Universidad del Azuay

Facultad de Ciencias de la Administración

Escuela de Ingeniería de Sistemas

Trabajo de Graduación previo a la obtención del Título de

Ingeniero de Sistemas

Tema:

“Patrones de diseño para el desarrollo de software basados en la herramienta case Genexus”

Autores:

Juan Pablo Pintado Duchimaza

Felipe Javier Sarmiento Rodríguez

Director:

Ing. Pablo Pintado

Cuenca – Ecuador

2008

Dedicatoria

Dedicamos este trabajo a nuestros padres, por su inmenso cariño y por sus innumerables sacrificios para cultivar en nosotros una autentica educación, estructurar una personalidad y una conciencia de lucha y sacrificio por conseguir nuestros ideales. Además, por enseñarnos que el éxito no es un fin, sino un medio para llegar más lejos.

Agradecimientos

Queremos agradecer a Dios por darnos la oportunidad de cumplir uno de los objetivos de nuestra vida. También queremos agradecer a la institución ya que fue en donde no solo recibimos la educación si no compartimos momentos inolvidables con nuestros compañeros, a quienes también debemos agradecer por el constante apoyo que recibimos.

Deseamos demostrar nuestro leal agradecimiento al Ing. Pablo Pintado, coordinador de la tesis y al Ing. Demetrio Toledo, por su amistad, su aliento en los momentos difíciles y su apoyo decidido en la realización de este trabajo.

Responsabilidad de Autores

Las ideas vertidas en la presente tesis son de exclusiva responsabilidad de sus autores.

Juan Pablo Pintado Duchimaza

Felipe Javier Sarmiento Rodríguez

Indice de Contenidos

CAPITULO 2: Herramientas Case

CAPITULO 3: Patrones en la Herramienta Case Genexus

CAPITULO 4: Desarrollo De La Aplicación

Índice de Anexos

Resumen

La realización de sistemas en la plataforma web por lo general ha sido una tarea complicada para los programadores, ya que tenían que invertir demasiado tiempo en el diseño de la interface web. Por esa razón nació la idea de crear una herramienta que solucione estos inconvenientes, que solo hacían retrasar la entrega del sistema, y sin que este tenga la calidad esperada; los patrones son la esperanza para los programadores que quieren realizar una aplicación a gran velocidad sin tener que preocuparse en el diseño del sistema en la plataforma Web.

Los patrones es una herramienta para todos aquellos que quieren realizar aplicaciones o quieren migrar su antiguo sistema a la plataforma Web, ya que los patrones nos brindan la facilidad de aplicación, una gran cantidad de procedimientos, consultas y reportes auto programadas.

Abstrac

The systems design in Web platform generally has been a complex task for programmers, since they had to invest too much time in the design of Web interface. Thus was born to create a tool that solves these drawbacks, this problems made delay the delivery of the systems, and without the awaited quality; the patterns are the hope for the programmers whom they want to make an application at great speed without having to worry in the design of the system in the Web platform.

The patterns are a tool for all those that they want to make applications or they want to migrate his old system to the Web platform, since the patterns offer the application facility us, a great amount of procedures, programmed consultations and dynamic reports.

Introducción

Este documento presenta un detallado análisis del funcionamiento y elaboración de Patrones de Diseño para aplicaciones Web, para aplicarlo con la Herramienta Genexus 9.0. Además, se explicara minuciosamente cuáles son los pasos que se deben seguir para realizar un patrón de acuerdo a las necesidades que tenga el programador.

Se explicara y demostrara los beneficios que nos brinda los patrones con un ejemplo práctico en una aplicación que es muy utilizada en las empresas de nuestro medio.

Los patrones demostraran que son muy importantes cuando se requiere un sistema de gran calidad y en un plazo reducido, ya que estos nos brindan procedimientos, aplicaciones y reportes auto generados.

Los reportes que generan los patrones deben ser fáciles de manipular por los usuarios y además hacerlo para una herramienta que lo conozcan, con esto no limitamos a los usuarios a un estándar pre-establecido y ellos pueden realizar los cambios que crean convenientes y presentarlo en el formato que más les convenga.

Los patrones se presentan como la solución para reducir el tiempo que se dedica al diseño de la presentación de una aplicación web, brindando a los programadores la opción de invertir este tiempo en potenciar la aplicación.

CAPITULO 1

1. Definiciones

1.1. Introducción

El desarrollo de patrones ha surgido para apoyar el desarrollo de software, se puede decir que un patrón es una solución exitosa a un problema común, de esta manera una vez determinado el patrón a utilizar se reducirá el tiempo y el esfuerzo en la etapa de desarrollo del sistema actual como de futuros sistemas.

1.2. Justificación

El desarrollo de software es una tarea complicada, la cual depende en gran medida de la experiencia de las personas involucradas, en particular de los desarrolladores.

El 80% de los aportes vienen del 20% del personal.

La comprensión del software es uno de los problemas más complicados en la tarea de mantenimiento y evolución.

El hombre durante su historia ha dominado cierta técnica pasando por un proceso:

- Se realizan las operaciones de una manera artesanal. Los expertos aprenden por un proceso de ensayo y error y por transmisión de otros expertos.
- Se crea una ciencia alrededor de la tarea.
- Se desarrollan las técnicas generalmente aceptadas en el área.
- Se logra un conocimiento común sobre cómo aplicar las técnicas. Hay un metalenguaje común ente los expertos.

La sociedad requiere sistemas más complejos y más grandes. Los recursos para desarrollarlos cada vez son más escasos. Debe existir un mecanismo de reutilización.

El 80% del esfuerzo esta en el 20% del código desarrollado.

Las Tecnologías Orientadas a Objetos son las más utilizadas en los últimos años para el desarrollo de aplicaciones software. Se ha comprobado como este paradigma de programación presenta muchas ventajas.

Uno de los objetivos que se buscan al utilizar esta técnica es conseguir la reutilización. Entre los beneficios que se consiguen con la reutilización están:

1. Reducción de tiempos.
2. Disminución del esfuerzo de mantenimiento.
3. Eficiencia.
4. Consistencia.
5. Fiabilidad.
6. Protección de la inversión en desarrollos.

Entre los diferentes mecanismos de reutilización están:

Componentes: elemento de software suficientemente pequeño para crearse y mantenerse pero suficientemente grande para poder utilizarse.

Frameworks: bibliotecas de clases preparadas para la reutilización que pueden utilizar a su vez componentes.

Objetos distribuidos: paradigma que distribuye los objetos de cooperación a través de una red heterogénea y permite que los objetos interoperen como un todo unificado.

Patrones de diseño: Los patrones del diseño tratan los problemas del diseño que se repiten y que se presentan en situaciones particulares del diseño, con el fin de proponer soluciones a ellas. Por lo tanto, los patrones de diseño son soluciones exitosas a problemas comunes.

Sin embargo los mecanismos de reutilización también presentan algunos obstáculos:

1. El síndrome No Inventado Aquí: los desarrolladores de software no se suelen fiar de lo que no está supervisado por ellos mismos.
2. Acceso a las fuentes de componentes y frameworks.

3. Impedimentos legales, comerciales o estratégicos.
4. Formato de distribución de componentes (CORBA, ActiveX,).
5. Dilema entre reutilizar y rehacer.
6. Existe una inercia personal a los cambios.

1.3. Introducción a los Patrones

Los patrones como elemento de reutilización, comenzaron a utilizarse en la arquitectura con el objetivo de reutilizar diseños que se habían aplicado en otras construcciones y que se catalogaron como completos.

Chistopher Alexander fue el primero en intentar crear un formato específico para patrones en la arquitectura. Alexander argumenta que los métodos comunes aplicados en la arquitectura dan lugar a productos que no satisfacen las demandas y requerimientos de los usuarios y son ineficientes a la hora de conseguir el propósito de todo diseño y esfuerzo de la ingeniería: mejorar la condición humana. Alexander describe algunos diseños eternos para tratar de conseguir sus metas. Propone, así, un paradigma para la arquitectura basado en tres conceptos: la calidad, la puerta y el camino.

La Calidad: la esencia de todas las cosas vivas y útiles que nos hacen sentir vivos, nos da satisfacción y mejora la condición humana.

La Puerta: el mecanismo que nos permite alcanzar la calidad. Se manifiesta como un lenguaje común de patrones. La puerta es el conducto hacia la calidad.

El Camino (El Camino Eterno): siguiendo el camino, se puede atravesar la puerta para llegar a la calidad.

De este modo el patrón trata de extraer la esencia de ese diseño para que pueda ser utilizada por otros arquitectos cuando se enfrentan a problemas parecidos a los que resolvió dicho diseño.

Christopher Alexander da la siguiente definición de patrón:

"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma".

En definitiva se puede definir un patrón como *"una solución a un problema en un determinado contexto"*.

1.3.1 Desarrollo histórico

El término patrón se utiliza inicialmente en el campo de la arquitectura, por Christopher Alexander, a finales de los 70's. Este conocimiento es transportado al ámbito del desarrollo de software orientado a objetos y se aplica al diseño. De allí es extrapolado al desarrollo en general y a las demás etapas.

Un patrón de diseño es una abstracción de una solución en un nivel alto. Los patrones solucionan problemas que existen en muchos niveles de abstracción. Hay patrones que abarcan las distintas etapas del desarrollo; desde el análisis hasta el diseño y desde la arquitectura hasta la implementación.

Muchos diseñadores y arquitectos de software han definido el término de patrón de diseño de varias formas que corresponden al ámbito a la cual se aplican los patrones. Luego, se dividió los patrones en diferentes categorías de acuerdo a su uso.

Los diseñadores de software extendieron la idea de patrones de diseño al proceso de desarrollo de software. Debido a las características que proporcionaron los lenguajes orientados a objetos (como herencia, abstracción y encapsulamiento) les permitieron relacionar entidades de los lenguajes de programación a entidades del mundo real fácilmente, los diseñadores empezaron a aplicar esas características para crear soluciones comunes y reutilizables para problemas frecuentes que exhibían patrones similares.

1.3.2 Patrones de Software

Los patrones para el desarrollo de software son uno de los últimos avances de la Tecnología Orientada a Objetos. Los patrones son una forma literaria para resolver

problemas de ingeniería del software, que tienen sus raíces en los patrones de la arquitectura.

Los diseñadores y analistas de software más experimentados aplican de forma intuitiva algunos criterios que solucionan los problemas de manera elegante y efectiva. La ingeniería del software se enfrenta a problemas variados que hay que identificar para poder utilizar la misma solución (aunque matizada) con problemas similares.

Por otra parte, las metodologías Orientadas a Objetos tienen como uno de sus principios "no reinventar la rueda" para la resolución de diferentes problemas. Por lo tanto los patrones se convierten en una parte muy importante en las Tecnologías Orientadas a Objetos para poder conseguir la reutilización.

La ingeniería del software, tomando conceptos aplicados por primera vez en arquitectura, intenta construir patrones software para la resolución de problemas en dicho campo. Para conseguir esto debe existir una comunicación entre los distintos ingenieros para compartir los resultados obtenidos. Por tanto debe existir también un esquema de documentación con el objetivo de que la comunicación pueda entenderse de forma correcta. Esta comunicación no se debe reducir a la implementación, ya que únicamente fomenta el uso del "cortar y pegar".

Pueden referirse a distintos niveles de abstracción, desde un proceso de desarrollo hasta la utilización eficiente de un lenguaje de programación. El objetivo de los patrones es crear un lenguaje común a una comunidad de desarrolladores para comunicar experiencia sobre los problemas y sus soluciones.

1.3.2.1 Características de Los Patrones De Software

Hay que tener en cuenta que no todas las soluciones que tengan, en principio las características de un patrón son un patrón, sino que debe probarse que es una solución a un problema que se repite. Para que se pueda considerar un patrón, éste debe pasar por unas pruebas que reciben el nombre de test de patrones. Mientras tanto esa solución recibe el nombre de proto-patrón.

Solucionar un problema: los patrones capturan soluciones, no sólo principios o estrategias abstractas.

Ser un concepto probado: los patrones capturan soluciones demostradas, no teorías o especulaciones.

La solución no es obvia: muchas técnicas de solución de problemas tratan de hallar soluciones por medio de principios básicos. Los mejores patrones generan una solución a un problema de forma indirecta.

Describe participantes y relaciones entre ellos: los patrones no sólo describen módulos sino estructuras del sistema y mecanismos más complejos.

El patrón tiene un componente humano significativo: todo software proporciona a los seres humanos confort o calidad de vida (estética y utilidad).

Los patrones indican repetición, si algo no se repite, no es posible que sea un patrón. Pero la repetición no es la única característica importante. También necesitamos mostrar que un patrón se adapta para poder usarlo, y que es útil. **La repetición** es una característica cuantitativa pura, **la adaptabilidad** y **utilidad** son características cualitativas.

Así que aparte de la repetición, un patrón debe describir como la solución salda o resuelve sus objetivos, y porque está es una buena solución.

1.4 Clases de Patrones de Software

Existen diferentes ámbitos dentro de la ingeniería del software donde se pueden aplicar los patrones:

1. **Patrones de arquitectura:** expresa una organización o esquema estructural fundamental para sistemas de software. Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades, e incluye una guía para organizar las relaciones entre ellos.

2. **Patrones de diseño:** proporcionan un esquema para refinar los subsistemas o componentes de un sistema software, o las relaciones entre ellos. Describe estructuras repetitivas de comunicar componentes que resuelven un problema de diseño en un contexto particular.

3. **Patrones de programación (Idioms patterns):** un idioma es un patrón de bajo nivel de un lenguaje de programación específico. Describe como implementar aspectos de componentes o de las relaciones entre ellos utilizando las facilidades del lenguaje de programación dado.

4. **Patrones de análisis:** describen un conjunto de prácticas que aseguran la obtención de un buen modelo de un problema y su solución.

5. **Patrones organizacionales:** describen la estructura y prácticas de las organizaciones humanas, especialmente en las que producen, usan o administran software.

La diferencia entre estas clases de patrones está en los diferentes niveles de abstracción y detalle, y del contexto particular en el cual se aplican o de la etapa en el proceso de desarrollo. Así, los patrones de arquitectura son estrategias de alto nivel que involucran a los componentes, las propiedades y mecanismos globales de un sistema.

Los patrones de diseño son tácticas de media escala relacionados con la estructura y el comportamiento de entidades y sus relaciones. No influyen sobre toda la estructura del sistema, pero define micro arquitecturas de subsistemas y componentes. Los patrones de programación son específicos de las técnicas de un lenguaje de programación que afectan a partes pequeñas del comportamiento de los componentes de un sistema.

Los patrones de análisis se refieren a la etapa de análisis del ciclo de vida de construcción de software. Los patrones organizacionales describen la estructuración del personal en el desarrollo de software.

1.5 Patrones de Diseño

Los patrones de diseño ayudan a los diseñadores a reutilizar con éxito diseños para obtener nuevos diseños. Un diseñador que conoce algunos patrones puede aplicarlos inmediatamente a problemas de diseño sin tener que descubrirlos.

El objetivo de los patrones de diseño es guardar la experiencia en diseños de programas. Cada patrón de diseño nombra, explica y evalúa un importante diseño en los sistemas. Es decir se trata de agrupar la experiencia en diseño de una forma que la gente pueda utilizarlos con efectividad. Por eso se han documentado los más importantes patrones de diseño y presentado en catálogos.

Los patrones de diseño hacen más fácil reutilizar con éxito los diseños y arquitecturas. Expresando estas técnicas verificadas como patrones de diseño se hacen más accesibles para los diseñadores de nuevos sistemas. Los patrones de diseño ayudan a elegir diseños alternativos, que hacen un sistema reutilizable y evitan alternativas que comprometan la reutilización.

Los patrones de diseño pueden incluso mejorar la documentación y mantenimiento de sistemas existentes. Es decir, los patrones de diseño ayudan a un diseñador a conseguir un diseño correcto rápidamente.

1.5.1 Elementos de un patrón

En general, un patrón tiene cuatro elementos esenciales:

1. El **nombre del patrón** se utiliza para describir un problema de diseño, su solución, y consecuencias en una o dos palabras. Nombrar un patrón incrementa inmediatamente nuestro vocabulario de diseño. Esto nos permite diseños a un alto nivel de abstracción.

2. El **problema** describe cuando aplicar el patrón. Se explica el problema y su contexto. Esto podría describir problemas de diseño específicos tales como

algoritmos u objetos. Podría describir estructuras de clases u objetos que son síntomas de un diseño inflexible. Algunas veces el problema incluirá una lista de condiciones que deben cumplirse para poder aplicar el patrón.

3. La **solución** describe los elementos que forma el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño particular o implementación, porque un patrón es como una plantilla que puede ser aplicada en diferentes situaciones. En cambio, los patrones proveen una descripción abstracta de un problema de diseño y como una disposición general de los problemas de diseño lo soluciona.

4. Las **consecuencias** son los resultados de aplicar el patrón. Estas son muy importantes para la evaluación de diseños alternativos y para comprender los costes y beneficios de la aplicación del patrón.

1.5.2 Descripción de Patrones de Diseño

Para describir los patrones de diseño se utiliza un formato consistente. Cada patrón es dividido en secciones de acuerdo con la siguiente plantilla. La plantilla nos muestra una estructura uniforme para la información, de tal forma que los patrones de diseño sean fáciles de aprender, comparar y utilizar.

1. Nombre del patrón

Esta sección consiste de un nombre del patrón y una referencia bibliografía que indica de donde procede el patrón. El nombre es significativo y corto, fácil de recordar y asociar a la información que sigue.

2. Objetivo

Esta sección contiene unas pocas frases describiendo el patrón. El objetivo aporta la esencia de la solución que es proporcionada por el patrón. El objetivo esta dirigido a programadores con experiencia que pueden reconocer el patrón como uno que ellos ya conocen, pero para el cual ellos no le han dado un nombre. Después de reconocer el patrón por su nombre y objetivo, esto podría ser suficiente para comprender el resto de la descripción del patrón.

3. Contexto

La sección de Contexto describe el problema que el patrón soluciona. Este problema suele ser introducido en términos de un ejemplo concreto. Después de presentar el problema en el ejemplo, la sección de Contexto sugiere una solución de diseño a ese problema.

4. Aplicabilidad

La sección Aplicabilidad resume las consideraciones que guían a la solución general presentada en la sección Solución. En que situaciones es aplicable el patrón.

5. Solución

La sección Solución es el núcleo del patrón. Se describe una solución general al problema que el patrón soluciona. Esta descripción puede incluir, diagramas y texto que identifique la estructura del patrón, sus participantes y sus colaboraciones para mostrar como se soluciona el problema. Debe describir tanto la estructura dinámica como el comportamiento estático.

6. Consecuencias

La sección Consecuencias explica las implicaciones, buenas y malas, del uso de la solución.

7. Implementación

La sección de Implementación describe las consideraciones importantes que se han de tener en cuenta cuando se codifica la solución. También puede contener algunas variaciones o simplificaciones de la solución.

8. Código del ejemplo

Esta sección contiene el código del ejemplo que enseña una muestra de la implementación para un diseño que utiliza el patrón. En la mayoría de estos casos, este será el diseño descrito en la sección de Contexto.

9. Patrones relacionados

Esta sección contiene una lista de los patrones que están relacionados con el patrón que se describe.

1.5.3. Cualidades de un Patrón de Diseño

Encapsulación y abstracción: cada patrón encapsula un problema bien definido y su solución en un dominio particular. Los patrones deberían de proporcionar límites claros que ayuden a cristalizar el entorno del problema y el entorno de la solución empaquetados en un entramado distinto, con fragmentos interconectados. Los patrones también sirven como abstracciones las cuales contienen dominios conocidos y experiencia, y podrían ocurrir en distintos niveles jerárquicos de granularidad conceptual.

Extensión y variabilidad: cada patrón debería ser abierto por extensión o parametrización por otros patrones, de tal forma que pueden aplicarse juntos para solucionar un gran problema. Un patrón solución debería ser también capaz de realizar una variedad infinita de implementaciones (de forma individual, y también en conjunción con otros patrones).

