad y reducir los tiempos de entrega, todo ello basándose en la idea de la reutilización de software. Las siguientes son algunas ventajas adicionales del uso de componentes:

- **Fácil adaptación:** Se refiere a que podemos seleccionar el componente que cumpla con los requerimientos del sistema o el que más se acerque. En otras palabras podemos decir que el sistema es como un rompecabezas, al cual le vamos poniendo las partes hasta completar este, pero en el proceso podemos remplazar componentes sin que esto afecte al resto del sistema.

- **Reduce el proceso de Riesgos :** Si un componente existe, se supone que ya fue probado, y además sabemos cuanto es el costo de este; por lo tanto disminuye la incertidumbre y el riesgo comparado con un nuevo desarrollo.

- **Poco mantenimiento:** Fácilmente se pueden reemplazar componentes obsoletos por otros que han sido mejorados o actualizados.

Así como existen ventajas de la utilización de componentes, también nos podemos enfrentar a algunas situaciones no deseadas. La siguiente lista muestra una serie de riesgos y retos a los cuales nos podemos enfrentar con el uso de componentes:

- **Satisfacción de los Requerimientos:** La selección de un componente es un proceso iterativo y el resultado depende de la clasificación y el mecanismo para obtenerlo. La búsqueda de un componente puede llevarse acabo en una amplia variedad de repositorios. Incluso cuando se ha encontrado un componente, puede que este no se adapte a los

requerimientos del sistema o falle en la interacción con algún otro componente. El proceso de búsqueda de componentes puede llegar a ser muy laborioso, debido a que quizá haya que consultar en diferentes repositorios, y puede darse el caso que no se encuentre un componente que cubra los características que se están requiriendo.

- **Interoperabilidad, Pruebas de unidad y de integración :** Es un reto considerable el asegurarse de que un componente que se elabora pueda ofrecer su funcionalidad a través de interfaces estándar, para así asegurar su interoperabilidad. Cada componente antes de ser liberado debió haber sido verificado y validado de forma independiente, pero representa un reto que este funcione adecuadamente al momento de integrarse a otros componentes y a otras aplicaciones.

Todo lo anterior permite comprender de forma general la naturaleza de los CBDs y el concepto de componente; sin embargo existen actualmente diferentes definiciones que describen más puntualmente estos.

Con el propósito de clarificar el término componente a continuación se listan algunas de ellas:

- Un componente es una unidad de composición con interfaces específicas, y un contexto bien definido. (S. Mahmood, R.lai and Y.S.KIM)

- Otra definición dice que un componente es "un paquete de software independiente que provee funcionalidad a través de interfaces bien definidas". (Alan W.Brown)

- un componente, los podemos definir como artefactos de software que tienen cierta funcionalidad, esta funcionalidad puede ser accesada a través de interfaces, usualmente sin necesidad de saber nada acerca de la parte interna del componente es decir de cómo esta hecho ([Szyperski y Pfister, 1997]).

Una vez que se ha descrito el paradigma de desarrollo de software basado en componentes y mostrado las diferentes definiciones de lo que es un componente,

en la siguiente sección hablaremos acerca del proceso de desarrollo de software para reutilización y desarrollo de software con reutilización.

1.2 Proceso de Desarrollo de Software Basado en Componentes.

Antes de comenzar con esta sección se debe aclarar que cuando hablamos del paradigma de desarrollo basado en componentes, este lo podemos dividir en dos partes fundamentales, una de ellas es el desarrollo de nuevos componentes, la cual tiene un proceso específico de desarrollo y que mencionaremos mas a detalle en la sección 1.2.1, La otra parte es conocida como el desarrollo de Sistemas utilizando componentes ya existentes y para el cual también existe un proceso propio que describiremos en la sección 1.2.3.

En la figura 1.1 podemos ver un esquema general del paradigma.

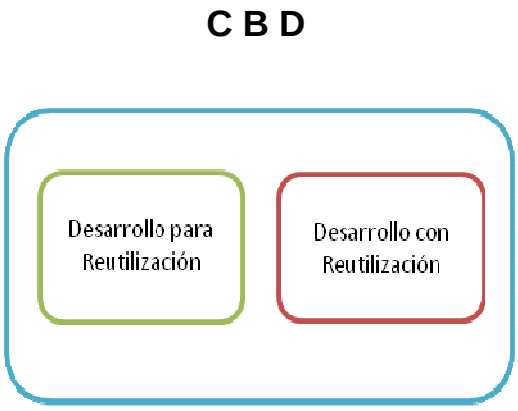


Fig 1.1 Desarrollo de Software basado en Componentes

1.2.1 Desarrollo para Reutilización

Cuando se desarrollan componentes se deben enfocar los esfuerzos en pensar en función de reutilización del código, es decir en diseñar componentes que ofrezcan una función genérica y estén dentro de un dominio de aplicación determinado, con la idea de que estos puedan servir para construir uno o mas sistemas.

A la persona encargada del desarrollo de componentes se le conoce como desarrollador de componentes.

Existen diferentes modelos para el desarrollo de software tradicional como el modelo Secuencial (sequential) y el modelo Evolutivo (evolutionary) ([Ivica CrnKovic, Stig Larsson] 2005]) . El primer modelo define una secuencia de actividades en la cual una actividad sigue después de que otra ya se completó, Ejemplos de este modelo son el de Cascada y el modelo en V ([Ivica CrnKovic, Stig Larsson] 2005]).

Por otra parte el modelo Evolutivo permite varias actividades en paralelo, sin necesidad de que alguna otra haya terminado, los modelos que lo representan son: el desarrollo Iterativo y el Incremental ([Ivica CrnKovic, Stig Larsson] 2005]).

Para el desarrollo de componentes en general se pueden aplicar los modelos antes mencionados, Considerando sin embargo la etapa de almacenamiento que consiste en enviar a un repositorio el componente ya terminado, probado y liberado, tal como se muestra en la figura 1.2.

A continuación se describen brevemente cada uno de estas etapas o actividades : ver figura 1.2

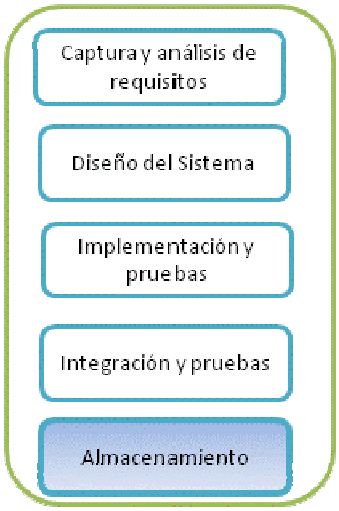


Fig. 1.2 Desarrollo para Reutilización

- **Captura y Análisis de Requisitos :** Con la participación del cliente se define la funcionalidad, restricciones y objetivos del sistema.

- **Diseño del Sistema:** El sistema se divide, se establece la funcionalidad del software, se diseña la arquitectura.

- **Implementación y Pruebas :** Se refiere al proceso de desarrollo o codificación del software.

- **Integración y pruebas:** Se refiere a la unión de las partes desarrolladas.

- **Almacenamiento o Envío a un Repositorio:** Como se mencionó anteriormente consiste en el almacenamiento o envió del componente a un repositorio.

1.2.2 Desarrollo con Reutilización

Como se mencionó anteriormente desarrollo con reutilización, Se refiere al desarrollo de sistemas o de aplicaciones con componentes ya existentes, es decir componentes específicos que alguien mas ya elaboró y los depositó en un repositorio.

Este tipo de desarrollo se enfoca en la identificación de estas entidades reutilizables, basándose en los requerimientos del sistema y la disponibilidad de los componentes existentes.

Para la realización de sistemas de este tipo, no se pueden utilizar los ciclos de vida de desarrollo de software tradicionales debido a que las actividades después del diseño cambian o son específicas para este tipo de desarrollos.

Podemos basarnos en el Cascada tradicional, pero se deben agregar algunas etapas.

A continuación se describen brevemente cada uno de estas etapas o actividades: ver figura 1.3

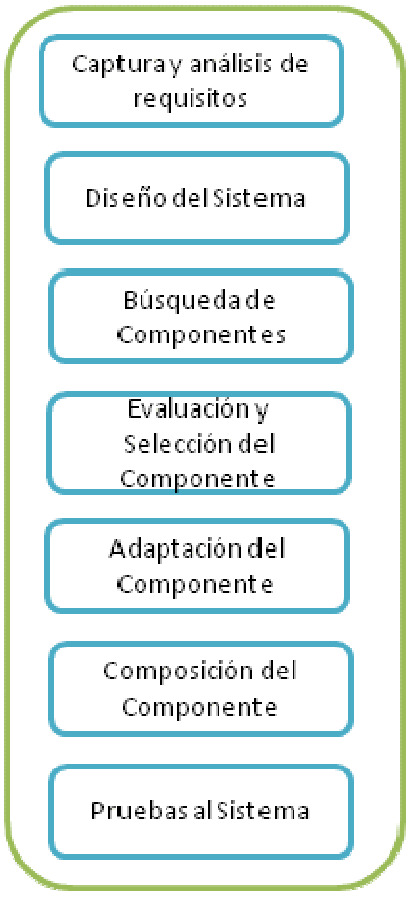


Fig. 1.3 Desarrollo con Reutilización

a) Captura y Análisis de Requisitos

En esta etapa, se definen las necesidades del cliente, se definen los objetivos del proyecto, se generan documentos para la especificación de requerimientos, se define el alcance del proyecto y finalmente se genera un documento con las especificaciones de los requerimientos del cliente.

b) Diseño del Sistema
En esta etapa se genera un modelo de cómo será el sistema final, por tanto es aquí donde surge la necesidad de utilizar algunos componentes.

c) Búsqueda de Componentes
De acuerdo a las necesidades del Sistema, se comienzan a buscar componentes que cubran dicho requerimiento, obviamente para esto debe existir un repositorio de componentes que permita realizar búsquedas de manera eficiente.

d) Evaluación y Selección del Componente
Una vez que hemos localizado el componente se procede a evaluar si contiene las características que necesitamos o si se le deben realizar modificaciones para poder ensamblarlo.

e) Adaptación del Componente
Una vez que ya hemos seleccionado el componente, si este no cumple al 100% con las características deseadas, podemos realizar lo que comenta el punto anterior para adaptarlo al diseño generado del sistema.

f) Composición del Componente
Esta etapa consiste en interrelacionar dos o más componentes con la idea de ensamblar el sistema.

g) Pruebas al Sistema
Una vez que el sistema ya fue ensamblado y paso por las etapas antes mencionadas, debemos probar que este funcione de acuerdo a las especificaciones, es decir que cubra los requerimientos.

Una vez que hemos descrito las generalidades de CBD, en la siguiente sección se describe el concepto de Ambiente de Desarrollo y se hace énfasis en aquellos orientados al paradigma CBD.

1.3 Ambientes de Desarrollo

Un ambiente de desarrollo de software es un conjunto de herramientas que soportan actividades relacionadas con el desarrollo de sistemas.

Los ambientes de desarrollo van desde un pequeño conjunto de herramientas usadas solo por un programador, hasta aquellos muy complejos que ayudan a

coordinar grandes grupos de desarrolladores y administradores en la producción y mantenimiento de grandes sistemas de Software.

Podemos mencionar también que un ambiente de desarrollo, es una herramienta que apoya a los programadores para facilitar su labor de programación, Ofrecen un entorno amigable. Estos entornos se componen generalmente de:

- Un Editor de Código
- Un Compilador
- Un depurador
- Interfaz Gráfica

1.3.1 Ambientes de Desarrollo basados en componentes

Este tipo de ambientes de desarrollo son muy parecidos a los mencionados anteriormente, la diferencia es que estos contienen algunas características especiales requeridas para la implementación del paradigma CBD.

Las Características a las que nos referimos son por ejemplo facilidad para composición de componentes, adaptación de nuevos componentes, almacenamiento de componentes, facilidad para almacenar componentes en repositorios remotos, etc.

En general un ambiente de desarrollo basado en componentes es aquel que permite desarrollar una o más actividades descritas en las fases de los ciclos de desarrollo de software basado en componentes que describimos anteriormente.

Capítulo 2. Características de los Ambientes de Desarrollo para CBD

Es difícil encontrar un IDE que cumpla con todas las características que cada desarrollador requiere para construir componentes o sistemas basados en componentes, es decir difícilmente encontraremos una herramienta ideal; sin embargo podemos seleccionar aquel que más se acerque a lo que estamos buscando.

2.1 Características Deseables

A continuación se describen una serie de características que deberían de ser incluidas en los IDEs para facilitar las actividades a realizar al momento de ensamblar aplicaciones con componentes.

Estas características se seleccionaron luego de la lectura e investigación de diferentes artículos de autores muy reconocidos en el área (*Chris Luer and Andre van der Hoek,* Richard A. Pyne and Ezra K. Mugisa). Ellos describen necesidades con respecto a rapidez y facilidad para ejecutar algunas actividades al momento de la composición de componentes.

Es importante hacer énfasis en que estas características son las básicas necesarias para la etapa de ensamble de componentes con el objetivo de crear una aplicación. Existen muchas más características que diversos autores comentan pero algunas de ellas son de acuerdo a la necesidad especifica de un programador o al gusto de los autores de los diferentes artículos.

2.1.1 Facilidades para la búsqueda de componentes

Una de las tareas que presenta mayor dificultad a la hora de construir aplicaciones, es precisamente el encontrar los componentes adecuados, si no se dispone de una herramienta para seleccionar componentes que permita hacerlo bajo diferentes criterios y en diferentes ubicaciones, esta actividad puede resultar un proceso largo y complicado.

2.1.1.1 Búsqueda remota

Se requiere que nuestro IDE contenga una herramienta para almacenar y encontrar componentes en un repositorio que físicamente se encuentra distante, en la cual bajo ciertos criterios de búsqueda sea capaz de encontrar un componente como el que estamos buscando o uno similar que requiera alguna adaptación para que lo podamos utilizar en nuestra aplicación, y después de esto poderlo descargarlo.