Generatividad y composición: cada patrón, una vez aplicado, genera un contexto resultante, el cual concuerda con el contexto inicial de uno o más de uno de los patrones del catálogo. Está subsecuencia de patrones podría luego ser aplicada progresivamente para conseguir el objetivo final de generación de un "todo" o solución completa. Los patrones son aplicados por el principio de evolución fragmentada. Pero los patrones no son simplemente de naturaleza lineal, más bien esos patrones en un nivel particular de abstracción y granularidad podrían guiar hacia o ser compuestos con otros patrones para modificar niveles de escala.

Equilibrio: cada patrón debe realizar algún tipo de balance entre sus efectos y restricciones. Esto podría ser debido a uno o más de un heurístico que son utilizados para minimizar el conflicto sin el contexto de la solución. Las invariaciones representadas en un problema subyacente solucionan el principio o

filosofía para el dominio particular, y proveen una razón fundamental para cada paso o regla en el patrón.

1.5.4 Clasificación de los Patrones de Diseño

Dado que hay muchos patrones de diseño es necesario organizarlos

Patrones de Diseño Fundamentales

Los patrones de esta categoría son los más fundamentales e importantes patrones de diseño conocidos. Estos patrones son utilizados extensivamente en otros patrones de diseño.

Patrones de Creación

Los patrones de creación muestran la guía de cómo crear objetos cuando sus creaciones requieren tomar decisiones. Estas decisiones normalmente serán resueltas dinámicamente decidiendo que clases instanciar o sobre que objetos un objeto delegará responsabilidades.

La valía de los patrones de creación nos dice como estructurar y encapsular estas decisiones.

Patrones de Partición

En la etapa de análisis, el levantamiento de información para identificar los actores, casos de uso, requerimientos y las relaciones que constituyen el problema. Los patrones de esta categoría proveen la guía sobre como dividir actores complejos y casos de uso en múltiples clases.

Patrones Estructurales

Los patrones de esta categoría describen las formas comunes en que diferentes tipos de objetos pueden ser organizados para trabajar unos con otros.

Patrones de Comportamiento

Los patrones de este tipo son utilizados para organizar, manejar y combinar comportamientos. Los patrones de comportamiento estudian las relaciones de llamadas entre los diferentes objetos, normalmente ligados con la dimensión temporal.

Patrones de Concurrencia

Los patrones de esta categoría permiten coordinar las operaciones concurrentes. Estos patrones se dirigen principalmente a dos tipos diferentes de problemas:

Recursos compartidos: Cuando las operaciones concurrentes acceden a los mismos datos u otros tipos de recursos compartidos, podría darse la posibilidad de que las operaciones interfirieran unas con otras si ellas acceden a los recursos al mismo tiempo. Para garantizar que cada operación se ejecuta correctamente, la operación debe ser protegida para acceder a los recursos compartidos en solitario. Sin embargo, si las operaciones están completamente protegidas, entonces podrían bloquearse y no ser capaces de finalizar su ejecución.

Secuencia de operaciones: Si las operaciones son protegidas para acceder a un recurso compartido una cada vez, entonces podría ser necesario garantizar que ellas acceden a los recursos compartidos en un orden particular. Por ejemplo, un objeto nunca será borrado de una estructura de datos antes de que esté sea añadido a la estructura de datos.

1.5.5 Patrones de Diseño, Reglas y Creatividad

La presencia combinada de todos los elementos de un patrón y cualidades es lo que hace que los patrones sean más que simples heurísticos, reglas o algoritmos. Los heurísticos y principios participan frecuentemente en los objetivos y/o razón fundamental de un patrón, pero solamente son elementos de un patrón.

Un patrón es el proceso que genera una solución, pero podría generar un inmenso número de soluciones variantes (imaginablemente sin repetir la misma solución dos veces). El elemento humano de los patrones es el que principalmente contribuye a su variabilidad y adaptabilidad, y normalmente necesita un gran grado de creatividad en sus aplicaciones y combinaciones. Así que, como el proceso de arquitectura y diseño son propósitos creativos, así también es la aplicación de patrones.

1.5.6 Patrones y Algoritmos

La sección titulada patrones de diseño, reglas y creatividad también se aplica en gran parte a los algoritmos y sus estructuras de datos. Ciertamente, los algoritmos y estructuras de datos pueden ser empleados en la implementación de uno o varios patrones, pero los algoritmos y las estructuras de datos generalmente solucionan problemas computacionales más concretos como ordenación y búsqueda. Los patrones son típicamente interesantes en arquitecturas de gran extensión, que tienen efectos de gran escala. Los patrones de diseño solucionan a la gente y desarrolladores cuestiones como mantenibilidad, reusabilidad, comunicabilidad y encapsulación.

Los algoritmos y estructuras de datos son normalmente relacionados casi exclusivamente con el contexto de optimización o tiempo, o algunos otros aspectos de la computación compleja y recursos en uso. Tratan de encontrar la más compacta y eficiente solución que realiza alguna computación importante o almacenar y recordar sus resultados.

1.5.7 Como usar un Patrón de Diseño

Una vez seleccionado el patrón de diseño que se va a aplicar hay que tener en cuenta las siguientes claves para utilizar el patrón:

Leer el patrón de diseño por encima. Prestar una atención particular a las secciones "Aplicabilidad" y "Consecuencias" para asegurarse de que el patrón es correcto para plantear una solución al problema.

Observar la estructura, los elementos que participan en el patrón y las colaboraciones entre ellos. Asegurarse de comprender las clases y objetos del patrón y como se relacionan.

Mirar la sección "Código del ejemplo" para ver un ejemplo concreto del patrón en código. Estudiar el código enseñará como implementar el patrón.

Escoger nombres significativos de los elementos que participan en el patrón para el contexto de la aplicación. Los nombres de los elementos de un patrón son normalmente demasiado abstractos para aparecer directamente en una aplicación. No obstante, es muy útil incorporar los nombres de los participantes en el nombre que

aparece en la aplicación. Esto ayudará a tener el patrón más explícito en la implementación.

1.5.8 El Futuro de los Patrones

La gran popularidad que han ido adquiriendo los patrones hace que la comunidad software los utilice y los soporte comúnmente. Varias notaciones usadas por las Tecnologías Orientadas a Objetos han añadido soporte para el modelado y representación de patrones de diseño. Muchas herramientas de desarrollo de software han añadido un soporte similar. Algunos proyectos de investigación están intentando codificar patrones de diseño con el propósito de generar código fuente.

Hay una especulación a cerca de sí los patrones sustituirán algún día a los programadores. Esta misma especulación ya ha aparecido en algunas ocasiones con la introducción de nuevas tecnologías. Los lenguajes y las herramientas que se usan para solucionar problemas software evolucionarán, pero se seguirá necesitando desarrolladores con la misma evolución.
Aunque la habilidad de codificar patrones como componentes de software pueda llegar a ser importante, más importante será el conocimiento de cómo y cuando aplicar y combinar patrones, en conjunción con la habilidad de usar un vocabulario de nombres de patrones para comunicarse con otros desarrolladores.

1.6. Conclusiones

El uso de los patrones permite aumentar la calidad del software ya que se basan en la experiencia en la resolución de problemas similares obteniendo además la reducción de tiempos ya que al fomentar la reutilización evitaremos hacer algo que ya esta echo y probado.
Es importante determinar bien el patrón que se desea utilizar para resolver un determinado problema ya que de esta manera se los resultados obtenidos serán óptimos.

GLOSARIO 1

Frameworks: bibliotecas de clases preparadas para la reutilización que pueden utilizar a su vez componentes.

Abstracción: permite concentrarse en un problema a nivel de generalización independiente de detalles de nivel interior

Encapsulamiento: de datos y operaciones que trabajan sobre un único paquete

Referencias

Libros

- UML y Patrones. Introducción al análisis y diseño orientado a objetos - Larman - Prentice Hall

- Design Patterns. Elements of Reusable Object-Oriented Software - Gamma, Helm, Johnson, Vlissides - Addison Wesley

Sitios de Internet

- http://www.vbdotnetheaven.com/Code/Jun2003/2014.asp.
- http://blogs.vbcity.com/jspano/articles/198.aspx.

CAPITULO 2: HERRAMIENTAS CASE

2.1 Introducción

Las Herramientas de Ayuda al Desarrollo de Sistemas de Información, surgieron para intentar dar solución a los problemas inherentes a los proyectos de generación de aplicaciones informáticas: plazos y presupuestos incumplidos, insatisfacción del usuario, escasa productividad y baja calidad de los desarrollos. Algunas de estas herramientas se dirigen principalmente a mejorar la calidad, como es el caso de las herramientas CASE (Computer Aided Software Engineering- Ingeniería de Software Asistida por Computadora). Otras van dirigidas a mejorar la productividad durante la fase de construcción, como es el caso de los lenguajes de cuarta generación.

2.2 Característica de las Herramientas Case

Las Herramientas Case es un software que facilita la producción de aplicaciones a la medida. Existe una amplia gama de posibles productos que se pueden incluir en esta definición, desde los lenguajes de programación, hasta sofisticados y complejos productos como las herramientas CASE.

Se consideran herramientas de ayuda al desarrollo a:

- Herramientas de ingeniería de software asistida por computadora o CASE.
- Lenguajes de cuarta generación o 4GL.
- Otras herramientas: gestión de proyectos, gestión de la configuración, ayuda en las pruebas, control de calidad, bibliotecas de clases de objetos, etc.

Es importante puntualizar que las fronteras entre unas y otras no siempre están claramente definidas.

Las herramientas Case son un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo del sistema de información, completamente o en algunas fases, son un conjunto de:

- Utilidad.
- Métodos.
- Técnicas.

También pueden mejorar la productividad en el desarrollo de una aplicación de bases de datos. Y por productividad se entiende tanto la eficiencia en el desarrollo, como la efectividad del sistema desarrollado.

La eficiencia se refiere al costo, tanto en tiempo como en dinero, de desarrollar la aplicación.

La efectividad se refiere al grado en que el sistema satisface las necesidades de los usuarios. Para obtener una buena productividad, subir el nivel de efectividad puede ser más importante que aumentar la eficiencia.

2.3 Objetivos de las Herramientas Case

1. Mejorar la productividad en el desarrollo y mantenimiento del software.
2. Aumentar la calidad del software.
3. Mejorar el tiempo y costo de desarrollo y mantenimiento de los sistemas informáticos.
4. Mejorar la planificación de un proyecto.
5. Aumentar la biblioteca de conocimiento informático de una empresa ayudando a la búsqueda de soluciones para los requisitos.
6. Automatizar, desarrollo del software, documentación, generación de código, pruebas de errores y gestión del proyecto.
7. Ayuda a la reutilización del software, portabilidad y estandarización de la documentación.
8. Gestión global en todas las fases de desarrollo de software con una misma herramienta.
9. Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

2.4 Clasificaciones de las Herramientas Case

No existe una única clasificación de herramientas CASE y, en ocasiones, es difícil incluirlas en una clase determinada. Podrían clasificarse atendiendo a:

1. Las plataformas que soportan.
2. Las fases del ciclo de vida del desarrollo de sistemas que cubren.
3. La arquitectura de las aplicaciones que producen.
4. Su funcionalidad.

2.4.1 Las fases del ciclo de vida del desarrollo de sistemas que cubren

Las herramientas CASE, en función de las fases de ***ciclo de vida*** que abarca, se pueden agrupar de la forma siguiente:

1. **Herramientas integradas, I-CASE** (Integrated CASE, CASE integrado): abarcan todas las fases del ciclo de vida del desarrollo de sistemas. Son llamadas también CASE workbench.

   Las herramientas I-CASE se basan en una metodología. Tienen un repositorio y aportan técnicas estructuradas para todas las fases del ciclo de vida. Estas son las características que les confieren su mayor ventaja: una mejora de la calidad de los desarrollos. Sin embargo, no todas ellas son modernas en el sentido de aprovechar la potencia de las estaciones de trabajo o la utilización de lenguajes de alto nivel o técnicas de prototipo.

2. **Herramientas de bajo nivel, L-CASE** (Lower CASE - CASE inferior) o back-end, dirigidas a las últimas fases del desarrollo: en la construcción e implantación de un sistema informático.
3. **Juegos de herramientas o toolkits**, son el tipo más simple de herramientas CASE. Automatizan una fase dentro del ciclo de vida. Dentro de este grupo se encontrarían las herramientas de reingeniería, orientadas a la fase de mantenimiento.

2.4.2 Su Funcionalidad

Otra posible clasificación, utilizando la ***funcionalidad*** como criterio principal, es la siguiente:

1. **Herramientas de planificación de sistemas de gestión:** Sirven para modelar los requisitos de información estratégica de una organización. Proporcionan un "Meta-Modelo" del cual se pueden obtener sistemas de información específicos. Su objetivo principal es ayudar a comprender mejor cómo se mueve la información entre las distintas unidades organizativas. Estas herramientas proporcionan una ayuda importante cuando se diseñan nuevas estrategias para los sistemas de información y cuando los métodos y sistemas actuales no satisfacen las necesidades de la organización.

2. **Herramientas de análisis y diseño:** Permiten al desarrollador crear un modelo del sistema que se va a construir y también la evaluación de la validez y consistencia de este modelo. Proporcionan un grado de confianza en la representación del análisis y ayudan a eliminar errores con anticipación. Se tienen:

   a. Herramientas de análisis y diseño (Modelamiento).
   b. Herramientas de creación de prototipos y de simulación.
   c. Herramientas para el diseño y desarrollo de interfaces.
   d. Máquinas de análisis y diseño (Modelamiento).

3. **Herramientas de programación:** Se engloban aquí los compiladores, los editores y los depuradores de los lenguajes de programación convencionales. Ejemplos de estas herramientas son:

   a. Herramientas de codificación convencionales.
   b. Herramientas de codificación de cuarta generación.
   c. Herramientas de programación orientadas a los objetos.

4. **Herramientas de integración y prueba:** Sirven de ayuda a la adquisición, medición, simulación y prueba de los equipos lógicos desarrollados. Entre las más utilizadas están:

   a. Herramientas de análisis estático.
   b. Herramientas de codificación de cuarta generación.
   c. Herramientas de programación orientadas a los objetos.

5. **Herramientas de gestión de prototipos:** Los prototipos son utilizados ampliamente en el desarrollo de aplicaciones, para la evaluación de especificaciones de un sistema de información, o para un mejor entendimiento de cómo los requisitos de un sistema de información se ajustan a los objetivos perseguidos.

6. **Herramientas de mantenimiento:** La categoría de herramientas de mantenimiento se puede subdividir en:
   a. Herramientas de ingeniería inversa.
   b. Herramientas de reestructuración y análisis de código.
   c. Herramientas de reingeniería.

7. **Herramientas de gestión de proyectos.** La mayoría de las herramientas CASE de gestión de proyectos, se centran en un elemento específico de la gestión del proyecto, en lugar de proporcionar un soporte global para la actividad de gestión. Utilizando un conjunto seleccionado de las mismas se puede: realizar estimaciones de esfuerzo, costo y duración, hacer un seguimiento continuo del proyecto, estimar la productividad y la calidad, etc. Existen también herramientas que permiten al comprador del desarrollo de un sistema, hacer un seguimiento que va desde los requisitos del pliego de prescripciones técnicas inicial, hasta el trabajo de desarrollo que convierte estos requisitos en un producto final. Se incluyen dentro de las herramientas de control de proyectos las siguientes:
   a. Herramientas de planificación de proyectos.
   b. Herramientas de seguimiento de requisitos.
   c. Herramientas de gestión y medida.

8. **Herramientas de soporte:** Se engloban en esta categoría las herramientas que recogen las actividades aplicables en todo el proceso de desarrollo, como las que se relacionan a continuación:
   a. Herramientas de documentación.
   b. Herramientas para software de sistemas.
   c. Herramientas de control de calidad.
   d. Herramientas de bases de datos.

2.4.3 Otra Clasificaciones.

Otra clasificación, diferencia las funciones CASE en cinco grupos:

1. **Repositorio:** Funcionan en torno a un repositorio central, siendo éste el núcleo fundamental que contiene todas las definiciones de los objetos y sus

relaciones. Los objetos pueden ser especificaciones del sistema en forma de diagramas de flujo de datos, diagramas entidad-relación, esquemas de bases de datos, diseños de pantallas, etc. El repositorio es un concepto más amplio que el de diccionario de datos y soporta a los demás grupos de funciones. No es fácil encontrar en el mercado productos CASE con funcionalidades estrictamente a las de repositorio, ya que, a pesar de su innegable importancia, tienen un carácter auxiliar de los demás grupos de funciones. Cualquier sistema CASE poseerá un repositorio propio o bien, trabajará sobre un repositorio suministrado por otro fabricante o vendedor.

2. **Reingeniería:** Los sistemas CASE permiten establecer una relación estrecha y fuertemente formada entre los productos generados a lo largo de distintas fases del ciclo de vida, permitiendo actuar en el sentido especificaciones-código (ingeniería "directa") y también en el contrario (ingeniería "inversa"). Ello facilita la realización de modificaciones en la fase más adecuada en cada caso y su traslado a las demás. Al conjunto de facilidades proporcionadas por la ingeniería "directa" e "inversa" se le denomina "reingeniería".

3. **Soporte del ciclo de vida:** El ciclo de vida de una aplicación o de un sistema de información se compone de varias etapas, que van desde la planificación de su desarrollo hasta su implantación, mantenimiento y actualización. Aunque el número de fases puede ser variable en función del nivel de detalle que se adopte, pueden de modo simplificado, identificarse las siguientes:

   a. Planeamiento.
   b. Análisis y Diseño.
   c. Implementación (programación y pruebas).
   d. Mantenimiento y actualización.

   Los sistemas CASE pueden cubrir la totalidad de estas fases o bien especializarse en algunas de ellas. En este último caso se pueden distinguir sistemas de "alto nivel" ("Upper Case"), orientados a la autonomía y soporte de las actividades correspondientes a las dos primeras fases y, sistemas de "bajo nivel" ("Lower Case"), dirigidos hacia las dos últimas. Los sistemas de

"alto nivel" pueden soportar un número más o menos amplio de metodologías de desarrollo.

4. **Soporte de proyecto:** Este tipo de funciones hace referencia al soporte de actividades que se producen durante el desarrollo, derivadas fundamentalmente del trabajo en grupos, tales como facilidades de comunicación, soporte a la creación, modificación e intercambio de documentación, herramientas personales, controles de seguridad, etc. Los sistemas CASE pueden conceder a estas cuestiones una importancia variable por lo cual el soporte de proyecto constituye un factor de diferenciación.

5. **Mejora continua de calidad:** Aunque frecuentemente se asocia a los sistemas CASE con la mejora de la productividad en el desarrollo de aplicaciones, debe tenerse en cuenta que una de las principales ventajas estriba también, en la mejora de la calidad de los desarrollos realizados. Determinados sistemas CASE enfatizan más sobre este punto que sobre el anterior, introduciendo herramientas que permiten ejercer un control intenso de garantía de calidad del software desarrollado desde las primeras fases de su ciclo de vida.

2.5 Ventajas y Desventajas de las Herramientas Case

2.5.1 Ventajas

Entre los beneficios o ventajas ofrecidos por la tecnología CASE se encuentran los siguientes:

1. **Facilidad para la revisión de aplicaciones**

   La experiencia muestra que una vez que las aplicaciones se implementan, se emplean por mucho tiempo. Las herramientas CASE proporcionan un beneficio substancial para las organizaciones al facilitar la revisión de las aplicaciones. Contar con un depósito central agiliza el proceso de revisión ya que éste proporciona bases para las definiciones y estándares para los datos. Las capacidades de generación interna, si se encuentran presentes,

contribuyen a modificar el sistema por medio de las especificaciones más que por los ajustes al código fuente.

2. **Soporte para el desarrollo de prototipos de sistemas**

   En general, el desarrollo de prototipos de aplicaciones toma varias formas. En ocasiones se desarrollan diseños para pantallas y reportes con la finalidad de mostrar la organización y composición de los datos, encabezados y mensajes. Los ajustes necesarios al diseño se hacen con rapidez para alterar la presentación y las características de la interface. Sin embargo, no se prepara el código fuente, de naturaleza orientada hacia procedimientos, como una parte del prototipo.

   Como disyuntiva, el desarrollo de prototipos puede producir un sistema que funcione. Las características de entrada y salida son desarrolladas junto con el código orientado hacia los procedimientos y archivos de datos.

   Muchas herramientas CASE soportan las primeras etapas del desarrollo del prototipo. Muy pocas brindan apoyo durante todo el proceso de desarrollo del prototipo. Las que proporcionan la capacidad para generar código soportan de hecho todo proceso, ya que el código puede ser generado al inducir la actividad de generación después de cambiar las especificaciones o requerimientos.

3. **Generación de código**

   Como ya se mencionó, algunas herramientas CASE tienen la capacidad de producir el código fuente. La ventaja más visible de esta característica es la disminución del tiempo necesario para preparar un programa. Sin embargo, la generación del código también asegura una estructura estándar y consistente para el programa (lo que tiene gran influencia en el mantenimiento) tenga menor ocurrencia de varios tipos de errores, mejorando de esta manera la calidad. Las características de la generación del código permiten volver a utilizar el software y las estructuras estándares para generar dicho código, así como el cambio de una especificación modular, lo que significa volver a generar el código y los enlaces con otros módulos. Ninguna de las herramientas que existen en el presente es capaz de generar un código completo en los dominios.

4. **Mejora en la habilidad para satisfacer los requerimientos del usuario**

   Es bien conocida la importancia de satisfacer los requerimientos del usuario, ya que esto guarda relación con el éxito del sistema. De manera similar, tener los requerimientos correctos mejora la calidad de las prácticas de desarrollo. Parece ser que las herramientas CASE disminuyen el tiempo de desarrollo, una característica que es importante para los usuarios. Las herramientas afectan la naturaleza y cantidad de interacción entre los encargados del desarrollo y el usuario. Las descripciones gráficas y los diagramas, así como los prototipos de reportes y la composición de las pantallas, contribuyen a un intercambio de ideas más efectivo.

5. **Soporte interactivo para el proceso de desarrollo**

   La experiencia ha demostrado que el desarrollo de sistemas es un proceso interactivo. Las herramientas CASE soportan pasos interactivos al eliminar el tedio manual de dibujar diagramas, elaborar catálogos y clasificar. Como resultado de esto, se anticipa que los analistas repasarán y revisarán los detalles del sistema con mayor frecuencia y en forma más consistente.

2.5.2 Desventajas

Las herramientas CASE tienen puntos débiles significativos, que van desde la confiabilidad en los métodos estructurados hasta su alcance limitado, los cuales amenazan con minar los beneficios potenciales descritos con anterioridad.

1. **Confiabilidad en los métodos estructurados**

   Muchas herramientas CASE están construidas teniendo como base las metodologías del análisis estructurado y del ciclo de vida de desarrollo de sistemas. Por si sola, esta característica puede convertirse en la principal limitante ya que no todas las organizaciones emplean métodos de análisis estructurado.

   Los métodos estructurados, introducidos en la década de los setenta, fueron muy elogiados por su habilidad para mejorar la exactitud de los

requerimientos específicos de las aplicaciones. El nivel de conocimiento de los métodos estructurados es alto entre los profesionales de sistemas de información – de acuerdo con algunas estimaciones, casi el 90% de todos los analistas están familiarizado con estos métodos -. Aproximadamente la mitad de todas las organizaciones en Estados Unidos han utilizado alguna vez estos métodos. A pesar de lo anterior, si la organización o el analista no utilizan los métodos propios del análisis estructurado y tampoco desean considerar su uso, entonces el valor del CASE disminuye. En algunos casos, los analistas evitan del todo emplear herramientas CASE.

2. **Falta de niveles estándar para el soporte de la metodología**

Aún no aparece un conjunto "estándar" de herramientas CASE. Por tanto, debe tener precaución al seleccionar una herramienta de este tipo.

Existen dos significados para las palabras "soporte de la metodología". Una herramienta puede:

1) dar soporte a los diagramas que emplea una metodología

2) soportarlos e imponer la metodología, sus reglas y procesos.

Las herramientas CASE que existen en el presente, tienen una de las siguientes características:

a. Son independientes de la metodología.
b. Permiten que los usuarios definan sus propias metodologías.
c. Soportan una metodología.
d. Soportan las metodologías más diseminadas.

En todas ellas existen ciertos compromisos. Las herramientas que son independientes de la metodología, no pueden fomentar el uso de las reglas y estándares de la misma. Estas herramientas quizá proporcionen los componentes de una metodología (por ejemplo: diagramas de flujos de datos, un diccionario de datos y facilidades para la descripción de procesos), pero no el marco de referencia, reglas y procedimientos que en realidad constituyen el núcleo de la metodología. Aunque se puede llevar a cabo acciones básicas

para la validación de diseños y diagramas para detectar componentes faltantes, éstas son sólo funciones mecánicas. Por otra parte, esta clase de herramientas no puede proporcionar ayuda metodológica o pedir al usuario que realice tareas necesarias para la metodología que aún está sin terminar.

Estas herramientas mejoran la productividad al efectuar tareas tediosas y de documentación, aunque ellas no puedan asegurar buenos resultados. Desde el punto de vista funcional, las capacidades que brindan para garantizar la calidad son mínimas.

3. **Conflictos en el uso de los diagramas**

Las herramientas difieren en el uso que hacen los diagramas. Algunas son herramientas exclusivamente para gráficas, que se aproximan al dibujo de diagramas para el análisis de entrada y salida de datos. Este tipo de herramientas puede restringir ya sea el proceso de desarrollo normal seguido por una organización o el estilo particular de trabajo de los analistas.

Otros vendedores de herramientas consideran los diagramas como documentación y aceptan entradas por medio de formas o lenguajes de especificación y, en ocasiones, en forma gráfica. Por tanto, se debe tener cuidado cuando se selecciona una herramienta para apoyar los métodos existentes en una organización.

4. **Diagramas no utilizados**

En general, los productos CASE emplean gráficas para modelar y generar informes sobre el análisis y desarrollo de sistemas. Una de las afirmaciones de los vendedores de herramientas es que las presentaciones gráficas y la documentación mejoran la comunicación entre los miembros del equipo de desarrollo, propician una calidad mayor de la entrada proporcionada por el cliente y mejoran la productividad de desarrollo de software. Sin embargo, los investigadores han encontrado que, en algunos casos, las herramientas gráficas, automatizadas o manuales, no se emplean del todo. O tal vez no se utilicen en la forma que deberían emplearse. Por otra parte, algunos analistas prefieren para algunas tareas un lenguaje estructurado o descriptivo.

Muchos profesionales de los sistemas de información no hacen uso de herramientas gráficas en el desarrollo de software; más bien las emplean para automatizar la producción de informes y documentación del sistema, como los diagramas de flujo utilizados por los programadores para documentar un programa una vez terminado.

5. **Función limitada**

Aunque una herramienta puede apoyar varias fases del ciclo de vida de desarrollo de sistemas o adaptarse a diferentes metodologías de desarrollo, por lo general su enfoque primario está dirigido hacia una fase o método especifico. Por ejemplo, los encargados de desarrollar un nuevo producto pueden afirmar que éste apoya todo el proceso de análisis y diseño. Sin embargo, las capacidades de comprobación y verificación de errores del producto quizá sean más rigurosas ya sea en el área de análisis o en la de diseño, pero no en ambas. Algunos productos están dirigidos hacia el diseño de bases de datos para la organización y al desarrollo de aplicaciones que giren en torno a la base de datos, omitiendo el soporte para pantallas de presentación visual, los informes sobre requerimientos o las necesidades de seguridad. Algunos productos capaces de generar el código hacen mayor hincapié en el desarrollo de prototipos como el principal método de desarrollo de sistemas de información. Muchas herramientas para la fase de desarrollo recalcan el mantenimiento y la reestructuración del código, pero ofrecen un soporte débil durante la fase de análisis para la determinación y especificación de requerimientos.

6. **Alcance limitado**

Aunque muchas herramientas basadas en computadoras incluyen la capacidad de verificar las especificaciones para determinar su complementes o consistencia, virtualmente no llevan a cabo ningún análisis de los requerimientos de la aplicación. Por tanto, el alcance de las actividades de desarrollo asociado con las herramientas existentes es bastante limitado.

La mayor parte de productos CASE describe (documenta) pero no analiza. De poca ayuda es proporcionar una regla de inclusión en los mejores enfoques y

una regla de exclusión para los que son poco satisfactorios. No ofrecen o evalúan, soluciones potenciales para los problemas relacionados con sistemas. Y tampoco existe una garantía clara para que dos analistas que utilicen los mismos métodos aplicados a información idéntica, formulen recomendaciones igualmente aceptables.

2.6 Conclusiones

Sin lugar a dudas las herramientas CASE han venido a revolucionar la forma de automatizar los aspectos clave en el desarrollo de los sistemas de información, debido a que éstas, brindan toda una gama de componentes que incluyen todas o la mayoría de los requisitos necesarios para el desarrollo de los sistemas, han sido creadas con una gran exactitud en torno a las necesidades de los desarrolladores de sistemas para la automatización de procesos incluyendo el análisis, diseño e implantación.

Las Herramientas CASE se clasifican por su amplitud en: TOOLKIT, WORKBENCH además también se pueden dividir teniendo en cuenta las fases del ciclo de vida que automatizan: UPPER CASE, MIDDLE CASE, LOWER CASE.

Desde que se crearon éstas herramientas (1984) hasta la actualidad, las CASE cuentan con una credibilidad y exactitud que tienen un reconocimiento universal, siendo usadas por cualquier desarrollador y / o programador que busca un resultado óptimo y eficiente, pero sobre todo que busca esa minuciosidad necesaria de los procesos y entre los procesos.

La principal ventaja de la utilización de una herramienta CASE, es la mejora de la calidad de los desarrollos realizados y, en segundo término, el aumento de la productividad.

GLOSARIO 2

CASE: Computer Aided Software Engineering/Ingeniería de Software Asistida por Computadora.

4GL: Lenguajes de cuarta generación.

I-CASE: CASE integrado.

L-CASE: CASE inferior.

Toolkits: Juegos de herramientas CASE.

Referencias

Libros

Castañeda G., Víctor. Revista Tecnología de Punta.

Herramientas para el desarrollo de Sistemas de Información. http://www.inei.gob.pe/cpi/bancopub/libfree. Instituto de Estadística e Informática. Lima Perú. Julio, 1997.

Sitios de Internet

http://www.geocities.com/SiliconValley/lab/7538/

http://es.wikipedia.org/wiki/Herramienta_CASE

http://ceds.nauta.es/Catal/Products/caselist2.htm

http://www3.uji.es/~mmarques/f47/apun/node75.html

http://www.iscmolina.com/Herramientas%20CASE.html

CAPITULO 3: PATRONES EN LA HERRAMIENTA CASE GENEXUS

3.1 Introducción

Este capítulo nos mostrara la importancia de realizar patrones para la herramienta Genexus 9, ya que esta nos brinda la facilidad de aplicar el patrón y una gama de poderosas herramientas para modificarlo cuando lo requiramos.

Gracias a estas herramientas es que los patrones tienen un valor agregado incrementando su valor y potencial para ayudar y mejorar el desempeño del programador sin tener que preocuparse en el diseño final del sistema.

3.2 ¿Cómo funciona los Patrones con la Herramienta Genexus?

Genexus 9.0 fue diseñado con el objetivo de brindar un aumento en la productividad de un orden de magnitud que el programador pueda lograr por unidad de tiempo. Para ello incorpora la herramienta Patterns que es un marco que permite generar código GeneXus a partir de patrones generados. Esto permite automatizar gran parte del trabajo repetitivo que se realiza creando objetos dentro de la propia Base de Conocimiento (KB), antes de generar la aplicación en la DBMS y en algún lenguaje, se pueden crear y adaptar Patterns a las necesidades de cada proyecto y requerimiento de los clientes. Asimismo, existen Patterns creados por Artech a disposición de todos los usuarios GeneXus. Algunos escenarios de uso y su correspondiente Pattern:

Migración de aplicaciones a Web – Pattern “WorkWith”

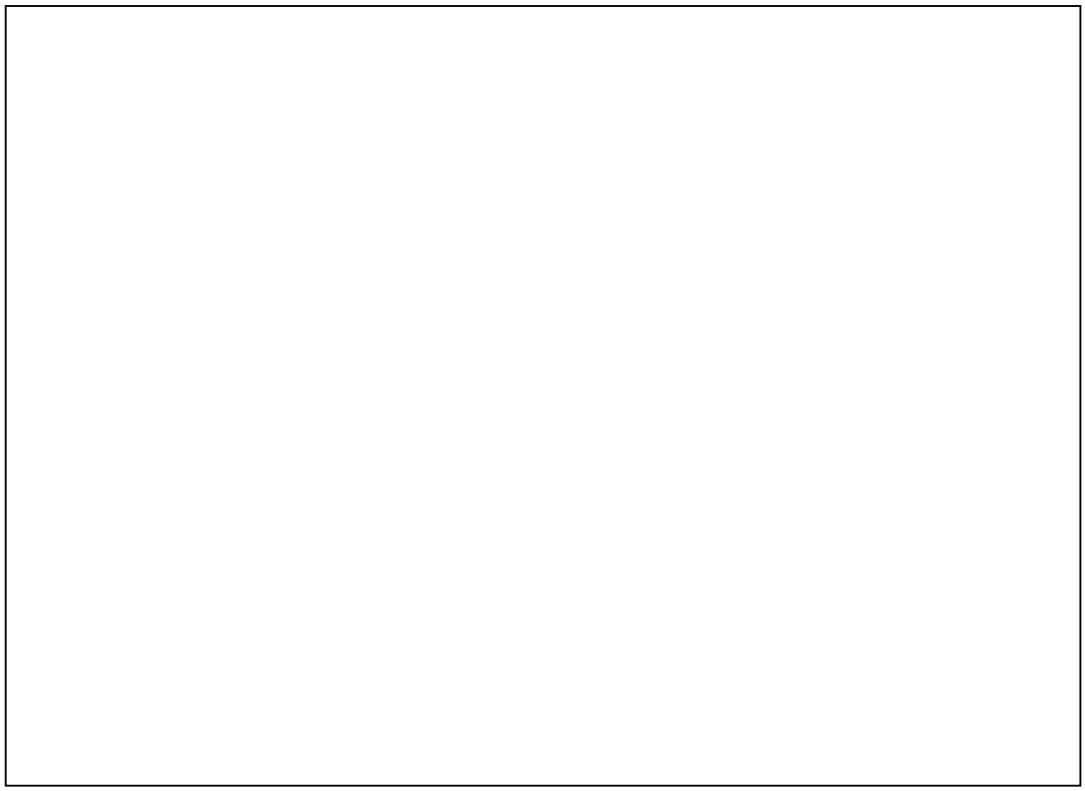
Parametrización de aplicaciones en tiempo de ejecución – Pattern OAV (Object Attribute Value)

Diseño de modelo de datos con características específicas – Pattern “Bill of Materials”

3.3 Facilidades de interacción con el Patrón

Se pude aplicar el patrón en una Base del Conocimiento ya exístete o se puede crear una nueva Base de Conocimiento para lo cual se utiliza la herramienta Patterns de Artech en donde se debe elegir el patrón que mejor se adapte a aplicación a desarrollar, una vez seleccionado el patrón le aplicaremos a las transacciones generadas en Genexus.

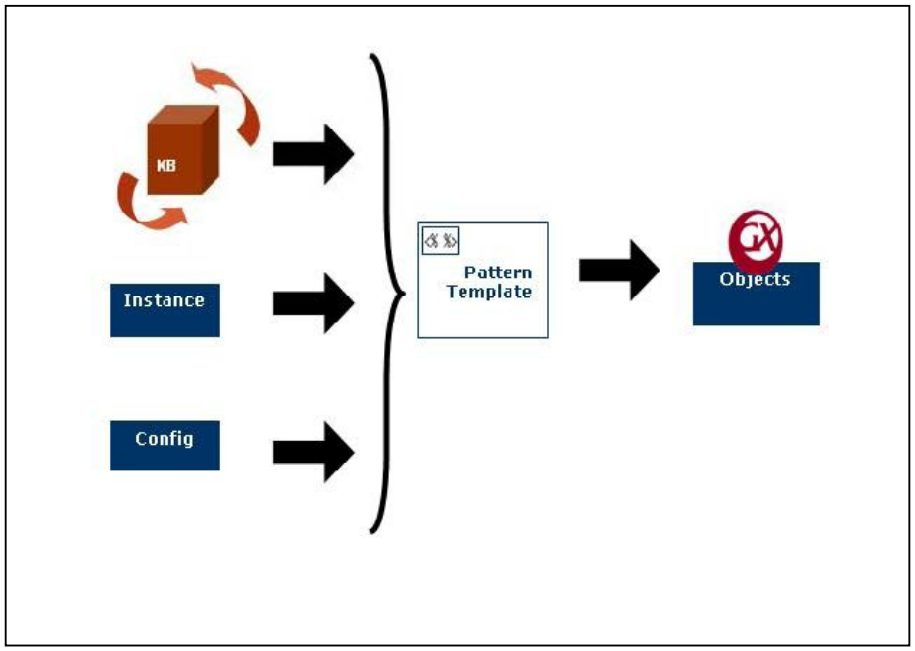
Una vez aplicado el Patrón, el programador puede realizar cambios en los objetos generados en caso de ser necesario o aumentar objetos en la base de Conocimiento dependiendo de las necesidades de la aplicación. Además Genexus nos brinda una herramienta poderosa llamada THEME EDITOR 2.0, que nos ayuda a modificar cualquier parte de patrón en lo que se refiere al diseño visual. Se podrán cambiar colores de botones, tipo de letras y otros cambios puntuales en los que el cliente este necesitando. Eso hace que el patrón generado nos ayude en la mayor parte de la implementación y además sea modificado cuando se guste, gracias a esto tenemos una mayor potencialidad en el patrón con la ayuda de la Herramienta Genexus 9.0.



En la **Figura 3.3** se muestra exactamente como se realiza el proceso. Como ya se explico partimos de una KB o Base del Conocimiento, que es donde se encuentra todas las transacciones y reglas del negocio, después se aplica el Patrón según la necesidad, al momento de aplicar el patrón este automáticamente genera nuevos objetos Genexus, se crean estilos para las páginas, procedimiento, reportes, relaciones entre tablas y otras reglas que el diseñador del patrón haya considerado.

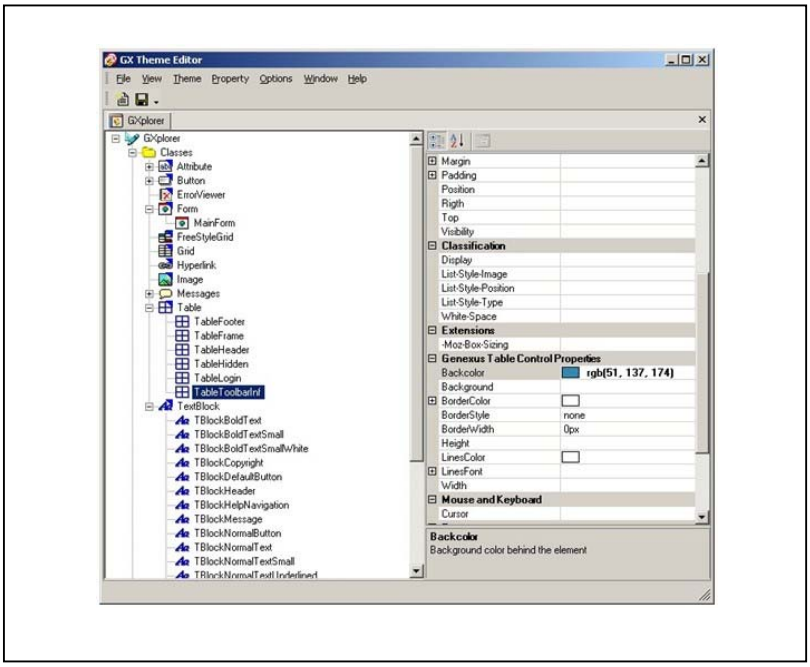
Después de haber generado el patrón y luego de que éste, haya creado e importado automáticamente objetos y demás instancias que son necesarias para su funcionamiento, la KB es modificada y forma una nueva KB que se la conoce como KB v2 o Base del Conocimiento que fue aplicada el patrón.