2.1.1.2 Búsqueda Local

El IDE debería contener un repositorio local en el cual podamos almacenar los componentes nuevos que vamos creando, además de una herramienta para encontrar componentes de acuerdo a ciertos criterios de búsqueda.

2.1.2 Seleccionar

2.1.2.1 Facilidades para Adaptar o modificar los componentes

Se refiere a Proporcionar herramientas para desempaquetar los componentes, facilidad para modificar código fuente, utilerías para modificar los componentes a través de diagramas, integrar la opción de ingeniería inversa a código binario pre compilado. Todo lo anterior con la idea de reducir el tiempo de modificación y así lograr que esta adaptación sea rentable y no se prefiera hacer nuevamente el componente en lugar de adaptarlo.

2.1.3 Facilidades para Composición

Esta es una de las características preponderantes dentro de los IDEs, es decir se requiere que se proporcione un entorno o área en la cual con el menor esfuerzo posible, seamos capaces de combinar y ensamblar componentes para crear una aplicación completa. La mayoría de los IDEs implementa ambientes gráficos, que nos ayudan a la composición con solo arrastrar y soltar.

2.1.3.1 Scripts

Los scripts son archivos de texto que contienen instrucciones en un lenguaje de Scripts, este lenguaje es más rápido de programar pero no necesariamente mas fácil de interpretar, la ventaja es que puede ser generado y leído de forma automática, por ejemplo, puede ser el resultado generado por un diagrama y posteriormente este puede ser leído por otra herramienta y convertirlo a algún lenguaje de aplicación final como Java, C, etc.

2.1.3.2 Diagramas

Los diagramas son especificaciones que son expresadas de forma gráfica, generalmente la notación que utilizan son cajas que representan componentes y líneas que muestran la relación entre estos. La utilización de los diagramas permite diseñar la aplicación completa de forma gráfica lo cual resulta más fácil y rápido, pero ahí no termina todo, lo ideal es que podamos generar código basándonos en el diseño, esto minimiza el tiempo de desarrollo.

2.1.4 Facilidades para probar (testing)

2.1.4.1 Diseño

Una característica que pocos IDEs implementan por la complejidad que esta implica, es la de tener la posibilidad de probar un diseño, es decir que existan las herramientas necesarias primero para de forma gráfica representar una aplicación y posteriormente antes de generar o ensamblar los componentes, verificar que el diseño sea correcto de acuerdo a las especificaciones solicitadas.

2.1.4.2 Implementación

Una vez ensamblada la aplicación debemos tener la facilidad para probar la funcionalidad de esta e incluso si se generan nuevos componentes probar estos de forma independiente.

Debemos hacer hincapié en esta característica, ya que la etapa de pruebas es en la que mas se invierte tiempo y dinero. Ahí radica precisamente la importancia de contar con herramientas seguras y fáciles de operar que nos permitan agilizar y optimizar las pruebas.

2.1.5 Facilidades para ejecutar las aplicaciones

La creación de aplicaciones ejecutables es el objetivo principal de todos los procesos de desarrollo, sin embargo existen IDEs que van más allá e incorporan mecanismos que permiten ejecutar aplicaciones ensambladas a través de componentes, tanto de forma parcial como total.

2.1.5.1 Ejecutar aplicación de forma Parcial

Algunos ambientes de composición requieren que toda la aplicación este terminada antes de que esta pueda ser ejecutada, sin embargo existen otros que permiten ejecutar de forma parcial la aplicación, además, permiten ejecutar de manera independiente los componentes con propósitos de prueba.

2.1.5.2 Empaquetar

Una vez que la aplicación ha sido construida esta debe poder ser empaquetada para poder ser distribuida y ejecutada fuera del ambiente de desarrollo. En algunos casos incluso para poder lograr esto, se debe agregar un código adicional que le llamamos “pegamento”, para que este pueda ser ejecutado en otros ambientes.

2.1.6 Documentación del componente y la aplicación

Los IDEs deberían implementar herramientas que permitan documentar de forma automática un componente y toda la aplicación. Deben generar documentación acerca de las interfaces, los métodos, los eventos, textos descriptivos de funcionalidad, diseño de la aplicación, proveedor del componente, versión, etc.

Capítulo 3. Ambientes de Desarrollo para CBD

Una vez que en el capítulo anterior se ha definido lo que es un ambiente de desarrollo y en especial aquellos basados en componentes, en este capítulo, describiremos las principales características de algunos ambientes actuales de desarrollo basados en CBD.

3.1 Revisión de Ambientes de Desarrollo Actuales

Con la idea de conocer más a fondo los ambientes de desarrollo de componentes, a continuación se describen las características principales de algunos de los IDEs más actuales.

3.1.1 Black Box Component Builder

3.1.1.1 Descripción

Black Box Component Builder es un ambiente de desarrollo basado en componentes con licencia open source, que permite programar utilizando un lenguaje de programación llamado Component Pascal. (Cuno Pfister,Oberon Systems,2001)
Pretende ser una herramienta que se pueda utilizar para aplicaciones científicas y de ingeniería. Debido a que no requiere grandes capacidades de hardware, puede ser utilizada en lugares donde no se tienen equipos muy sofisticados.
Oberon Microsystems desarrolló en 1992 un Framework para componentes llamado BlackBox, este fue escrito en component pascal.

Simplifica el desarrollo de componentes para interfaces gráficas de usuario, este incluye un procesador de palabras, componentes de acceso a bases de datos sql, compilador y un ambiente integrado de desarrollo, además posee un repositorio con componentes previamente desarrollados y que están listos para utilizarse.

3.1.1.2 Facilidades para la búsqueda de componentes

3.1.1.2.1 Búsqueda remota

No existe propiamente una opción que de manera automática permita buscar en repositorios remotos; sin embargo Existe una página en internet a la cual se puede acceder desde el sitio de la compañía Oberón, de la cual podemos descargar

nuevos componentes y compartir los nuestros. La dirección es la siguiente: http://www.zinnamturm.eu/topics.htm#Converter

3.1.1.2.2 Búsqueda Local

Black Box posee un repositorio de componentes con una gran cantidad de ellos ya listos para ser utilizados.

A través de una interfaz gráfica nos permite seleccionar el componente o módulo (como se le llama en Black Box) que necesitamos o queremos utilizar.

Estos componentes están organizados o agrupados de acuerdo a su función, por ejemplo los de conexiones a sql, los de control, de formas, de texto, los componentes COM, etc. Ver fig. 3.1

Los nuevos componentes que se desarrollen se pueden agregar a este repositorio ubicándolos en la clasificación que les corresponde.