Luego, se retorna a Genexus 9.0 (como se muestra en la figura) y se abre la KB que se estaba trabajando, inmediatamente nos podemos dar cuenta que se han generado automáticamente nuevos objetos y además se a modificado el estilo visual de la aplicación en la que estábamos trabajando, en este momento es donde se puede cambiar parte de el diseño si se lo requiere con ayuda de la herramienta THEME EDITOR 2.0. o dependiendo de las necesidades de la aplicación se puede generar nuevos objetos o modificar los existentes para de esta manera poder obtener la funcionalidad deseada.



La manera más fácil de interactuar con el patrón para modificar el estilo visual es la herramienta THEME EDITOR 2.0, esta herramienta nos brinda una interfaz amigable y sencilla que facilita y potencia la edición del tema asociado a la aplicación Web. Los pasos necesarios para realizar la edición con GeneXus Theme Editor son:

- Abrimos el archivo donde se encuentra en el directorio de la aplicación.
- Se nos presentarán las clases incluidas en el tema con una visualización de árbol en donde podremos editar las opciones de cada una.



- Luego de realizar las modificaciones deseadas debemos guardar los cambios y automáticamente se genera el archivo GXplorer.css con la opción.
- El archivo generado debe ser ubicado en el directorio Web dentro del directorio de instalación de GXplorer.

3.4 Tipos de patrones para Genexus

Los Patterns creados por Artech a disposición de todos los usuarios GeneXus. Algunos escenarios de uso y su correspondiente Pattern:

Migración de aplicaciones a Web – Pattern “WorkWith”

Parametrización de aplicaciones en tiempo de ejecución – Pattern OAV (Object Attribute Value)

Diseño de modelo de datos con características específicas – Pattern “Bill of Materials”

3.5 Ventajas de la utilización de patrones con Genexus

Es bastante común cuando se desarrollan aplicaciones, sentir que algunas partes de la misma son similares, pero no exactamente las mismas. ¿Qué hacer en esos casos?, ¿cuál es la solución ideal para no tener que reescribir todas varias veces?. La Respuesta a estas interrogantes cuando se trabaja con Genexus es el uso de Patrones mediante la herramienta Patterns ya que la introducción de este nuevo objeto, brinda varias ventajas al momento de desarrollar aplicaciones Web con, que detallamos a continuación:

Con muy poco esfuerzo (unos pocos clicks) podemos generar programas con la funcionalidad deseada.

Se evitan errores en el código. Puede haber errores en el generador de código, pero con el uso estos errores se van corrigiendo. En cambio si se programa a mano, con cada programa nuevo podemos estar introduciendo errores.

Reducir el tiempo ya que el programador no se preocupara del diseño si no que invertirá ese tiempo en potenciar la aplicación.

La curva de aprendizaje es relativamente corta.

La herramienta Patterns fue diseñado con el objetivo de brindar un aumento de la productividad.

Otro de los objetivos alcanzados con la utilización de estos los patrones es que todo el desarrollo de la interfaz gráfica se resuelve a través de los mismos y sin necesidad de conocer código HTML ni su lógica

Crear una aplicación Web implica crear una interfaz “visualmente linda” para el usuario.

Como contra parte, no es el desarrollador del sitio quien tiene los conocimientos y el tiempo para diseñar un sitio con esas características.

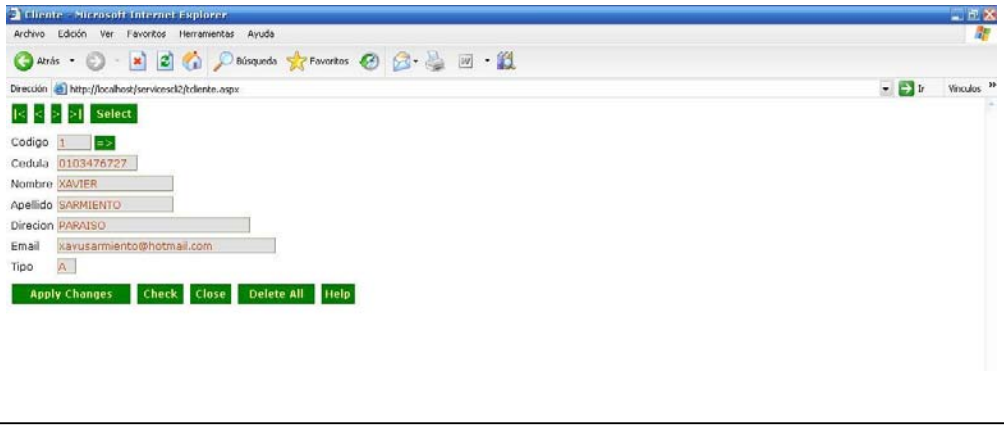
En versiones anteriores, una vez creada la estructura y lógica de un objeto Web, era necesario agregarle a posteriori el trabajo de diseño, para que el mismo fuese visualmente aceptable para integrarse al sitio Web.

Por ejemplo, el diseño default de una web transaction es como se muestra en la figura (Transacción de Clientes):

Web Transaction

Fuente: Autor Tesis

Dada la misma web transaction, con el solo hecho de asociarle un Theme, se puede lograr el siguiente cambio en diseño:



Como se pudo observar se puede generar un cambio rápido y que cause impacto a la vista del usuario sin tener que realizar mucho esfuerzo, ni invertir demasiado tiempo. Claro que si se va a querer realizar cambios en cada cosa y modificarlo totalmente el diseño estaríamos realizando tareas que se pueden obviar gracias a otros avances como los patrones de diseño.

3.5.1 Controles vs Configurar Propiedades

El trabajo de diseño no es fácil, tiene muchas exigencias, y por lo tanto es necesario invertir mucho tiempo en él; cuando ese tiempo debería ser dedicado al desarrollo de la funcionalidad de la aplicación.

Hasta ahora hacía falta una forma de hacer que el desarrollador de la aplicación se olvidara por completo del diseño gráfico de la misma.

Ahora el desarrollador puede focalizarse en la funcionalidad de la aplicación.

Un Theme agrupa en "clases" la configuración de los controles, es decir, las clases son "diseños" para todos los controles, por ejemplo, clases de botones, clases de grillas, clases de tablas, etc.

Lo único que el usuario tiene que hacer es asignar determinada clase a los controles, y los controles van a reflejar en la configuración de esa clase.

Es decir que ahora no será necesario actualizar los valores de las propiedades de los controles individualmente, y en cada form o transacción que se realice.

Entonces el usuario se desentiende completamente de lo que es el diseño propiamente dicho, del control.

3.5.2 Mayor nivel de productividad al diseñar aplicaciones Web.

Por lo antes dicho, el nivel de productividad en el desarrollo de las aplicaciones Web se ve incrementado, ya que la tarea de diseño se concentra en un único objeto: Theme.

El Theme contiene la definición de las clases, de esa forma, la tarea de diseño está concentrada en él, y no es necesario configurar las propiedades de cada control de cada objeto Web individualmente.

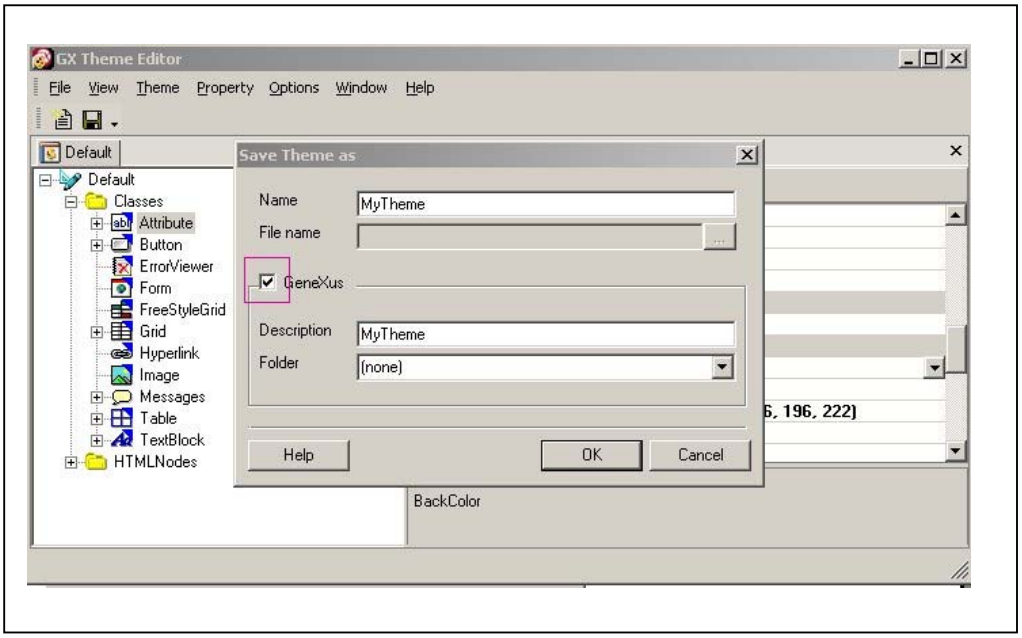
3.5.3 Se reduce notoriamente el costo de mantenimiento del sitio

Cuando se hace una aplicación Web es necesario que el sitio se vea “uniforme”.

Esto implícitamente requiere de un alto costo de mantenimiento, ya que si por ejemplo hay que cambiar el color de una grilla de azul a celeste, probablemente habría que hacerlo en todas las páginas del sitio para mantener la uniformidad del mismo.

Antes, había que ir por cada uno de los controles grid de cada objeto. Hoy el cambio se realiza en un solo lado: el Theme.

Todos los objetos basados en él van a reflejar el cambio automáticamente, y no solo eso; Sino que si se realiza un cambio en el Theme, basta con guardarlo usando el editor (así se obtiene un archivo con extensión .css), y llevar ese archivo a producción. Nada mas, no es necesario generar ni compilar nada.



Esto permite, por ejemplo que una Software pueda hacer personalizaciones por cliente sin necesidad de tocar el código GeneXus.

3.5.4 Mayor funcionalidad al uso de styles Genexus en Web

Mediante la nueva funcionalidad “Themes” introducida, será posible complementar el manejo de styles GeneXus en ambiente web.

Los cambios en las propiedades de los controles se realizarán a través de las clases del Theme, no a través del style propiamente dicho. Como consecuencia de esto, los cambios en el style (indirectamente a través del Theme) se van a reflejar en el objeto basado en él, asegurando el dinamismo esperado.

3.6 Conclusiones

Como vimos en este capítulo nos podemos dar cuenta que la herramienta Genexus 9 potencializa a los patrones que se puedan realizar, ya que este brinda un gran soporte y una facilidad para poder aplicarlo en cualquier sistema o KB que se desee.

Tenemos grandes herramientas que nos ayuda a modificar la apariencia del sistema sin tener que invertir mucho tiempo en los cambios, y además desde el propio Genexus se pueden realizar mejoras al patrón creando filtros, validaciones y cálculos que son.

Otras de la razones de realizar patrones para Genexus, es que esta Herramienta se está difundiendo muy rápidamente en nuestro medio por lo cual se ve mucho futuro a los patrones ya que esta además que nos ayuda a realizar tareas muy rápidamente, ya que es mucho más amigable y no requiere programadores expertos.

Al no tener que preocuparnos en los que es diseño, obtenemos un software de calidad ya que al no tener que escribir código evitamos errores dentro del sistema.

Referencias

http://www.gxtechnical.com/gxdlsp/pub/genexus/devenv/docum/releasenotes/8.0/..%5C..%5C..%5C..%5Cinternet%5Cdocum%5Cmanuals%5C8.0%5Cthemes.htm

http://www.genexus.com/

http://wiki.gxtechnical.com/commwiki/servlet/hwiki?UseThemeeditor

http://www.acpsistemas.com.ar/gxpsites/hgxpp001.aspx?1,6,182,O,S,0,,

http://www.genexus.com/portal/hgxpp001.aspx?2,32,666,O,S,0,MNU;E;87;2;MNU;

CAPITULO 4: DESARROLLO DE LA APLICACION

4.1 Introducción

En este capítulo se centrara la atención principalmente en a la elaboración de los patrones y cómo interactúan con los diferentes objetos de Genexus para trasformar una KB y dar paso a un sistema completamente trasformado en su diseño, con calidad y el funcionamiento deseado.

En el trascurso del capítulo se intentara dar todos los pasos necesarios que se debe seguir para la elaboración de un patrón para la herramienta case Genexus y como se debe manipular para su funcionamiento.

4.2 Análisis

4.2.1 Fundamento para el desarrollo de los Patrones

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Los problemas fundamentales es el vacío existente que el programador tiene que tomar decisiones de diseño. Para solventar estos problemas se produce un nuevo enfoque: el diseño con patrones. La idea nace del arquitecto Christopher Alexander, que aplica el concepto a la construcción urbanística.

Lo primero para la construcción de los patrones es agrupar una colección de soluciones de diseño que son válidas en distintos contextos y que han sido aplicadas con éxito en otras ocasiones.

Un patrón de diseño es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias).

Los patrones son soluciones de sentido común que deberían formar parte del

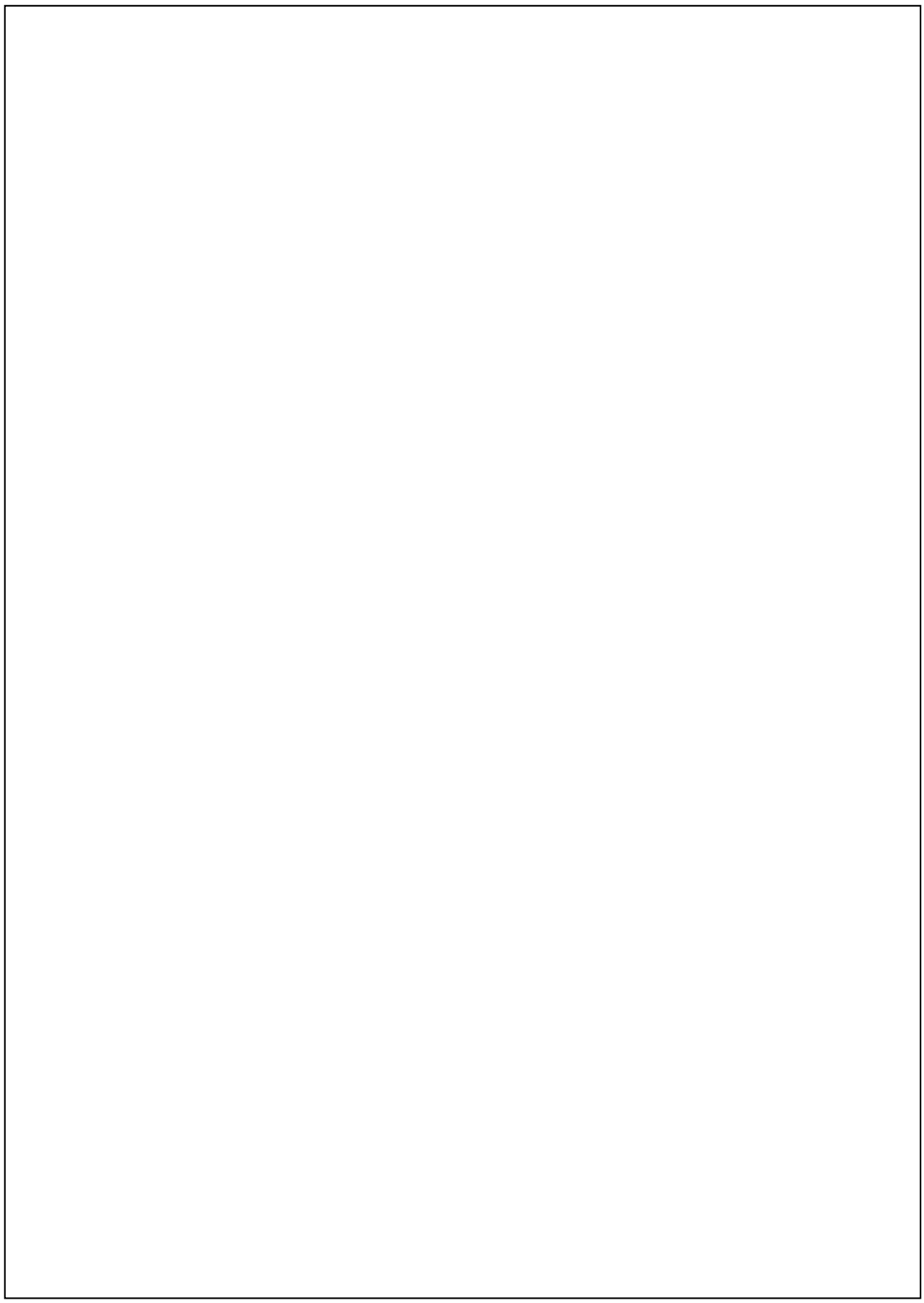
conocimiento de un diseñador experto. Además facilitan la comunicación entre diseñadores, pues establecen un marco de referencia (terminología, justificación).
Además, los patrones de diseño, también nos ayudarán a especificar las interfaces, identificando los elementos claves en las interfaces y las relaciones existentes. De igual modo nos facilitará la especificación de la implementación.
También, y de forma casi automática, nos ayudan a reutilizar código, facilitando la decisión entre "herencia o composición", agilitando de está manera la realización de un sistema informático.

4.2.2 Funcionamiento detallado de Patrones

Antes de iniciar daremos a conocer unas definiciones que nos ayudaran a entender mejor el funcionamiento de los Patrones, se darán términos que se van a utilizar a lo largo del capítulo.

- **Patrón.-** El Patrón determina un comportamiento común de un conjunto de objetos.
- **Instancia.-** La Instancia es donde se determinan los elementos diferenciales de un objeto en particular. Es el instrumento sobre el que actúan los programadores de GeneXus para resolver el comportamiento de los Web Pannels que se van a generar.
- **Generador.-** El generador de un Patrón es el recurso que tomando la instancia definida genera el Objeto GeneXus (Web Pannel).

En la siguiente figura se muestra paso a paso el funcionamiento de los patrones.



Para aplicar el patrón partimos de una KB o Base del Conocimiento, que es donde se encuentra todas las transacciones y reglas del negocio del sistema en el cual se va a trabajar durante este proceso, después utilizamos la herramienta Patterns 1.1 de Artech y elegimos el Patrón elaborado según la necesidad de las funciones que tiene

el programador, al momento de aplicar el patrón este automáticamente genera nuevos objetos Genexus, se crean estilos para las páginas, procedimiento, reportes, relaciones entre tablas y otras reglas que el diseñador del patrón haya considerado.

Después de haber generado el patrón y luego de que éste, haya creado e importado automáticamente objetos y demás instancias que son necesarias para su funcionamiento, la KB es modificada y forma una nueva KB que se la conoce como KB v2 o Base del Conocimiento que fue aplicada el patrón.

En la KB v2 a través de Genexus, el programador puede realizar los cambios que considere necesarios para potenciar la aplicación, luego puede elegir el lenguaje en el que desea generar y el gestor de base de datos que se utilizara. Dependiendo del lenguaje que se utilice Genexus generara las líneas de código necesarias para que el programa funcione. El siguiente paso es compilar y ejecutar la aplicación y disfrutar de las bondades que nos brinda el patrón.

4.2.2.1 Creación de la instancia por defecto

Instancia

Instancia es un archivo xml que define el caso de un patrón, es decir, tiene toda la información necesaria para poder crear a partir de ella y con la aplicación de ciertos programas (templates) un archivo xpz para poder ser consolidado en GX y crear objetos GX.