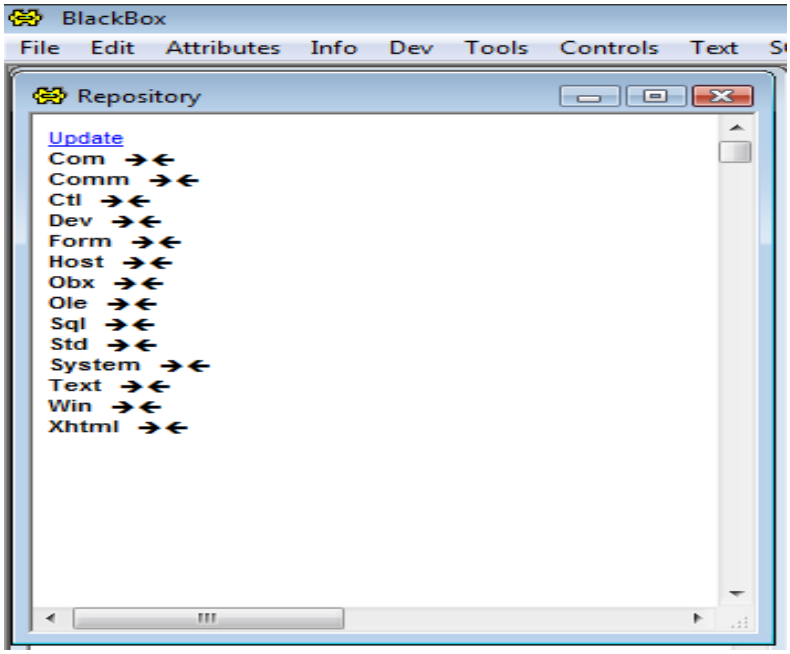


Fig. 3.1 Repositorio local de componentes BlackBox.

3.1.1.3 Seleccionar

3.1.1.3.1 Facilidades para Adaptar o modificar los componentes

Como se vio en la sección anterior este ambiente de desarrollo posee un repositorio local que esta organizado y clasificado para que sea fácil la selección de componentes.
Los componentes se encuentran en formato ejecutable de tal manera que los podamos mandar llamar, pasarles parámetros y utilizar su funcionalidad, pero además el repositorio contiene el código fuente de estos componentes así como su documentación, por lo tanto podemos modificarlos y adaptarlos a nuestras necesidades sin necesidad de utilizar algún algoritmo especial, lo cual hace muy fácil la adaptación.
Podemos utilizar un objeto que ya existe para crear un objeto similar pero de otro tipo, es decir adaptarlo; para este propósito el objeto contiene una función llamada Factory, esta función o método permite crear diferentes tipos para el mismo objeto.

3.1.1.4 Facilidades para Composición

3.1.1.4.1 Scripts

Esta herramienta no genera ningún script, debido a que los componentes contienen el código fuente.

3.1.1.4.2 Diagramas

Este ambiente de desarrollo no posee ninguna herramienta para generar diagramas.

3.1.1.5 Facilidades para probar (testing)

3.1.1.5.1 Diseño

No contiene herramientas para probar diseño

3.1.1.5.2 Implementación

Black Box implementa el concepto de desarrollo de aplicaciones de forma rápida RAD por sus siglas en ingles, contiene una interfaz gráfica que nos permite desarrollar seleccionando o arrastrando.

Para evitar errores en las aplicaciones o para corregirlos, se tiene una herramienta que checa los links de los componentes de todo el sistema o de algunos módulos en especial. Ver Fig. 3.2

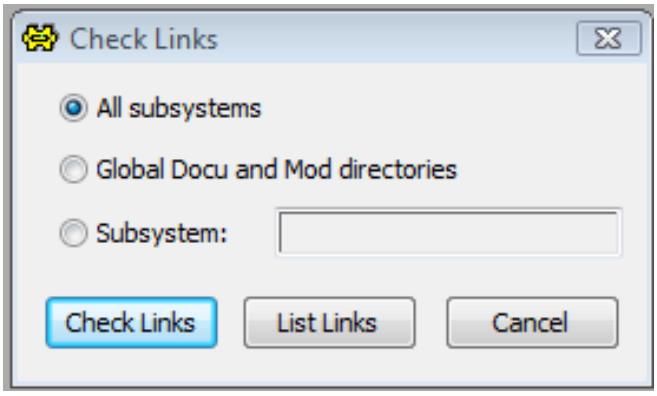


Fig. 3.2 Herramienta para checar links

3.1.1.6 Facilidades para ejecutar las aplicaciones

3.1.1.6.1 Ejecutar aplicación de forma Parcial

Este ambiente de desarrollo permite ejecutar de forma parcial la aplicación, se pueden ejecutar los módulos o componentes que se tengan ensamblados aun y que la aplicación no este terminada.

3.1.1.6.2 Empaquetar

Los componentes unas vez creados se pueden distribuir en código fuente o aplicarles una codificación, Así mismo tenemos la opción de decodificar un componente recibido y convertirlo a código fuente. Los componentes además se pueden organizar o agrupar en paquetes.

.

3.1.1.7 Documentación del componente

La documentación es una de las fortalezas de este ambiente de desarrollo, ya que implementa el concepto de componentes de texto en los cuales se describe de forma detallada la funcionalidad del componente y sus interfaces, luego el código fuente del componente va embebido en este, creando así un solo componente.

Además contiene una herramienta que nos permite extraer del componente solo las interfaces si así lo deseamos.

3.1.2 NetBeans IDE

3.1.2.1 Descripción

Es un entorno de desarrollo de software escrito en java que permite crear aplicaciones utilizando el lenguaje java, entre otros mas como C/C++. Este IDE permite desarrollar bajo el paradigma de desarrollo de software basado en componentes utilizando el modelo JavaBean. (Netbeans.org 2009)

Permite crear nuevos componentes (beans), así como ensamblar aplicaciones utilizando componentes ya existentes.

3.1.2.2 Facilidades para la búsqueda de componentes

3.1.2.2.1 Búsqueda remota

No contiene ninguna facilidad especial para buscar en repositorios remotos, pero los beans se pueden transportar en paquetes y estos a su vez comprimirlos con el formato JAR que puede incluir firmas digitales, y enviarlos a través de algún medio remoto.

3.1.2.2.2 Búsqueda Local

NetBeans contiene una paleta de componentes que sirve como repositorio para ir almacenando los beans que vamos creando, para que estos puedan estar disponibles y sean utilizados por diferentes clases si así se requiere. Ver fig. 3.3
Otra forma de crear repositorios locales, es organizando en paquetes los beans y colocándolos en directorios específicos y ordenados.