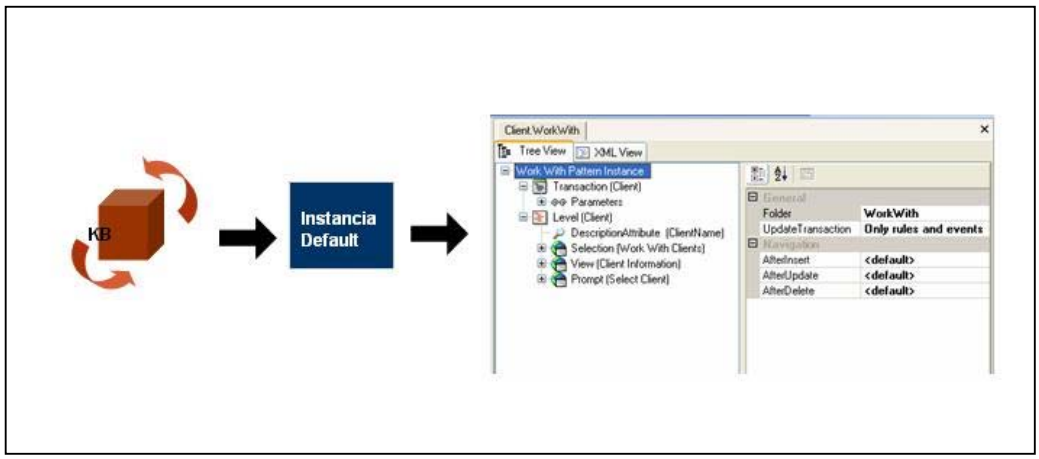
Aquí se explica cómo se genera una instancia por defecto a partir de una TRN (Transacción) y qué archivos y configuraciones se toman en cuenta en el momento de la creación de la instancia.

Partimos de una TRN de una KB de Genexus.

Para obtener toda la información de la TRN y generar la instancia por default se usa la workwith.dll y dentro de ella la GXKnowledgeBase.dll para recorrer las tablas subordinadas, para saber como generar los Tabs de las tablas asociadas.

Junto con estas clases y con la definición de los patrones (pattern.definition) es que se crea la instancia por defecto.

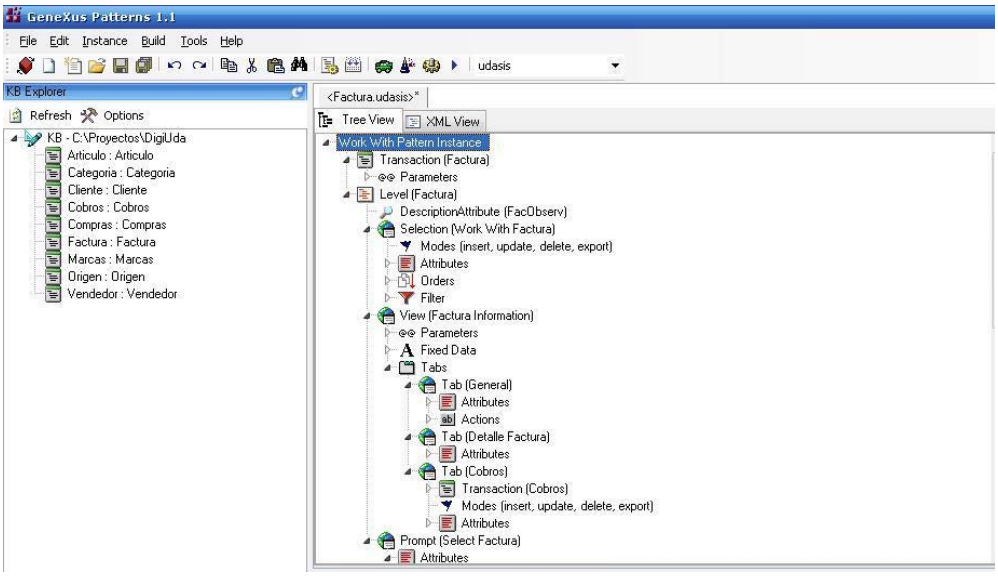
En el pattern.definition indica los nodos y qué valores se va a generar la instancia por defecto.



Para indicarle a los patrones cual es la clase que se va a usar para crear la instancia por default, se configura en el pattern.definition en (opción Tools / Edit pattern definition) la propiedad DefaultGenerator que por defecto tiene el valor de GXPatterns.WorkWith.WorkWithGenerator.

Como dijimos, una instancia es un archivo xml, y para escribir ese archivo, se hace como un xml cualquiera.

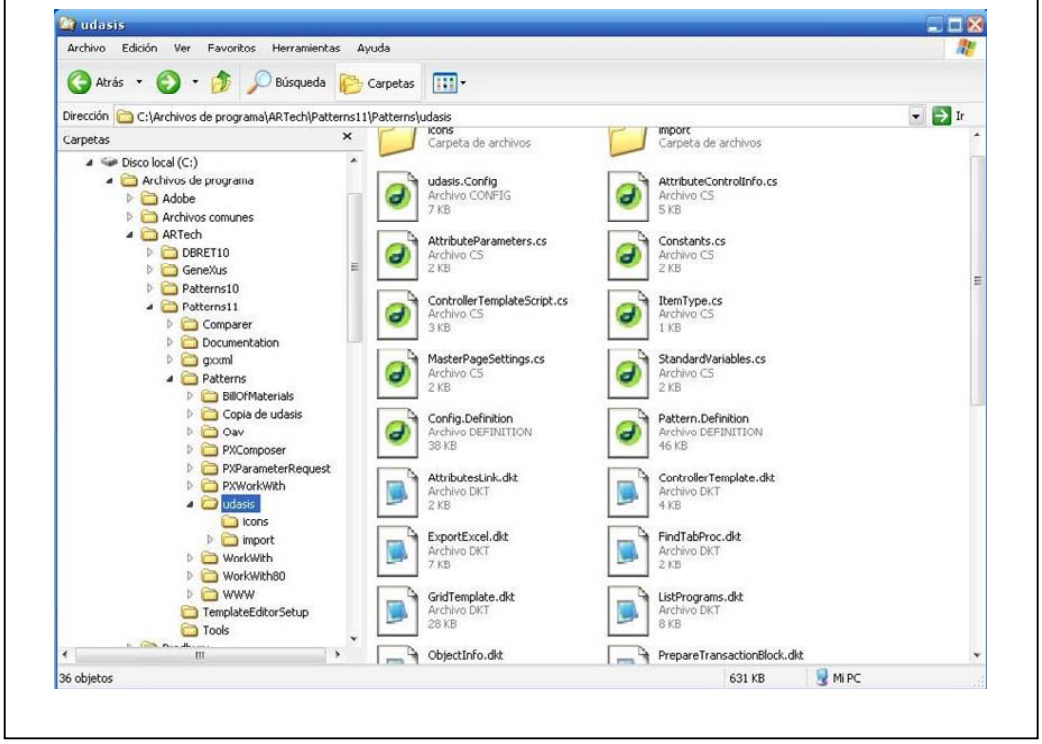
De esa forma se va generando el xml correspondiente con cada uno de sus nodos, que luego se puede visualizar más fácilmente en el Tree View.



4.2.2.2 Archivos de definición

Por otro lado tenemos la definición de los patrones. Estas definiciones están en el archivo **pattern.definition**, que indica por ejemplo, que nodos, atributos y propiedades aparecerán en la instancia por defecto al aplicar un patrón.

Se definen todos los atributos que se van a utilizar dentro de los patrones, con los valores que pueden tomar, tipo de atributo, categoría y todo que tenga que ver con los atributos en general.



4.2.3 En busca de la funcionalidad deseada

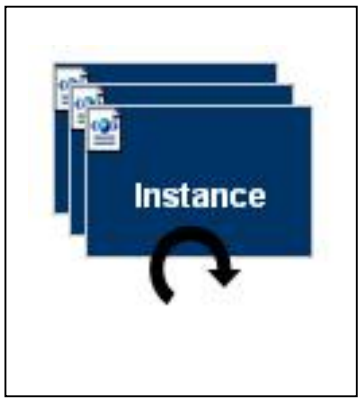
Las instancias generadas por los patrones se pueden modificar, cada cambio que se realice en una instancia, generará una nueva y al guardarla se tendrá la última versión de la instancia con sus cambios.

Una vez que se decida que la instancia obtenida será la definitiva, a partir de ella, se generarán los objetos GX necesarios.

El nivel de estos cambios puede variar, dependiendo si se quiere que el cambio afecte a una determinada instancia o a todas ellas.

En el caso que se necesite modificar una instancia en particular, se hará sobre la definición de esa instancia, hasta lograr la funcionalidad o comportamiento que se requiera.

En ese caso se van a tener diferentes instancias, aunque cada vez que se guarde una instancia, se tendrá esa como última versión.



Cuando se necesiten realizar cambios comunes a un conjunto de instancias, en ese caso se debe realizar el cambio en el archivo de configuración **udasis.config**, por ejemplo.

Luego del cambio, cada instancia que se genere tomará ese cambio. El archivo **udasis.config** tiene la tarea de generar todo los cambios producidos por el patrón, importaciones de imágenes, estilos y objetos para la creación del patrón.

Un ejemplo de lo que genera el archivo **udasis.config** está a continuación:

```
<Configuration Specification="udasis" Version="1.1.0">
 <Config>
  <ImagePath>images/</ImagePath>
  <Template>
   <Folder>udasis</Folder>
   <InsertModeAt>begin</InsertModeAt>
   <AutoLinkAttributes>true</AutoLinkAttributes>
   <UpdateTransaction>Only rules and events</UpdateTransaction>
   <SelectionIsMain>false</SelectionIsMain>
   <ViewDescription>&lt;Object&gt; Information</ViewDescription>
   <TabsForParallelTransactions>false</TabsForParallelTransactions>
   <AfterInsert>Go to View</AfterInsert>
   <AfterUpdate>Go to View</AfterUpdate>
   <AfterDelete>Go to Selection</AfterDelete>
  </Template>
…….
```

En la tabla anterior se muestra un extracto de lo que encontramos en el archivo udasis.config, mostrando muy claramente como se importa las carpetas, las imágenes y los objetos.

4.2.4 Generación del xpz

Como dijimos, la generación del .xpz se va haciendo desde los .dkt.

Los dkts combinan código C# y código GX (KMW), al principio del .dkt aparece:

```
<%@ Template Language="C#" TargetLanguage="GX" Description="Work With Main Template"
%>
```

Lo que se hace con el C# es ir recorriendo y procesando el xml de la instancia y a partir de allí se va escribiendo un nuevo .xml de salida (el xpz).

El xpz es también un xml con un formato exclusivo para Genexus:

```
ExportFile>
  <Model>
>>> Información sobre las props. del modelo
    <Id>1</Id>
    <Name>NewKB</Name>
    <AttLen>30</AttLen>
    <TblLen>30</TblLen>
    <ObjLen>30</ObjLen>
    </Model>
  <KMW>
>>> Información sobre la KB
    <MajorVersion>2</MajorVersion>
    <MinorVersion>2</MinorVersion>
    <Path>D:\AJAXSample</Path>
    <MaxGxBuildSaved>2601</MaxGxBuildSaved>
  </KMW>
  <GXObject>
>>> Aquí empieza la Información de los objetos que están dentro del xpz
<Transaction>                                          (en este caso es solo una TRN)
      <Info>
        <Name>Client</Name>
        <Description>Client</Description>
      </Info>
       .....................
       .....................
      <Structure>
>>> Información de la estructura
        <Source>
        .............
        </Source>
        <LevelInfo>
>>> Información de los atributos
          <KeyId>1</KeyId>
          <Name>Client</Name>
          <Description>Client</Description>
```

```
      </LevelInfo>
      <AllowNulls>
        <Name>ClientName</Name>
        <Value>No</Value>
      </AllowNulls>
     .....................
    </Structure>
    <Variable>
>>> Información de las variables
      <Name>Today</Name>
      <Title>Today</Title>
      <Type>Date</Type>
      <Length>8</Length>
      <Decimals>0</Decimals>
      <Picture>99/99/99</Picture>
      <Property>
        <Type>string</Type>
        <Name>ATT_PICTURE</Name>
        <Value>99/99/99</Value>
      </Property>
    </Variable>
.....................
<Form>WinForm</FormInfo>
>>> Información de los forms
<HTMLForm>WebForm</HTMLForm>
<Events>
...................
</Events>
>>> Información de los eventos
.................
  </Transaction>
</GXObject>
<GXObject>
  <Attributes>
   .................
  </Attributes>
    .............
 </GXObject>
</ExportFile>
```

Entonces lo que se hace es recorrer la instancia con los templates e ir escribiendo el xpz como se describió antes.

Así encontramos en los templates el siguiente código:

```
<GXObject>
<Transaction>

>>> Esto se escribe en el xpz
<%
foreach (XmlNode subNode in trnNode.SelectNodes("*"))

>>> Se comienza a recorrer la instancia
{
switch (subNode.Name)
{
case "LastUpdate" :
// Strip LastUpdate info.
break;
case "HTMLForm" :
if (updateWebForm)
{
%>
<%@ CallSubTemplate WWTrnWebForm Transaction='transaction'
TransactionNode='TransactionNode' HTMLFormNode='subNode'
PgmDescVarId='GetVarId(trnNode, "PgmDesc")' GxRemoveVarId='GetVarId(trnNode,
"GxRemove")' Update='Update' %>
<% }
else
{
%>
<%= subNode.OuterXml %>

>>> Se escribe toda la información del nodo en el xml
??????????????
case "Events" :
<Events>
??????????????
```

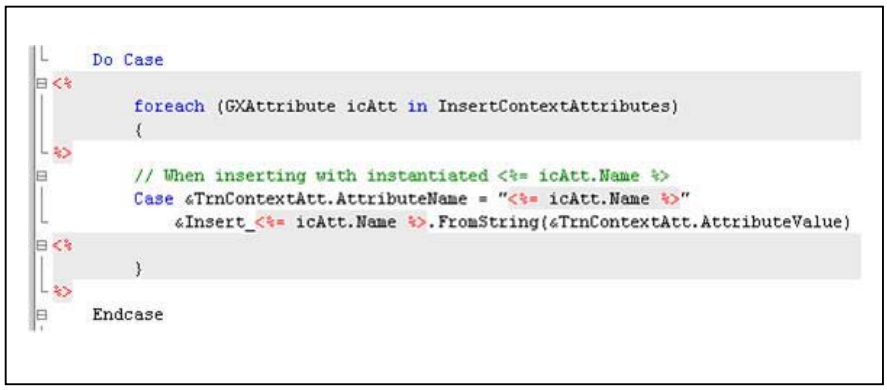
```
</Events>
??????????????
</Transaction>
</GXObject>
```

4.2.5 Templates

Los patrones es una herramienta que está basada en templates (archivos dkt). Esto es que en base a las instancias generadas, a la aplicación de ciertos archivos de configuración y a ciertos templates, se generan los archivos xpz que se podrán consolidar en GX para generar más objetos GX.

Los templates mezclan código C# con el lenguaje KMW que se utiliza en GX, de esa forma el texto de salida que se produce (xml correspondiente al xpz) tiene código copiado directamente desde el template y código xml generado por las sentencias C#.

Si se visualiza el código de un template se puede ver algo como lo siguiente:



En la **Figura 4.2.5** se puede ver que se está generando un Case para colocar el nombre de una transacción para luego ser colocada en el xpz; y se pueden distinguir dos componentes:

El código que está entre <% y %> es código C#, por ejemplo <%=icAtt.Name%>, en ese caso se sustituye ese trozo por el valor de la variable en ese momento o el resultado de procesar con el foreach?

Lo que está fuera de los signos <% y %>, es código GX, se copia directamente al xpz.

Hay algunos templates importantes y mencionamos especialmente (aunque los nombres son bastante descriptivos y es fácil saber que genera cada uno):

GridTemplate.dkt

Generar toda la información de los work with, grids y prompts.

ViewTemplate.dkt

Generar toda la información de los views.

TabularTemplate.dkt

Generar toda la información referente al componente General que se usa en los Views.

WWTransaction.dkt

Generar toda la información de las Transaciones.

Dentro de los Templates, también se utilizan algunas dlls (assemblies), que ayudan a la generación del xml.

En el **ViewGenerator.dkt** se usan:

```
<%@ Assembly Name="GXKnowledgeBase" %>
<%@ Assembly Name="WorkWith" %>
<%@ Assembly Name="TemplateHelpers" %>
.....
```

La **GXKnowledgeBase.dll** y la **TemplateHelpers.dll** ayudan a recorrer la KB y a la generación del xml, en particular estas dlls ayudan en la generación de variables, generación de controles, relaciones entre tablas etc.

Estas dlls son genéricas para cualquier patrón.

También dentro del template se utiliza la **WorkWith.dll** ya que dentro de esta dll se encuentran varias rutinas para generar partes de los objetos y llamadas a las Clases cs. Los **cs** son clases que contienen código C# que interactúan con la base de

conocimiento para generar nuevos objetos Genexus con sus propia funcionalidades y apariencia.

Por ejemplo el **PagingButton.cs** es la fuente que genera el código en el xpz para generar los botones de paginación.

En el **GridTemplate.dkt** se puede ver que se hace referencia a esta rutina:

```
private string PagingButtons(ArrayList buttons)
{
 string r = "";
 foreach (PagingButton button in buttons)
  r += button.ToHtmlImage();
 return r;
}
```

Y en el **PagingButton.cs** se tiene:

```
public string ToHtmlImage()
  {
   return HtmlToKmw.Image(Name,
GlobalWorkWithConfig.Config.Theme("Image"), m_Caption,
GlobalWorkWithConfig.Config.ImagePath() + Image) + "\r\n";
  }
```

De esta forma se devuelve el código para generar el trozo de xpz de los botones.

Otra rutina interesante que se encuentra dentro de la workwith.dll es la **IsLastInstance**.
Al dar F5 también se controla si es necesario o no generar en el xpz los objetos que se consolidarán una sola vez (list programs, recent links, etc.).

Esta rutina es la que controla si se tienen que generar estos objetos o no, como estos objetos van una sola vez se generan en la última instancia.

4.2.5.1 ViewGenerator.dkt

A continuación se comentan algunos puntos importantes de este template, que servirán de guía para entender como está implementado y como funciona.

A su vez, puede servir para el resto de los templates ya que todos siguen la misma estructura.

Aquí se indican los parámetros que recibe el template.

FileName - Nombre de la instancia

```
<%@ Property Name="FileName" Type="System.String"%>
```

Toma la información del folder donde se generarán los objetos

```
string folder = General.AttributeValue(instanceNode, "folder");
```

Se recorren las instancias y se lee el nodo Level. El template procesa el xml de la instancia por nodos.

```
foreach(XmlNode objectNode in instanceNode.SelectNodes("level"))
```

Se va obteniendo información de la KB (nombres de objeto, descripciones, etc.) usando la templatehelpers.dll (General.NodeName(objectNode) accede a la templatehelpers.dll).

```
string objName = General.NodeName(objectNode);
string objDescription = General.NodeDescription(objectNode);
string descriptionAttribute =
General.NodeName(objectNode.SelectSingleNode("descriptionAttribute"));
```

Llama al subtemplate ViewTemplate para generar los Tabs del View con todos los datos pasados por parámetro.

```
<%@ CallSubTemplate ViewTemplate ViewNode='viewNode'
ViewObjName='viewName' Folder='folder' SelectionLink='selectionLink'
SelectionDescription='selectionDescription' ObjName='objName'
Parm='parameterList' DescriptionAttribute='descriptionAttribute' %>
```

Para cada nodo en Tabs\Tab se llama al TabularTemplate y al GridTemplate dependiendo si se generará un componente con tabular o con grid (propiedad Type del nodo Tab).

```
foreach (XmlNode node in viewNode.SelectNodes("tabs/tab"))
```

4.2.5.2 WWTransaction.dkt

Este template genera un xpz de la TRN original y se queda con el nodo GXObject\Transaction del xml (usando el ObjNode).

```
foreach (XmlNode objNode in trnExportFile.SelectNodes("/ExportFile/GXObject
if ((objNode.ChildNodes0.Name == "Transaction") &&
(objNode.SelectSingleNode("Transaction/Info/StyleClass") == null ||
((XmlElement)objNode.SelectSingleNode("Transaction/Info/StyleClass")).InnerText
== ""))
 {
  // This is the transaction node; all others (tables, styles, subtype groups, etc.) are
ignored.
  trnNode = objNode.ChildNodes0;
  break;
 }
}
```

Esto se hace porque es la parte de la TRN que se va a modificar, ya que se preservan todas las reglas y eventos de la Transacción.

Por lo tanto el único nodo del xml de la TRN que se modifica es Transaction, el resto se copia al xpz final igual que el xpz original.

```
bool updateWebForm = (Update != "only rules and events");
```

Luego se modifica o no el web form de la TRN dependiendo de la propiedad update transaction.
Dentro de este dkt, también está el insertContextAttributes que se usa para configurar la PK cuando se inserta un registro y debe instanciar la PK (cuando se da de alta una factura desde el WW de Clientes, el código de cliente va instanciado).

4.2.6 Template Engine

Es un programa que procesa un texto de entrada (instancia, formato xml) y produce otro texto, en el caso de los patrones es un xpz (archivo xml).

De esta forma, a partir de la KB, de la generación de las instancias y de los archivos de configuración, aplicando los templates, se obtienen las definiciones de los objetos GX que se desean consolidar en la nueva versión de la KB.

Como dijimos hasta ahora, para generar una instancia asociada a un objeto, basta con dar doble click sobre el objeto y en ese momento se utiliza el programa configurado en la propiedad DefaultGenerator para generar la instancia por defecto.

Una vez que las instancias están listas para ser procesadas, al dar F5 desde la Aplicación Patterns (opción Build\Apply and Consolidate) es cuando se comienza el proceso de instanciación, en este proceso se ejecutan los siguientes pasos.

Por cada instancia que se tenga:

- Se genera un xml (que representa el xpz correspondiente a esa instancia).
- El xpz mencionado contiene todos los objetos que se consolidarán en GX asociados a la instancia mencionada.
- Se genera un único xpz con todos los xml de las instancias relacionadas.
- Se consolida el output.xml.xpz en la KB obteniéndose la versión 2 de la KB.

El primer paso (generar los xml de cada una de las instancias) lo hace el template engine, el segundo paso (generar un solo xpz) lo hace directamente la herramienta.

El proceso de template engine es el que se encarga de tomar una instancia y aplicarle un template para generar a partir de ella un xml (xpz) con todos los objetos.

Además de los dkts para realizar este proceso se utilizan algunas dlls auxiliares.

4.3 Diseño

4.3.1 Etapas del Patrón

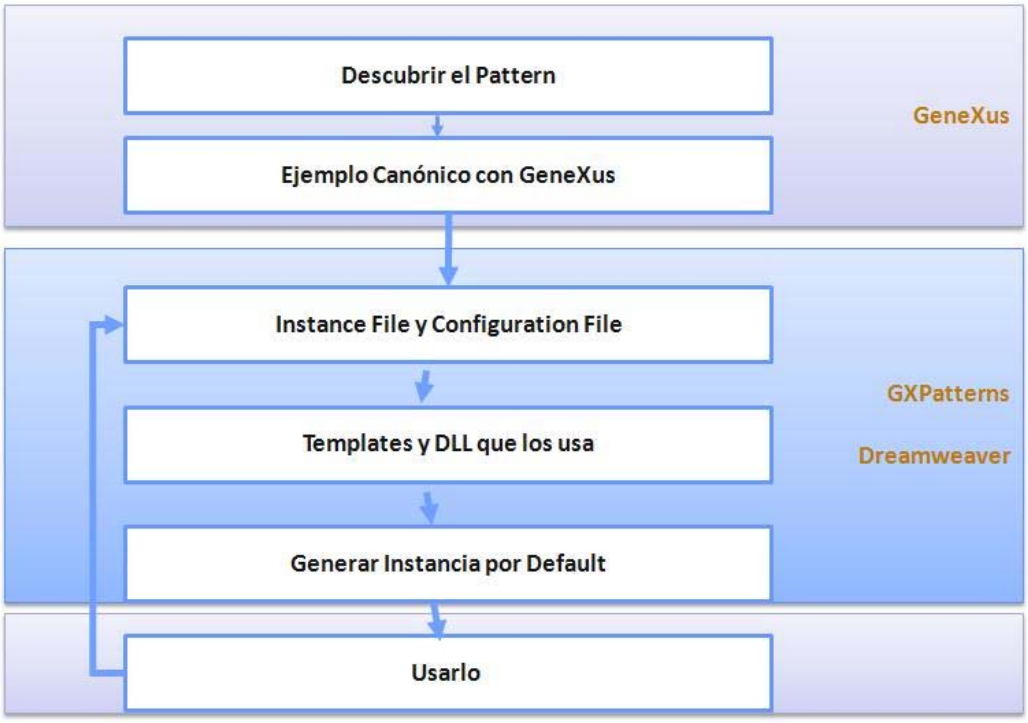
El primer paso para el diseño de un patrón es describir un problema que se presenta a los programadores, en este trabajo, el problema se centrara en la reducción de tiempo en la estandarización de pantallas y el diseño de todas las interfaces en web; Además, se pretende facilitar la navegación y visualización de las tablas relacionadas que son ocupadas en el sistema y brindar un reporte que se pueda manipular fácilmente; esto se lo logra exportando un reporte para Excel.

El segundo paso es diseñar lo que queremos que sea el patrón en Genexus, esto es importante para tener claro cómo lo deseamos que quede nuestro patrón visualmente y posteriormente seguir los estándares impuestos para realizar el patrón.

Una vez que tenemos claro el funcionamiento que deseamos alcanzar el siguiente paso es generar los archivos de configuración, los templates y los estilos que se aplicaran a las transacciones existentes y que se aplicaran a los nuevos objetos que se crearan obteniendo de esta manera un estándar a lo largo de toda la aplicación.

Es el momento de utilizar el patrón y realizar las pruebas necesarias que realimentaran la información necesaria para de esta manera realizar los cambios que se consideren necesarios hasta lograr la funcionalidad deseada.

Figura 4.3.1 Ciclo para la realización de un Patrón

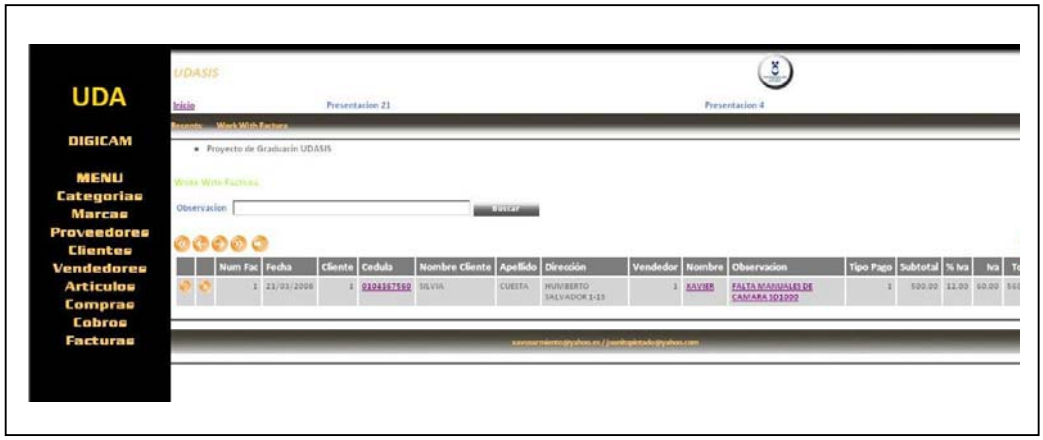


Fuente: Autor Tesis

4.4 Implementación

4.4.1 Ejemplo Canónico

El punto de partida del patrón será el ejemplo canónico que consiste en realizar la programación en Genexus sin omitir detalles y luego distribuir para obtener el XML y de esta manera esto nos facilita para ir armando los archivos necesarios para el diseño del patrón.



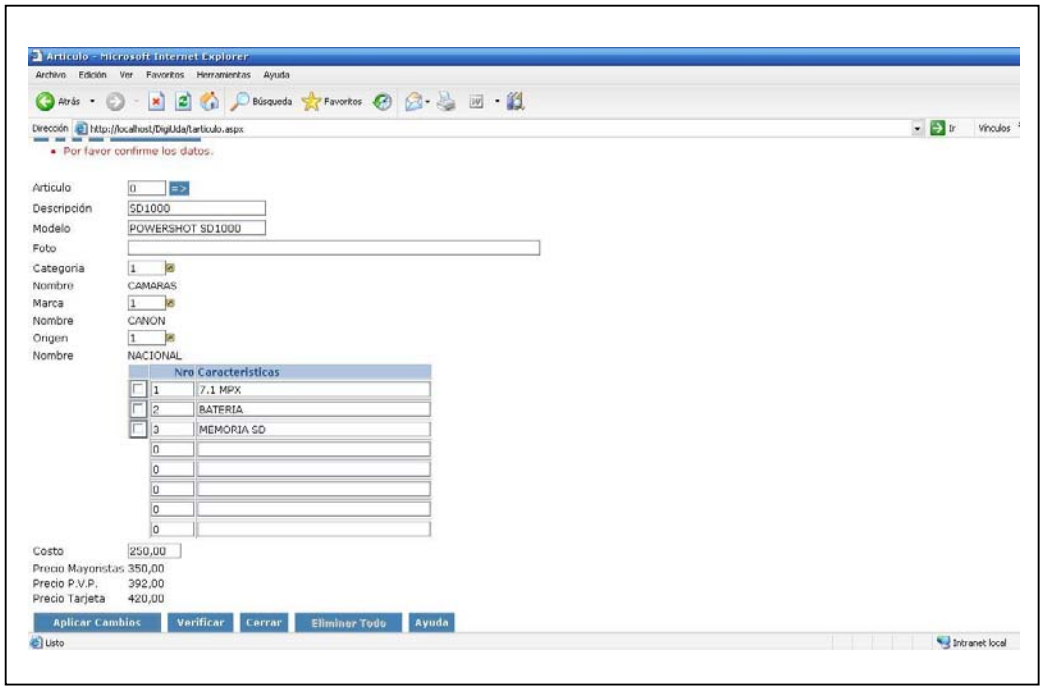
4.4.2 Estandarización Diseño Grafico

Una vez establecidos los estándares de Diseño Gráfico del Patrón que se quiere elaborar, van a generar todas las pantallas siguiendo ese mismo patrón, para que después el programador no se preocupe de la implementación gráfica, ni de las funcionalidades que va incorporando al Web Pannel, pues dicha implementación va a ser resuelta por el patrón automáticamente.

A continuación mostramos unos ejemplos del diseño grafico:

Transacción

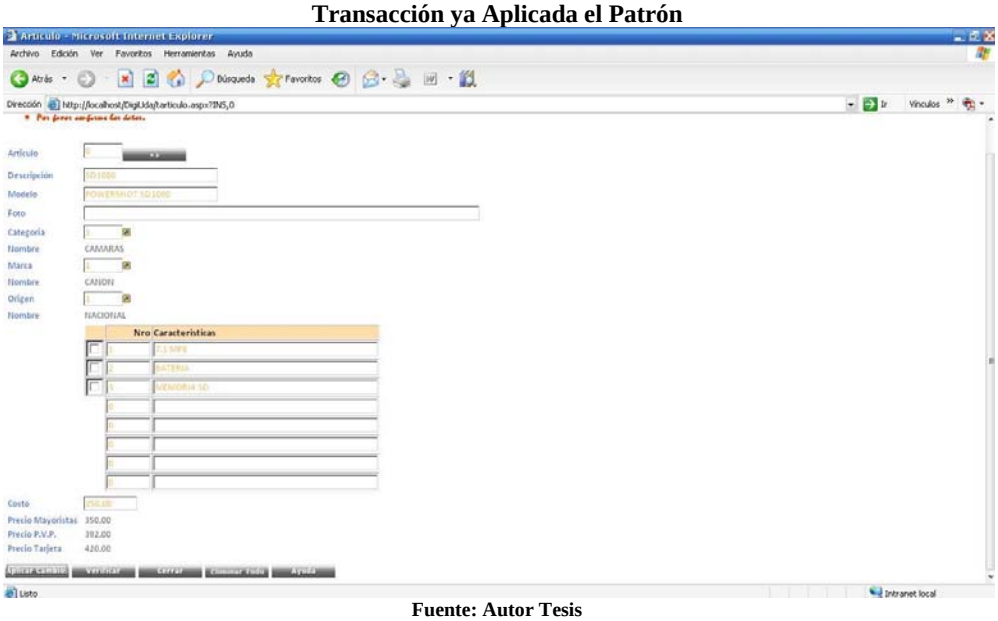
Transacción normal sin aplicar el patrón.



Transacción ya aplicada el patrón.

Para el patrón se propuso que todas las pantallas sean llamativas y con un estilo moderno, para eso decidió que todos los labels que identifican un atributo, vayan de color celeste con letra calibri de tamaño 9 pts, de manera que resalte la diferencia con los valores que se van dando a cada atributo que son de color tomate con la misma fuente y tamaño. Los botones se decidió colocar en la parte inferior con un degrade de color para darle un aspecto visual más llamativo.

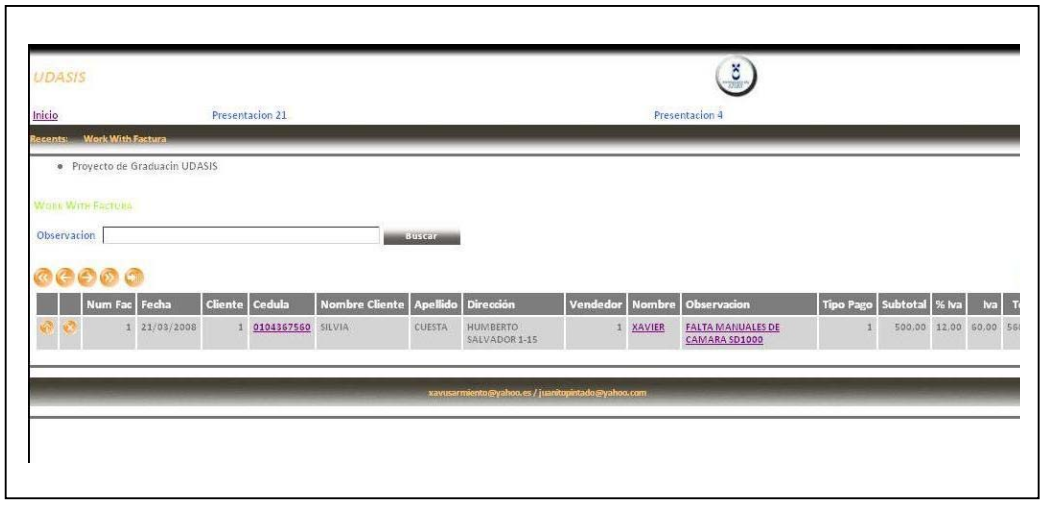
Transacción ya Aplicada el Patrón



Fuente: Autor Tesis

Web Panel

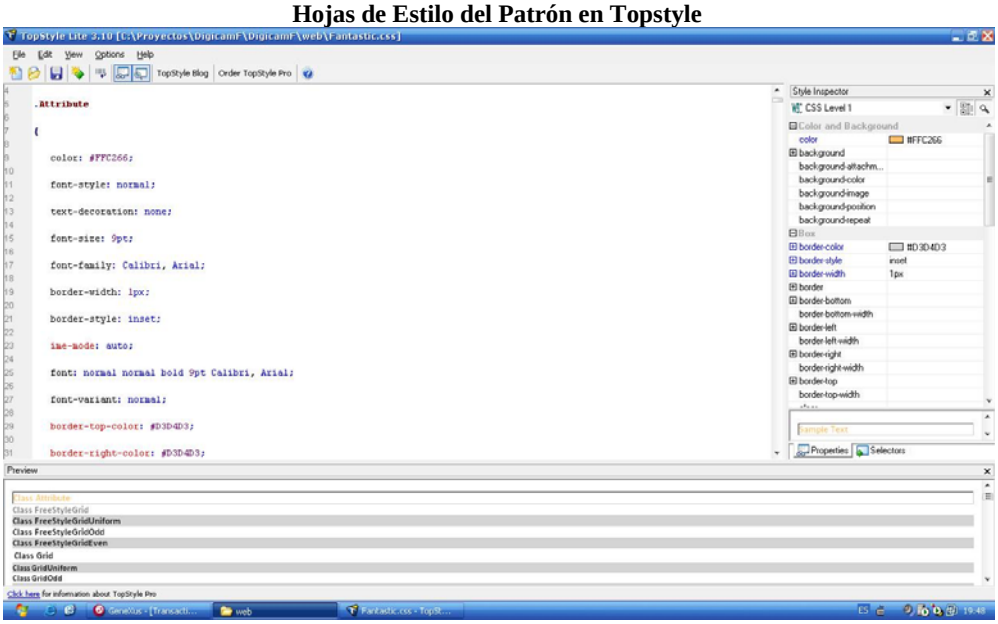
No existe web panel si no se realiza manualmente, mientras una vez que se aplica el patrón tenemos web panel de consultas muy interactivos y con un estilo muy innovador.



Para alcanzar este diseño utilizamos hojas de estilo y para que Genexus pueda interpretar tenemos que convertir la hoja de estilo en lenguaje xml, de esta manera cuando se realice la importación desde la herramienta Patters 1.1 a genexus se creara la hoja de estilo la misma que se aplicara a todos los objetos generados.

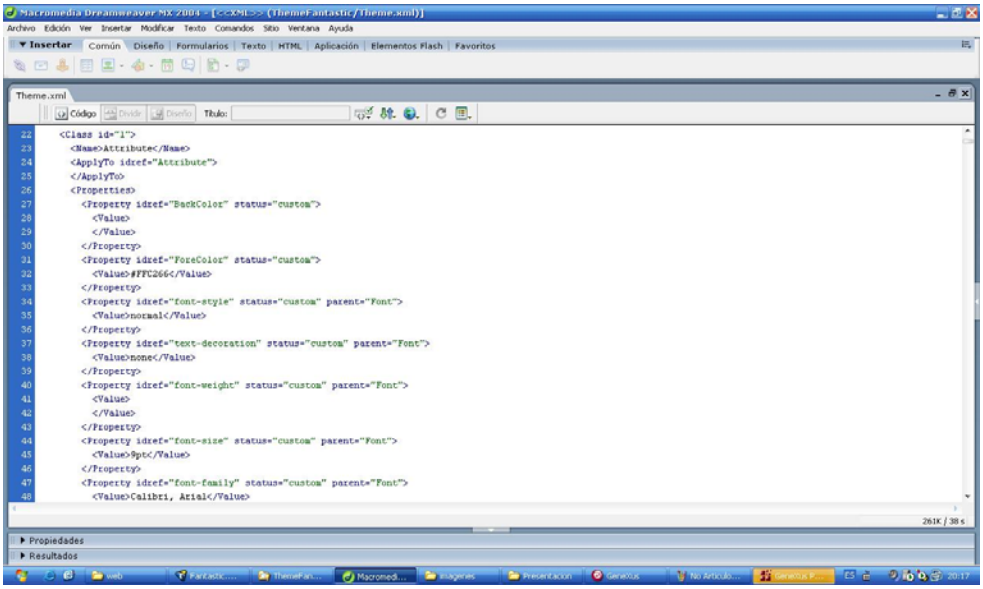
Las visualizaciones de las Hojas de Estilo, se ven así en TopStyle que es una herramienta que ayuda a visualizar las hojas de estilo rápidamente que se pretende implementar.