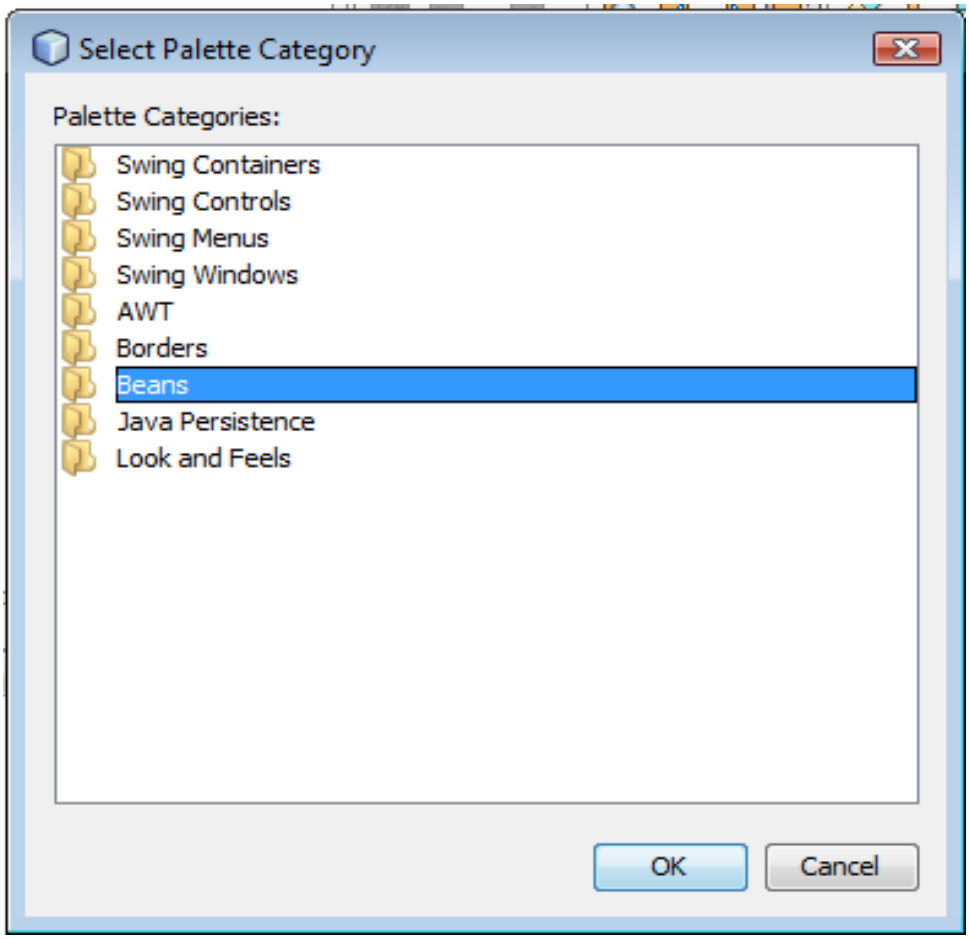


Fig. 3.3 Repositorio local de beans.

3.1.2.3 Seleccionar

3.1.2.3.1 Facilidades para Adaptar o modificar los componentes

En este IDE es muy fácil adaptar o modificar los componentes ya que los Beans tienen la característica de ser serializables es decir los componentes se pueden convertir a bytes y regresar a su estado original.

Los Beans generalmente están organizados en paquetes, por lo tanto basta con abrir estos paquetes, apoyándonos en la interfaz gráfica de NetBeans, y modificar el código.

Este IDE contiene herramientas que nos facilitan la modificación de los componentes:

- Administrador y editor de propiedades que permite modificar el componente
- Interfaz gráfica para personalizar Java Beans
- Netbeans info Editor que proporciona información sobre los elementos que componen al componente como métodos, eventos, propiedades, etc.
- Herramienta para extraer interfaces de forma muy simple. Ver fig. 3.4

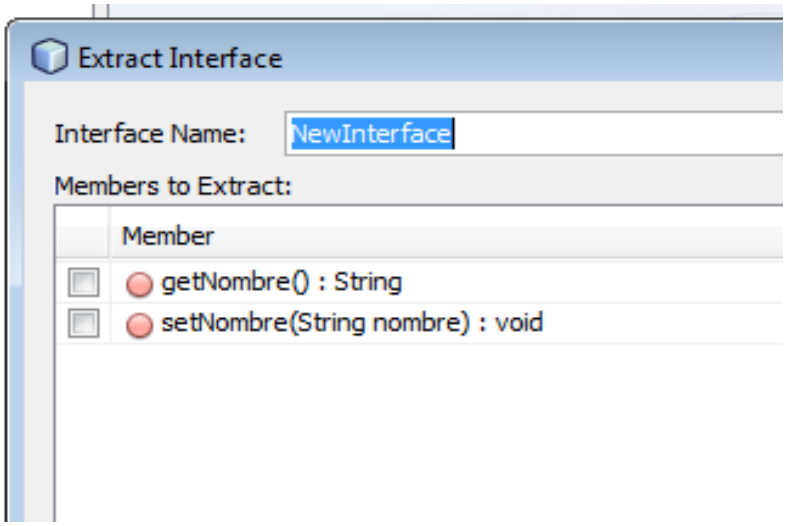


Fig 3.4 Extracción de Interfaces en NetBeans

3.1.2.4 Facilidades para Composición

A través del diseñador gráfico, podemos agregar el bean que acabamos de crear a la forma así como otros componentes Swing, AWT,combinar varios beans creados etc. Pero en realidad no presenta facilidades para composición.

3.1.2.4.1 Scripts

Esta herramienta no genera ningún script

3.1.2.4.2 Diagramas

Este ambiente de desarrollo no posee ninguna herramienta para generar diagramas

3.1.2.5 Facilidades para probar (testing)

3.1.2.5.1 Diseño

No contiene herramientas para probar diseño

3.1.2.5.2 Implementación

En netbeans podemos crear pruebas de manera automática y evaluar los resultados con respecto a los requerimientos, a través de un conjunto de Bibliotecas llamadas Junit , al cual accedemos a través de un plug in. Esta herramienta nos permite crear pruebas automatizadas de caja negra y de caja Blanca.