Hojas de Estilo del Patrón en Topstyle



Fuente: Autor Tesis

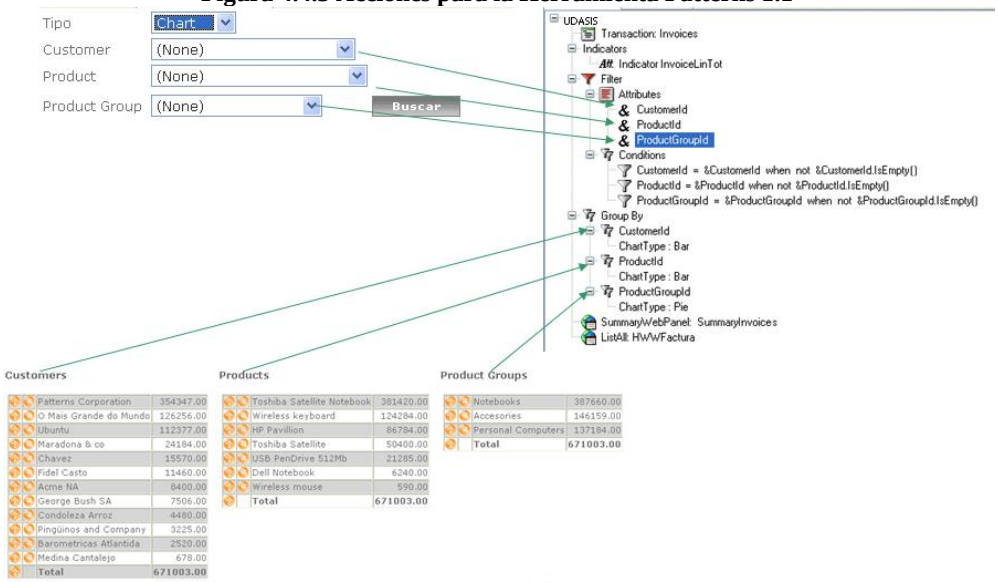
Pero para que el patrón funcione, todo este código debe ser transcrito en el formato xml, para luego ser procesado por Genexus y al final poder aplicarlo en cualquier transacción que uno desee.



4.4.3 Auto-Programación del Patrón.

Acciones en la Herramienta Patterns

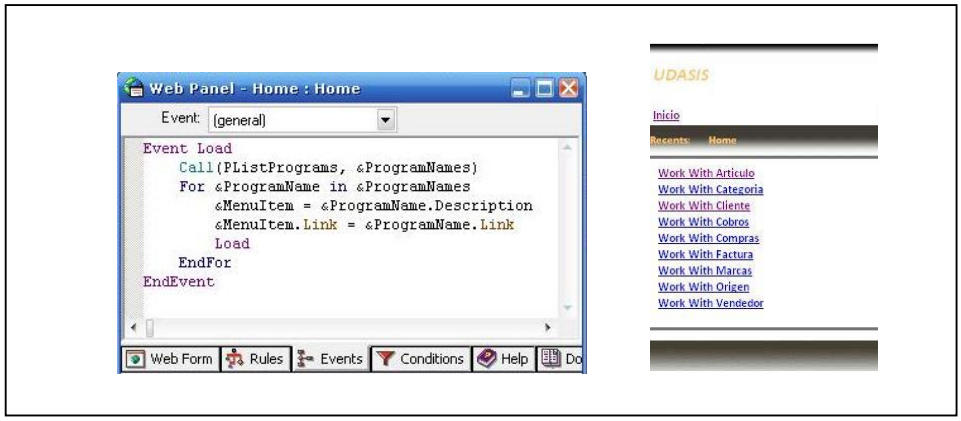
Figura 4.4.3 Acciones para la Herramienta Patterns 1.1



Fuente: Autor Tesis

En la FIGURA 4.4.3 se muestra que con la herramienta Patterns 1.1 se pueden agregar filtros, campos de ordenamientos, variables, etc. Los cuales automáticamente se presentan en la aplicación ya generados con el estilo propio del patrón.

Vínculos



El patrón genera hipervínculos automáticamente dependiendo la cantidad de objetos que se creen en la KB, y estos nos ayudaran para la realización del mantenimiento de las tablas. Además genera otros hipervínculos de los últimos páginas visitadas de la aplicación y de cualquier otro links que el programador haya decidido inicialmente, esto nos facilita la navegación con dentro del sistema.

Filtros

Fuente: Autor Tesis

El patrón crea filtros que permitirán la búsqueda más rápida dentro de la grilla estos filtros son creados en todas las grillas de los mantenimientos logrando así una consulta más rápida y eficiente, además de los filtros automáticos que nos crea por defecto nos da la posibilidad de crear filtros dependiendo de las necesidades del programador.

Ordenamientos

Otra de las ventajas que nos brinda el utilizar el patrón udasis es la generación de campos de ordenamiento, esto brinda una gran facilidad de navegación al usuario cuando intenta buscar un registro con determinadas características.

El patrón tiene la opción de incluir los campos de búsqueda que el programador considere necesario, esto campos de búsqueda toma automáticamente el estilo potenciando el patrón estándar.

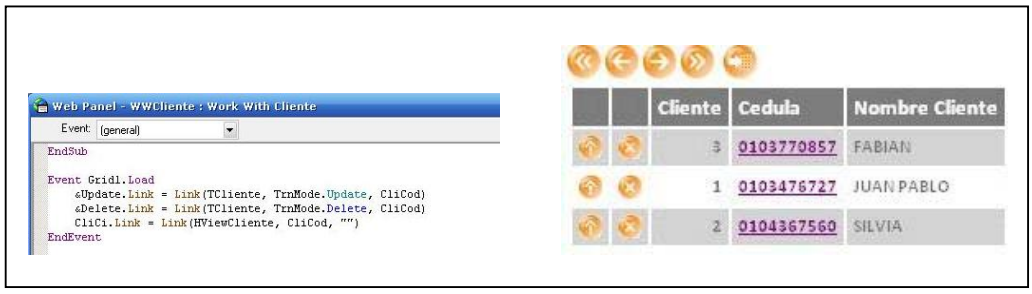
Navegación entre tablas relacionadas

Tabs

Fuente: Autor Tesis

Al utilizar el patrón tenemos la ventaja de que automáticamente se nos generan tabs con las tablas relacionadas y de esta manera se puede ver la relación y los movimientos que ha realizado esa instancia con los demás objetos.

Vinculo con las llaves principales



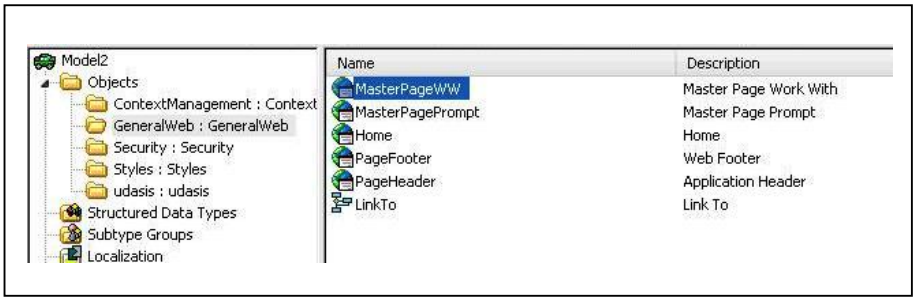
Gracias a los patrones se pueden generar vínculos con las llaves principales y cuando el usuario quiere conocer toda la información con respecto a un registro en especial basta con dar click sobre el vínculo para obtener toda la información de ese registro incluido las relaciones entre tablas.

4.4.4 Elementos del Patrón.

Masters Pages

Las **Master Pages** son templates que evitan que haya que establecer qué webcomponents usa cada Webpanel, por lo que disminuyen el esfuerzo de desarrollo. Además, otorgan mayor flexibilidad a la hora de modificar la aplicación porque para agregar un componente a todo el sitio Web basta agregarlo en la Master Page.

Para el caso del patrón udasis nos facilito la tarea gracias a esto se pudo tomar el mismo estilo en la posición y diseño de cada uno de los componentes.



Los componentes de la MasterPage son:

- **Page Header.-** Aquí ubicamos el logotipo del patrón o cualquier información que se quiere que este visible en todas las paginas y está ubicado en la parte superior.
- **Page Footer.-** Aquí ubicamos cualquier información que se quiera mostrar a lo largo de toda la aplicación esta puede ser links, contenidos, información del webmaster, etc. y está ubicado en la parte inferior de todas las páginas.
- **Home.-** Aquí se encuentra los accesos para todos los mantenimientos de las tablas que se están utilizando en la aplicación.
- **Recent Link.-** Aquí se ubicaran un link que nos conecta con el home y además se creara un link con las últimas páginas visitadas por el usuario.

Reportes en Excel

Para la emisión de reportes se ha elegido una herramienta que sea fácil de usar y que es muy difundida a continuación presentamos un extracto del templete que permite la generación de un procedimiento que envía reportes a Excel.

```
>>> Para definir las propiedades del procedimiento
<%@ CallSubTemplate ObjectInfoTemplate ObjName='objName' ObjDescription=
'objDescription' Folder='Folder' IsMain='false' %>
>>> Para definir las variables
<%
    ObjectVariables vars = new ObjectVariables(GXClass.Procedure);
    bool hasCustomVariables = DefineCustomVariables(GridNode, vars);
    vars.AddExtendedType("ExcelDocument", "Excel Document",
ExtendedType.Excel Document);
    vars.AddCommon("Filename", "File name", DataType.Varchar, 512, 0);
    vars.AddCommon("ErrorMessage", "Error Message", DataType.Varchar, 512, 0);
    vars.AddCommon("CellRow", "Cell Row", DataType.Numeric, 8, 0);
    vars.AddCommon("FirstColumn", "First Column", DataType.Numeric, 8, 0);
    vars.AddCommon("Random", "Random", DataType.Numeric, 8, 0);

>>> Para escribir los títulos
<%
    int column = 0;
    foreach (XmlNode itemNode in GridNode.SelectNodes("attributes/*"))
    {
%>
&ExcelDocument.Cells(&CellRow, &FirstColumn + <%= column %>).Bold =
Boolean.True
&ExcelDocument.Cells(&CellRow, &FirstColumn + <%= column %>).Text = '<%=
General.NodeDescription(itemNode) %>'
<%
    column++;
    }
%>

>>> Para escribir los datos y guardar el documento
foreach (XmlNode itemNode in GridNode.SelectNodes("attributes/*"))
{
```

```
&ExcelDocument.Cells(&CellRow, &FirstColumn + <%= column %>).<%= cellType %> = <%= itemValueExpression %>
<%
        column++;
}
%>
Endfor

&ExcelDocument.Save()
Do 'CheckStatus'
&ExcelDocument.Close()
```

Este código permite generar las instrucciones necesarias para enviar reportes a Excel.

```
&FirstColumn = 1

// Write titles
&ExcelDocument.Cells(&CellRow, &FirstColumn + 0).Bold = Boolean.True
&ExcelDocument.Cells(&CellRow, &FirstColumn + 0).Text = 'Cliente'
&ExcelDocument.Cells(&CellRow, &FirstColumn + 1).Bold = Boolean.True
&ExcelDocument.Cells(&CellRow, &FirstColumn + 1).Text = 'Cedula'
&ExcelDocument.Cells(&CellRow, &FirstColumn + 2).Bold = Boolean.True
&ExcelDocument.Cells(&CellRow, &FirstColumn + 2).Text = 'Nombre Cliente'
&ExcelDocument.Cells(&CellRow, &FirstColumn + 3).Bold = Boolean.True
&ExcelDocument.Cells(&CellRow, &FirstColumn + 3).Text = 'Apellido'
&ExcelDocument.Cells(&CellRow, &FirstColumn + 4).Bold = Boolean.True
&ExcelDocument.Cells(&CellRow, &FirstColumn + 4).Text = 'Dirección'
&ExcelDocument.Cells(&CellRow, &FirstColumn + 5).Bold = Boolean.True
&ExcelDocument.Cells(&CellRow, &FirstColumn + 5).Text = 'Mail'

For each
    order CliCi  when &OrderedBy = 0
    order CliNom  when &OrderedBy = 1

    where CliCi like &CliCi when not &CliCi.IsEmpty()

    // Write cell values
    &CellRow += 1
    &ExcelDocument.Cells(&CellRow, &FirstColumn + 0).Number = CliCod
```

La manera de enviar los datos a Excel es crear un documento extensión xls y comenzar a barrernos los datos de la tabla y ubicarlos en la hoja de Excel indicando el tipo de dato que deseamos ingresar y en que celda.

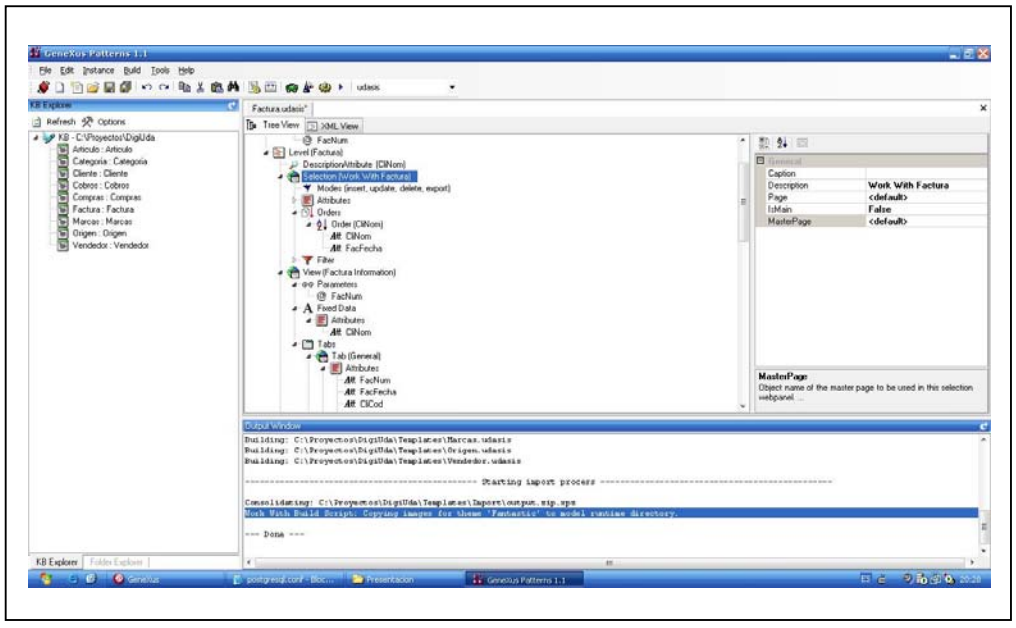
Reporte en Excel

Fuente: Autor Tesis

Cada grilla de los webpanels de mantenimiento tendrá una invocación a un procedimiento que generará un reporte de Excel.

4.5 Pruebas

4.5.1 Aplicación del patrón en la Base del Conocimiento.



El primer paso para poner en funcionamiento el patrón udasis es ingresar a la herramienta Patterns 1.1 de Artech y abrir la base del conocimiento en la que se desea trabajar. Una vez ubicado en la base de conocimiento, escogemos el patrón Udasis.

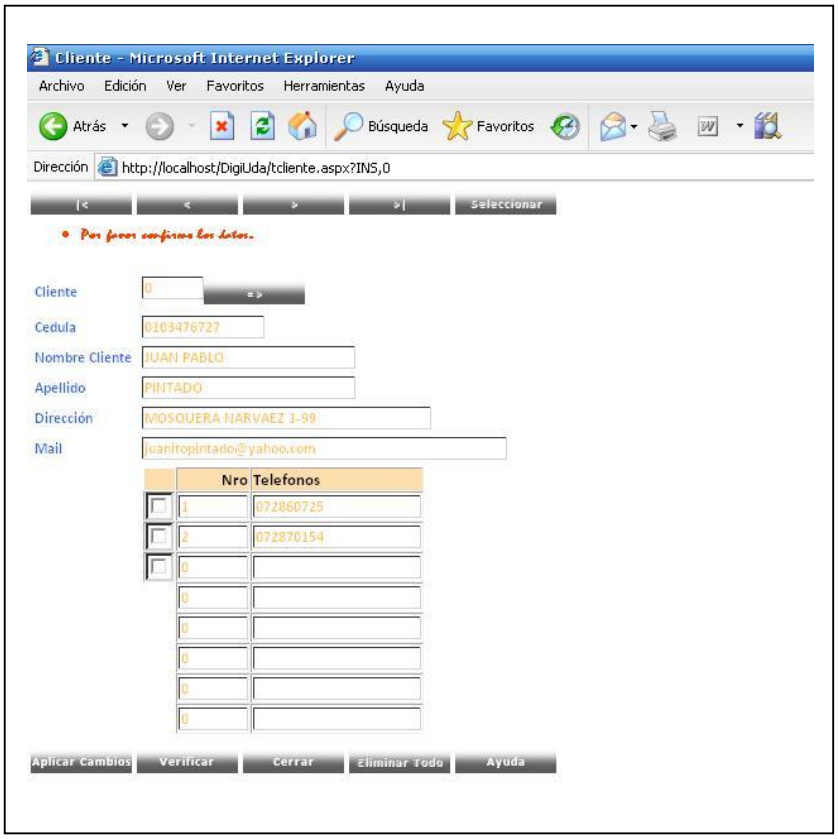
A partir de aquí ya se puede ejecutar el patrón tal como fue creado por defecto o se pueden aumentar filtro, búsquedas, ordenamiento y otras funciones que potencian el patrón dándole la funcionalidad que el programador está necesitando.

Una vez que ya está definido el comportamiento que se espera del patrón, se genera el patrón con los botones de aplicación, y si la aplicación es exitosa el indicador nos debe dar "**Done**" en la parte inferior de la pantalla, de lo contrario nos indicara el cual es el error.

4.5.2 Funcionamiento del patrón.

En esta parte del capítulo pondremos en funcionamiento una aplicación que esta generada con el patrón Udasis, así nos podremos dar cuenta de los beneficios que nos brinda.

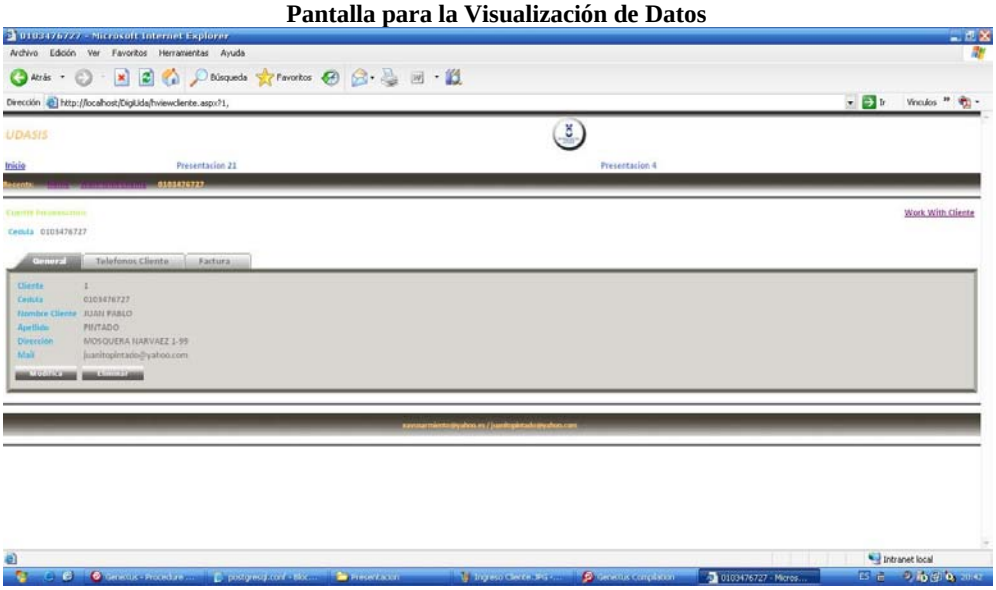
4.5.2.1 Ingreso de datos



Los datos que provienen de otras tablas podemos navegar para escoger el dato que deseamos insertar, de igual manera para los campos que almacenan fechas nos aparecerá un calendario para poder elegir la fecha. Una vez insertados los datos nos aparecerá un mensaje de confirmación si todos los datos están correctamente ingresados, caso contrario nos aparecerá un mensaje de error.