3.1.2.6 Facilidades para ejecutar las aplicaciones

3.1.2.6.1 Ejecutar aplicación de forma Parcial

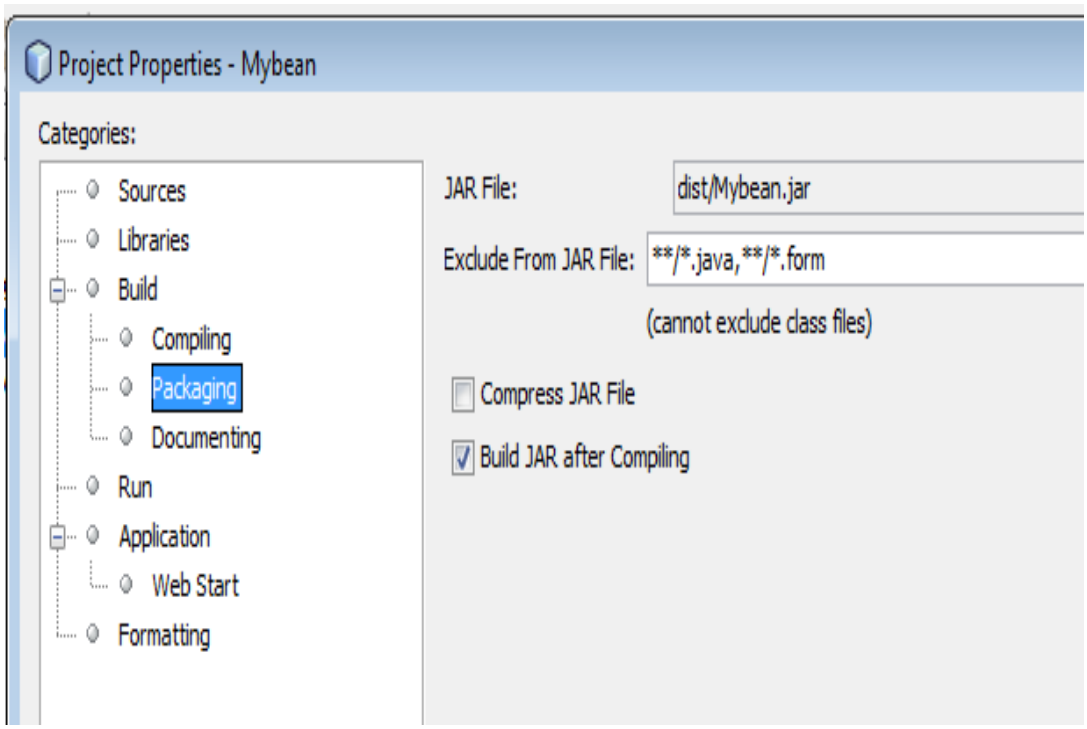
Este ambiente de desarrollo permite ejecutar de forma parcial la aplicación, se pueden ejecutar las clases dentro de un paquete o componente, aun y que la aplicación no este terminada.

3.1.2.6.2 Empaquetar

Este IDE permite crear archivos JAR (Java Archive),estos son la forma sencilla y eficiente de transportar recursos. Con un archivo JAR facilitamos la distribución e instalación de una gran variedad de archivos, applets, video, sonido, imágenes, texto, y sobre todo beans.

Además un archivo JAR puede contener firmar digitales que aseguran la integridad y autenticidad de los datos.

Esto lo podemos hacer desde el editor de propiedades del proyecto. Ver fig. 3.5



.

Fig 3.5 Empaquetado de clases

3.1.2.7 Documentación del componente

NetBeans contiene una herramienta que nos permite generar documentación de una clase de forma simple y rápida. Los datos que contiene la documentación son nombre de la clase, de que clase extiende, los constructores, los métodos, las interfaces, etc. Ver. fig. 3.6

Method Summary

java.lang.String	getNombre() Get the value of nombre
void	setNombre(java.lang.String nombre) Set the value of nombre

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

nombre

```
protected java.lang.String nombre
```

Constructor Detail

MyBean

```
public MyBean()
```

Method Detail

Fig. 3.6. Documentación NetBeans

3.1.3 Odyssey Share IDE

3.1.3.1 Descripción

OdysseyShare es un ambiente de desarrollo que soporta CBD . Las herramientas que provee permiten la administración de flujos de trabajo, recuperación de componentes, obtención de requerimientos, patrones, administración de componentes, e ingeniería inversa. (Claudia Werner, Marco Mangan,OdysseyShare).

Es un entorno de desarrollo basado en la reutilización de modelos en el uso eficaz de una estrategia de reutilización en todo el proceso de desarrollo de software, proporcionando métodos, herramientas y procedimientos que son adecuados para la especificación de modelos y aplicaciones de un campo específico.

El principal objetivo de Odyssey es proporcionar mecanismos basados en la reutilización para el desarrollo de software, que actúa como un marco en el que los

modelos, conceptos, arquitecturas de software, patrones de diseño y modelos de aplicación pertenecen a áreas específicas previamente seleccionadas.

3.1.3.2 Facilidades para la búsqueda de componentes

3.1.3.2.1 Búsqueda remota

OdysseyShare implementa una arquitectura llamada ComPublish, la cual permite que los desarrolladores puedan publicar y recuperar artefactos de software que están disponibles en el internet como modelos, diagramas, código fuente, etc. Tal como se muestra en la figura 2.3
Provee mecanismos para recuperar componentes de diferentes repositorios, esto lo hace a través de un servidor llamado le select . así como lo podemos apreciar en la figura 3.7

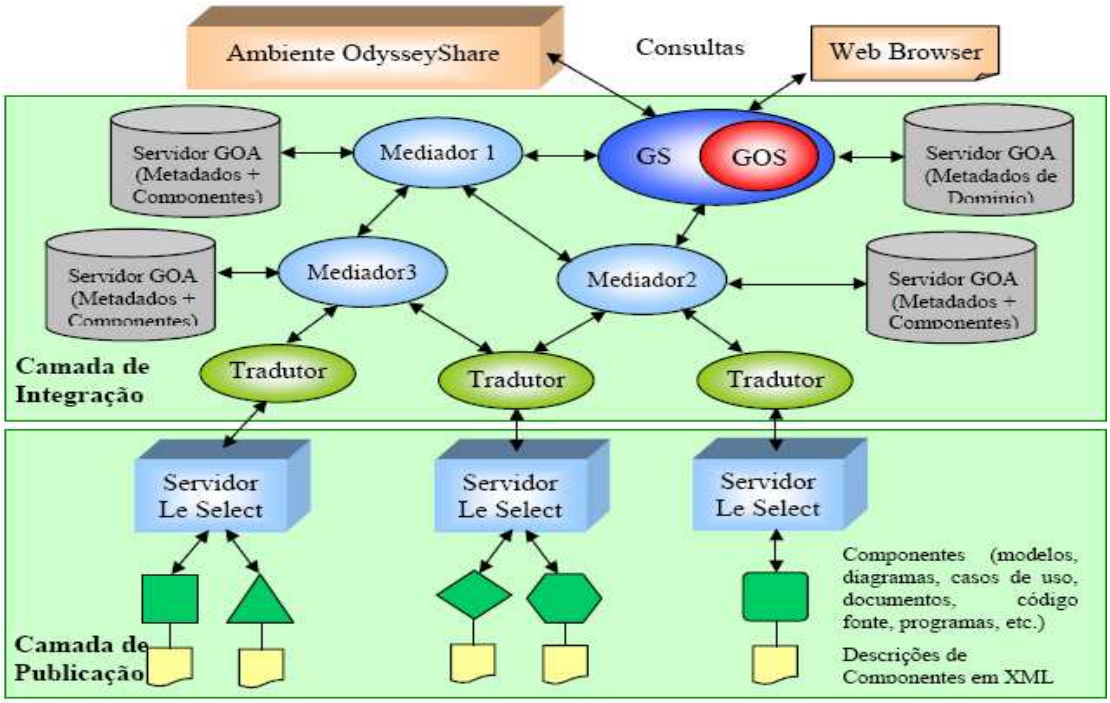


Fig. 3.7 Arquitectura ComPublish

3.1.3.2.2 Búsqueda Local

.

La herramienta permite también el almacenamiento de los componentes en un repositorio local, así como la búsqueda y recuperación de estos.

3.1.3.3 Seleccionar

3.1.3.3.1 Facilidades para Adaptar o modificar los componentes

Los componentes se implementan en Lenguaje Java, estos a su vez están distribuidos en paquetes que contienen código fuente, diagramas, textos descriptivos, etc, por lo tanto la modificación de componentes resulta muy simple utilizando el editor de Oddysey.

3.1.3.4 Facilidades para Composición

Contiene un ambiente gráfico amigable, además de que se apoya de diagramas UML, para diseñar o mostrar la relación o unión entre los componentes

3.1.3.4.1 Scripts

Genera scripts de ejecución.

3.1.3.4.2 Diagramas

Se basa en Diagramas de Actividad de UML, para describir los flujos de trabajo. Por lo tanto si contiene esta característica, Implementa Diagramas UML.

3.1.3.5 Facilidades para probar (testing)

Existen Herramientas para probar tanto diseño como implementación a través de simulación.

3.1.3.5.1 Diseño

Para probar diseño se aplica un proceso de simulación que tiene como objetivo la detección de errores en el modelo.

3.1.3.5.2 Implementación

Se aplica un modelo de simulación para generar la ejecución esperada de la aplicación, los métodos que se aplican para pruebas de diseño e implementación son llamados: “Event Based Simulation y Montecarlo Simulation”.

3.1.3.6 Facilidades para ejecutar las aplicaciones

3.1.3.6.1 Ejecutar aplicación de forma Parcial

Se puede ejecutar de forma parcial la aplicación, a través de procesos de simulación.

3.1.3.6.2 Empaquetar

Los Componentes son desarrollados de acuerdo a un dominio específico de ingeniería y están disponibles en paquetes que incluyen código fuente, diagramas, descripciones textuales y el código binario del componente.

3.1.3.7 Documentación del componente

Contiene una herramienta llamada Le Select Metadata que permite describir los componentes de acuerdo al estándar de XML. Esta herramienta permite extraer documentos, código, interfaces, autor, lenguaje, dominio de aplicación, etc.

3.1.4 ORION IDE

3.1.4.1 Descripción

Es un ambiente de desarrollo de Software que integra diferentes herramientas para facilitar la creación de componentes y software en general. (Daniel Lucrédio, Calebe de Paula Bianchini, Sao Carlos Brazil)

Algunas de las herramientas que incluye son las siguientes:

- Herramienta de modelado con UML
- Herramienta de programación con Java
- Herramientas de Red
- Plataforma Middleware
- Modelo de procesos.

3.1.4.2 Facilidades para la búsqueda de componentes

3.1.4.2.1 Búsqueda remota

Contiene un repositorio que implementa mecanismos para de forma remota almacenar, buscar y recuperar artefactos de software. Las búsquedas y almacenamiento son posibles a través de middleware ver Fig. 3.8

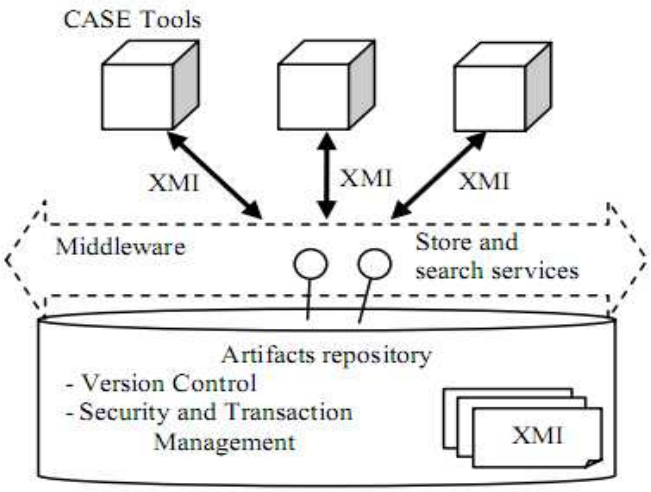


Fig. 3.8. Arquitectura del repositorio de Artefactos

3.1.4.2.2 Búsqueda Local

El repositorio mencionado anteriormente permite de forma local almacenar y recuperar artefactos de software

3.1.4.3 Seleccionar

3.1.4.3.1 Facilidades para Adaptar o modificar los componentes

Contiene una Herramienta llamada JADE que permite editar el código de los componentes y realizar modificaciones de forma simple y rápida.

3.1.4.4 Facilidades para Composición

3.1.4.4.1 Scripts

Genera scripts en un lenguaje de metadatos llamado XMI que es un formato usado para representar descripciones de UML.

3.1.4.4.2 Diagramas

Contiene una herramienta de modelado llamada MVCASE que provee técnicas gráficas y textuales basadas en notación UML. Permite especificar los sistemas en diferentes niveles de abstracción y en cuatro vistas: casos de uso, lógica, Componentes y vista de deployment. Ver fig. 3.9

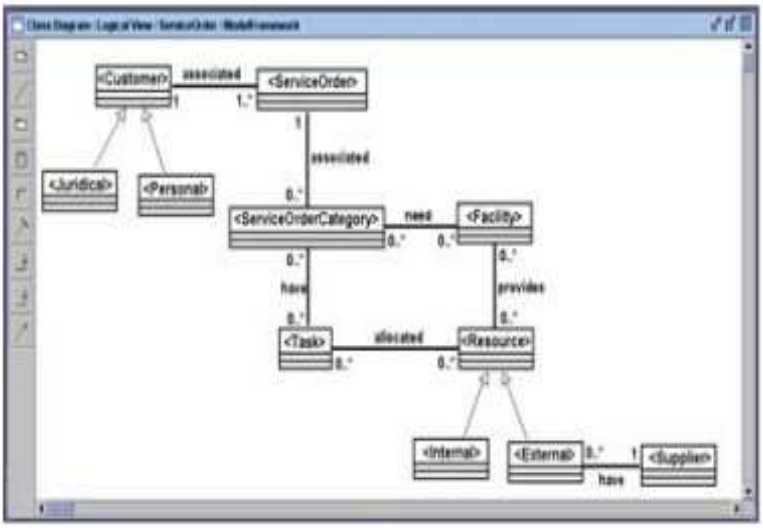


Fig. 3.9 diagramas MVCASE

3.1.4.5 Facilidades para probar (testing)

3.1.4.5.1 Diseño

No se prueba el diseño aunque contiene herramientas de Diagramación para diseñar utilizando UML.

3.1.4.5.2 Implementación

Contiene una herramienta llamada JADE la cual permite editar programas en lenguaje JAVA, Esta permite compilar, ejecutar y probar el código utilizando un debug.

El Código es generado de forma parcial por la herramienta de diagramación llamada MVCASE, y transferido a la herramienta JADE en donde se puede manipular el código.

3.1.4.6 Facilidades para ejecutar las aplicaciones

3.1.4.6.1 Ejecutar aplicación de forma Parcial

Como ya se mencionó anteriormente la Herramienta de diagramación MVCASE genera el código de los componentes de forma casi total, por lo tanto podemos diseñar un componente, generar automáticamente el código en lenguaje JAVA y ejecutarlo en JADE.

3.1.4.6.2 Empaquetar

Los componentes se empaquetan y se guardan en el repositorio. El paquete contiene el código del componente, así como el diseño de este.

3.1.4.7 Documentación del componente

La documentación del componente se genera de forma automática desde la herramienta de Diseño y generación de código llamada MVCASE, una vez que hemos realizado el diseño con nomenclatura UML.

3.1.5 PECOS IDE

3.1.5.1 Descripción

El proyecto PECOS, permite implementar el paradigma de desarrollo de Software basado en componentes , es utilizado para pequeños sistemas embebidos, proporcionando un entorno que apoya la especificación, la composición, configuración de control, y el despliegue de los sistemas integrados construidos a partir de componentes de software. (Benedikt Schulz, Thomas Genssler, Alexander,1999)

PECOS se aplicó en el contexto de equipos pequeños de dispositivos integrados.

3.1.5.2 Facilidades para la búsqueda de componentes

3.1.5.2.1 Búsqueda remota

No presenta facilidades para la búsqueda remota

3.1.5.2.2 Búsqueda Local

Permite la navegación sencilla a través del repositorio. El repositorio es capaz de almacenar diferentes versiones de un mismo componente.

3.1.5.3 Seleccionar

3.1.5.3.1 Facilidades para Adaptar o modificar los componentes

El entorno de desarrollo contiene facilidades para editar las especificaciones del componente en un lenguaje llamado Coco.

Provee facilidades para la modificación de las propiedades de los componentes durante la adaptación de estos.

3.1.5.4 Facilidades para Composición

3.1.5.4.1 Scripts

Genera una descripción en lenguaje Coco que posteriormente es usada para generar código en C++ o Java.

3.1.5.4.2 Diagramas

No contiene herramientas para generar diagramas.

3.1.5.5 Facilidades para probar (testing)

3.1.5.5.1 Diseño

No contiene herramientas para probar diseño

3.1.5.5.2 Implementación

Se prueban los componentes de forma independiente durante el desarrollo, cada uno de estos se prueba utilizando “Test Bed” y “Driver Generation”.

3.1.5.6 Facilidades para ejecutar las aplicaciones

3.1.5.6.1 Ejecutar aplicación de forma Parcial

Se puede ejecutar de forma parcial la aplicación

3.1.5.6.2 Empaquetar

Los componentes se distribuyen en paquete, este contiene la implementación del componente y su documentación.

3.1.5.7 Documentación del componente

Contiene una herramienta para documentación que el desarrollador puede utilizar para documentar los componentes que va creando.

3.2 Cuadro Comparativo de Ambientes de Desarrollo Basado en Componentes

La siguiente tabla muestra un cuadro comparativo de los ambientes de desarrollo de software estudiados en la sección anterior y las características deseables para todo ambiente de desarrollo que se describen en el Capítulo anterior.

La intensión de esta comparación es mostrar que características se implementan más, que ambiente de desarrollo implementa más características, cuales deberían ser implementadas en futuros desarrollos, etc.

Aun y que no se estudiaron una gran cantidad de IDEs, esta muestra, nos da una tendencia en cuanto a la incorporación de características en los ambientes de desarrollo.

Ambientes de Desarrollo CBD	F A C I L I D A D E S									
	Busquedas		Adaptar	Composición		Pruebas (Testing)		Ejecutar		Docu -
	Remota	Local	Modificar	Scripts	Diagramas	Diseño	Implementación	Parcial	Empaquetar	mentac
Black Box	No	Si	Si	No	No	No	Si	Si	Si	Si
NetBeans	No	Si	Si	No	No	No	Si	Si	Si	Si
Odyssey Share	Si	Si	Si	No	Si	Si	Si	Si	Si	Si
Orion	Si	Si	Si	Si	Si	No	Si	Si	Si	Si
PECOS	No	Si	Si	Si	No	No	Si	Si	Si	Si

Tabla 3.1 Cuadro comparativo características de IDEs

3.3 Conclusión

Como hemos podido observar ninguno de los ambientes de desarrollo que se analizaron implementa todas las características básicas; esto nos hace pensar que no existen IDEs lo suficientemente completos que nos sirvan como una herramienta ágil de desarrollo.

Es difícil encontrar una herramienta que implemente todo lo que como desarrolladores y ensambladores necesitamos.

En mi opinión si los modelos o los conceptos de una metodología o de un paradigma no pueden ser implementados de forma sencilla a través de una herramienta; difícilmente estos tendrán éxito.

Por lo tanto para facilitar el éxito de la implementación de la ingeniería de Software basada en Componentes, se debe seleccionar una herramienta que contenga al menos la mayoría de las características básicas antes mencionadas.

Un punto importante es que la mayoría de los IDEs surgen como proyectos aislados de algunas universidades y solo se les da seguimiento en estas, en realidad no han logrado que la comunidad los adopte.

Como trabajo futuro propondría el desarrollo de IDEs que se especialicen en el desarrollo con componentes y que estén apegados a las características que se requieren para hacer mas popular y efectivo este tipo de desarrollo y de esta manera poder obtener los beneficios que se prometen con la utilización de este paradigma.(reducciones de tiempo de desarrollo, menos errores, inversión económica menor, etc.)

.

Bibliografía

Alan Brown y Balbir Barn,, Enterprise-Scale CBD: Building Complex Computer Systems From components, pp-20-30

Andrew McVeigh,2006, Jeff Krammer,Using Resemblance to Support Component Reuse and Evolution, pp 5,7.

Irfan Pyarali and Douglas C. Schmidt,, An Overview of the CORBA Portable Object Adapter, pp 10-15

Ivica CrnKovic, Stig Larsson and Michel Chaudron,2005, Component-Based Development Process and Component LifeCycle*,pp. 321-327*

S. Mahmood, R. Lai and Y.S. Kim, 2007, Survey of component-based software development ,The Institution of Engineering and technology , pp 57-66

Spiros Mancoridis,, A Multidimensional Taxonomy of Software Development Environments,pp-10-15

Richard A. Pyne and Ezra K. Mugisa, Essential Elements of a Component-Based Development Environment for the software Supermarket,pp 3-10

Chris Luer and Andre van der Hoek, Composition Environments for Deployable Software Scomponentes, pp 3-20

Cuno Pfister 2001, Component Pascal,Oberon Microsystems, pp 3-13

Netbeans.org 2009, Netbeans IDE Java Tutorial, pp 1-20

Claudia Werner, Marco Mangan, OdysseyShare : an Environment for Collaborative Component-Based Development. pp 1-8

Daniel Lucrédio, Calebe de Paula Bianchini, Orion a Component Based Software, University of Sao Carlos Brazil. pp 5-15

Benedikt Schulz, Thomas Genssler, Alexander,1999, PECOS , Requirements for the Composition Environment, pp 10-16

Claudia Wemerl, Marco Mangan , 2003 ,OdysseyShare: an Environment for Collaborative Component-Based Development, pp 2-6

www.netbeans.org

Microsoft, 2008, www.microsoft.com/presspass/press/1998/feb98/rationpr.mspx

Tuma, 2001, //dsrg.mff.cuni.cz/teaching/seminars/2001-11-13-Tuma-Jiazzi.pdf