4.5.2.2 Visualización de datos

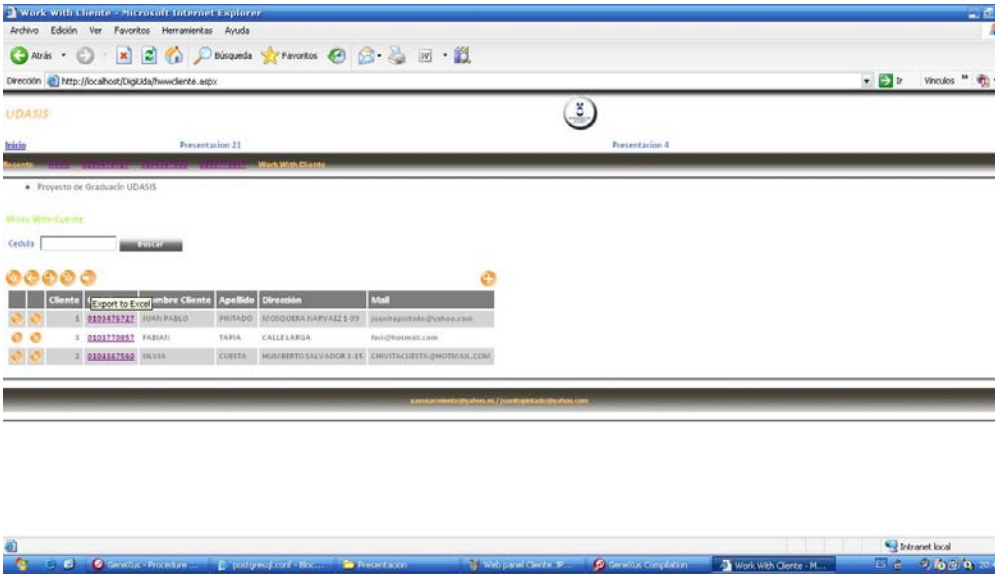
Pantalla para la Visualización de Datos



Fuente: Autor Tesis

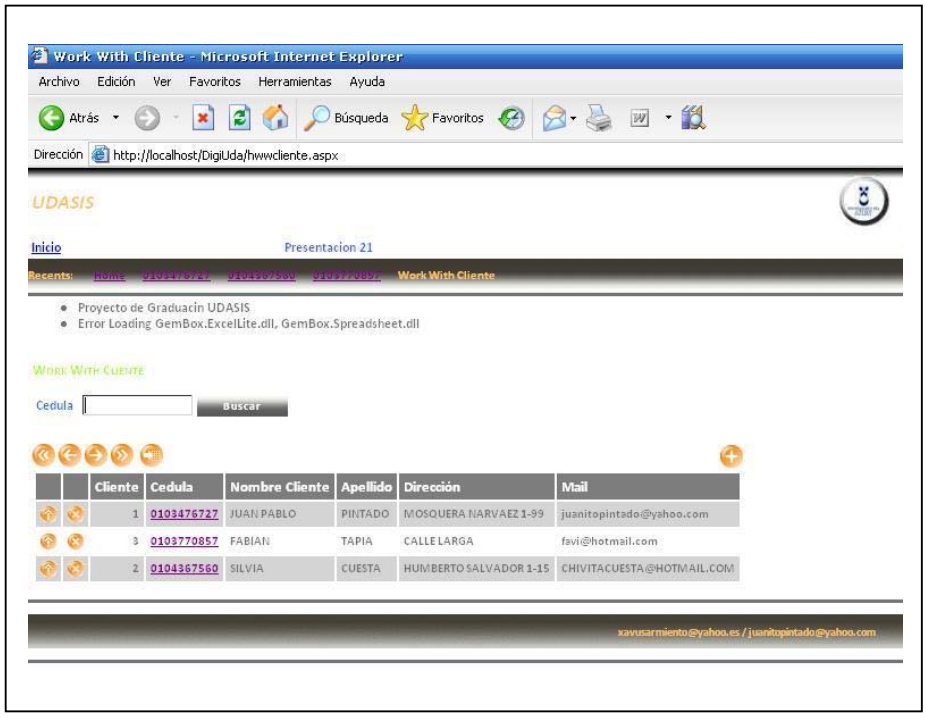
Una vez ingresados los datos nos muestra una pantalla con los datos que fueron almacenados en la base, además se pueden realizar directamente ingresos a otras tablas relacionadas.

4.5.2.3 Manipulación de los datos



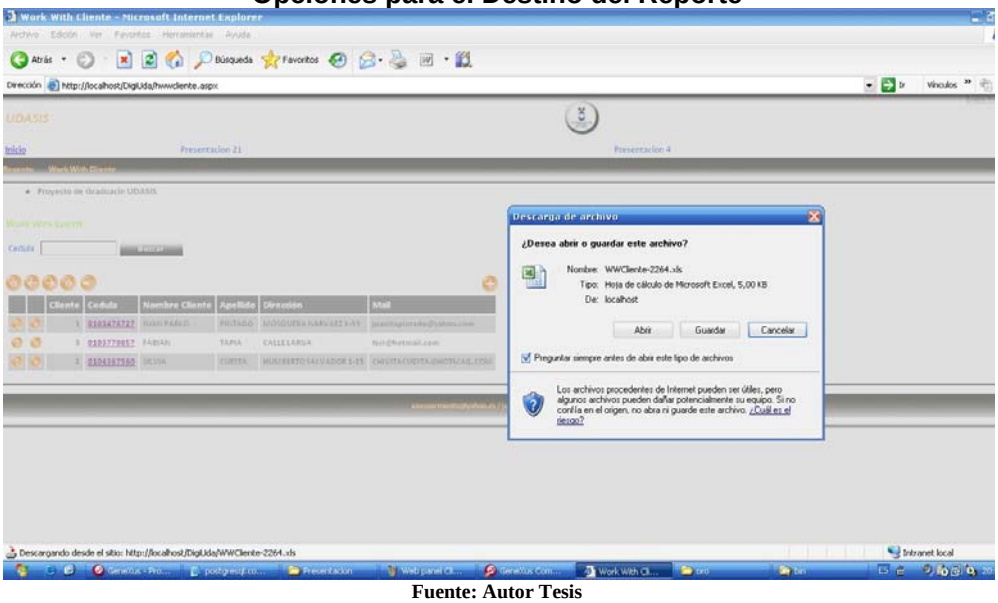
Una vez ingresados los datos gracias al patrón es mucho más fácil manipular los datos; es decir, modificar los datos ingresados, eliminar, buscar, ordenar, navegar y realizar un reporte. Los botones a pesar de ser intuitivos tienen una leyenda que nos indica la función que realiza, dicha leyenda se muestra al pasar el puntero del ratón sobre el botón.

4.5.2.4 Reportes en Excel



Para poder enviar los datos a Excel es necesario instalar una dll que nos permite la conexión del browser con la herramienta Microsoft Excel; de no contar con la dll nos dará un erro y no podernos realizar la exportación de los datos.

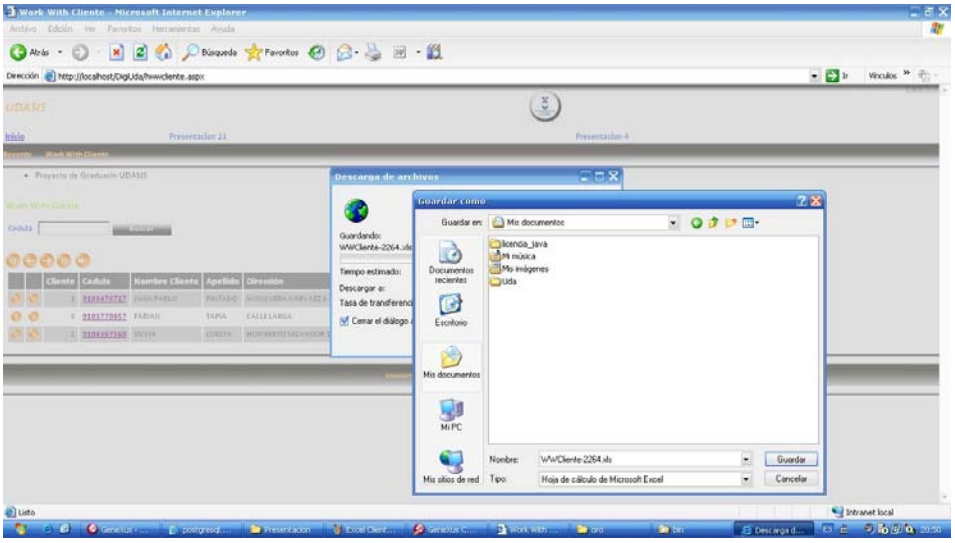
Opciones para el Destino del Reporte



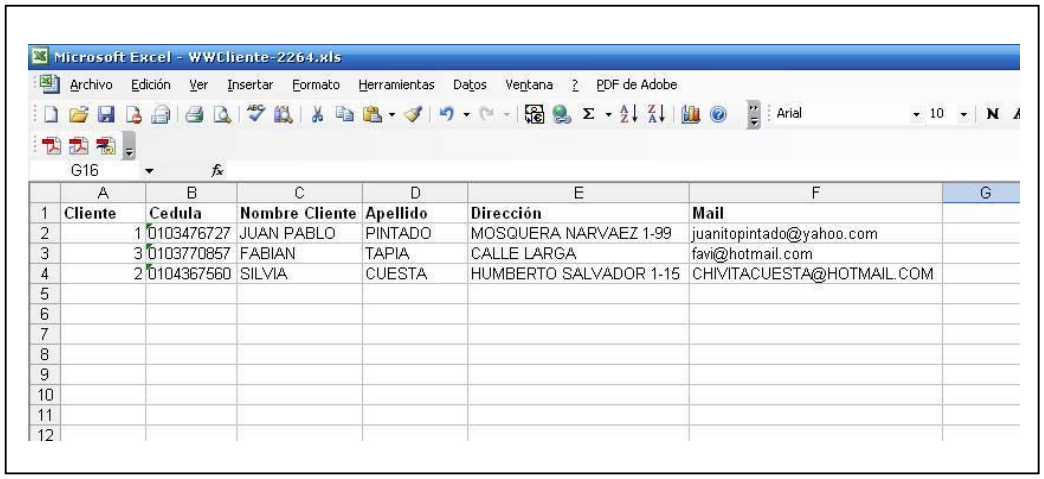
Fuente: Autor Tesis

Los reportes pueden ser abiertos directamente en el navegador si se desea únicamente observar los datos, caso contrario si se desea modificar, imprimir o exportar a un formato pdf se tiene la opción de guardar para luego realizar las acciones antes nombradas.

El patrón genera un nombre por defecto que identifica la tabla de la cual se está realizando el reporte. En caso de no desear ese nombre se puede asignar el nombre que el usuario desee.



El reporte se podrá ser guardado en cualquier lugar de la maquina o dispositivo de almacenamiento que el usuario desee.



	A	B	C	D	E	F	G
1	**Cliente**	**Cedula**	**Nombre Cliente**	**Apellido**	**Dirección**	**Mail**	
2	1	0103476727	JUAN PABLO	PINTADO	MOSQUERA NARVAEZ 1-99	juanitopintado@yahoo.com	
3	3	0103770857	FABIAN	TAPIA	CALLE LARGA	favi@hotmail.com	
4	2	0104367560	SILVIA	CUESTA	HUMBERTO SALVADOR 1-15	CHIVITACUESTA@HOTMAIL.COM	
5							
6							
7							
8							
9							
10							
11							
12							

El reporte en Excel es una excelente opción para que el usuario manipule los datos y pueda realizar los cálculos que crea conveniente.

4.6 Conclusiones

En este capítulo se explica con la mayor claridad los componentes que son necesarios para la construcción de un patrón que sirva para la herramienta case Genexus. Se muestra los pasos que se beben seguir cuando uno quiere o necesita realizar un patrón.

Como se mostro se tiene que tener las bases para poder construir un patrón, identificar los problemas los cuales se quieren resolver y conocer a detalle las herramientas que sirven para su elaboración.

También se ve como se recalca que el patrón, sea lo más fácil de utilizar por el usuario y brindarle todas las herramientas necesarias para cumplir con los requerimientos de la persona que manipule el sistema.

Glosario

KB: Base del conocimiento, donde se guardan todas las transacciones y reglas del negocio de sistema.

GX: Abreviatura de Case Genexus 9.0.

TRN: Transacción.

Archivos dkt: Son archivos comprimidos que utiliza Genexus para generar objetos y dar funcionalidades a las aplicaciones.

Referencias

http://www.gxtechnical.com/gxdlsp/pub/genexus/devenv/docum/releasenotes/8.0/Cthemes.htm

http://www.genexus.com/

http://www.gxopen.com

CONCLUSIONES

Como se vio, el desarrollo del patrón hizo que se puedan realizar muchas tareas requeridas sin tener que programar demasiado. Así, de esta manera logramos que el programador se enfoque solo en la programación puntual sin preocuparse en detalles de diseño.

El uso de patrones permite mantener un estándar durante el desarrollo de la aplicación, tanto en el diseño como en la codificación, permitiendo de esa manera reducir el tiempo en fijar estándares.

Una de las cosas muy relevantes del patrón realizado es los reportes se generaron en un archivo para Excel .xls, puesto que los usuario puede manipular a su gusto la información generada y calculada por el sistema, sin limitarlos a un formato estándar impuesto por nosotros.
Se puede palpar que la utilización de patrones reduciría en gran cantidad el plazo de desarrollo de un sistema, y este tendría una alta tasa de satisfacción ya que gracias a esto se evitaría de escribir código y de esta manera tener errores típicos.

El uso de los patrones con la herramienta CASE Genexus, permite a los programadores potenciar la aplicación, ya que amas de los objetos generados por el patrón, se pueden añadir más objetos dependiendo de la necesidad y los requerimientos de la aplicación.

RECOMENDACIONES

Es muy importante que antes de empezar el desarrollo de los patrones que se conozca muy bien el lenguaje CSharp, ya que con esta se elaborara la mayor parte del patrón, también se debe tener un conocimiento amplio en XML ya que en este formato se armara la mayoría de pantallas y los estilos para las mismas, se debe tener una idea en Dreamweaver ya que esta herramienta nos ayudara a diseñar las pantallas del patrón.

El conocimiento previo del case Genexus 9.0 es también la base para poder desarrollar el patrón, se debe conocer mucho de esta herramienta ya que gracias a esta se pueden consolidar los objetos de Genexus y así podemos darnos cuenta de mejor manera como están creados los objetos y como se dan los cambios.

Analizar con mucha precaución las herramientas de desarrollo que se pretendan usar y sobre todo tener cuidado de cambiar de tecnología o de herramientas nuevas y desconocidas a lo largo del camino, ya que puede volverse el proyecto demasiado largo.

Con respecto a la instalación de nuestro software (ANEXOS), una vez que se encuentre instalado, se debe tener mucha precaución con ejecutar el proceso crear base de datos debido a que si no es la primera vez que se ejecute este procedimiento se borrarán todos los datos.

Además como requisitos de funcionamiento asegurarse de instalar previamente el motor de la base de datos MySQL y para los reportes .xls Microsoft Excel.

REFERENCIAS

BIBLIOGRAFIA

Libros

UML y Patrones. Introducción al análisis y diseño orientado a objetos - Larman - Prentice Hall

Design Patterns. Elements of Reusable Object-Oriented Software - Gamma, Helm, Johnson, Vlissides - Addison Wesley

Castañeda G., Víctor. Revista Tecnología de Punta.

Herramientas para el desarrollo de Sistemas de Información. http://www.inei.gob.pe/cpi/bancopub/libfree. Instituto de Estadística e Informática. Lima Perú. Julio, 1997.

Sitios de Internet

http://www.geocities.com/SiliconValley/lab/7538/

http://es.wikipedia.org/wiki/Herramienta_CASE

http://ceds.nauta.es/Catal/Products/caselist2.htm

http://www3.uji.es/~mmarques/f47/apun/node75.html

http://www.iscmolina.com/Herramientas%20CASE.html

http://www.vbdotnetheaven.com/Code/Jun2003/2014.asp

http://www.gxtechnical.com/gxdlsp/pub/genexus/devenv/docum/releasenotes/8.0/Cthemes.htm

http://www.genexus.com/

http://wiki.gxtechnical.com/commwiki/servlet/hwiki?UseThemeeditor

http://www.acpsistemas.com.ar/gxpsites/hgxpp001.aspx?1,6,182,O,S,0,

http://blogs.vbcity.com/jspano/articles/198.aspx

http://www.genexus.com/portal/hgxpp001.aspx?2,32,666,O,S,0,MNU;E;87;2;MNU

http://www.gxopen.com

http://www.gxopen.com/commwiki/servlet/hwiki?Patterns

ANEXO 1

¿Qué es Genexus?

GeneXus es una poderosa herramienta para el diseño y desarrollo de software multiplataforma. Permite el desarrollo incremental de aplicaciones críticas de negocio de forma independiente de la plataforma.

GeneXus genera el 100% de la aplicación. Basado en los requerimientos de los usuarios realiza el mantenimiento automático de la base de datos y del código de la aplicación, sin necesidad de programar.

GeneXus hace posible que los clientes tengan sistemas actualizados, tanto a la realidad empresarial como tecnológica, y pueden concentrarse en su negocio sin preocuparse por la evolución de la tecnología, permitiéndoles migrar hacia cualquier plataforma, gracias al diseño de una base de conocimiento independiente de cualquier lenguaje, base de datos, sistema operativo o arquitectura.

GeneXus soporta las plataformas y lenguajes líderes y los DBMS más populares.

TECNOLOGIAS SOPORTADAS

- **Plataformas**
  Plataformas de ejecución
  JAVA, Microsoft .NET, Pocket PC.

- **Sistemas Operativos**
  IBM OS/400, LINUX, UNIX, Windows NT/2000/2003 Servers, Windows NT/2000/XP, y Windows Mobile.

- **Internet**
  JAVA, ASP.NET, Visual Basic (ASP), HTML, WebServices.

- **Bases de Datos (DBMS)**
  IBM DB2, Informix, Microsoft SQL Server, MySQL, Oracle, PostgreSQL.

- **Lenguajes**
  JAVA, C#, COBOL, RPG, Visual Basic, Visual FoxPro.

- **Servidores Web**
  Microsoft IIS, Apache, WebSphere, etc.

- **Arquitecturas**
  Arquitecturas de múltiples capas, basadas en web, Cliente/Servidor, y centralizadas.

- **Herramientas de Business Intelligence y Workflow**
  Soluciones para Reporting, Data Warehousing, y Workflow para todos los servidores soportados.

La Versión de GeneXus versión 9.0. Incluye como novedades:

- Desarrollo de aplicaciones Web más rápido: Gracias a los Patterns que hará el trabajo por usted.
- Experimente lo último en aplicaciones Web: AJAX!! Cree un cliente inteligente con validaciones propias.
- Venda su aplicación en casi cualquier lenguaje extranjero haciendo pocos o ningún cambio en el código, usando Application Localization.
- Acceso a más bases de datos que nunca: Ha sido agregado el soporte a MySQL.
- Cree aplicaciones más fáciles de usar.
- Encapsule la lógica de su aplicación con Businnes Components y luego cónsumalos usted mismo o deje que otros lo hagan mediante Web Services o J2EE.
- Herede bases de datos más rápido y fácil. Use la nueva Database Reverse Engineering Tool.

Productividad

GeneXus 9.0 Productividad: GeneXus 9.0 fue diseñado con el objetivo de brindar un aumento de la productividad de un orden de magnitud en lo que Ud. y su

organización pueden lograr por unidad de tiempo. Para ello se incorporan las siguientes herramientas:

GeneXus Patterns Herramienta que permite la generación automática de objetos GeneXus que siguen un patrón determinado a partir de otros objetos GeneXus. Esto permite automatizar gran parte del trabajo repetitivo que se realiza creando objetos dentro de la propia Base de Conocimiento, antes de generar la aplicación en la DBMS y lenguaje seleccionados. Se pueden crear y adaptar Patterns a las necesidades de cada proyecto y cliente. Asimismo, existen Patterns creados por Artech a disposición de todos los usuarios GeneXus. Algunos escenarios de uso y su correspondiente Pattern:

- Migración de aplicaciones a Web – Pattern "WorkWith"
- Parametrización de aplicaciones en tiempo de ejecución – Pattern OAV (Object Attribute Value)
- Diseño de modelo de datos con características específicas – Pattern "Bill of Materials"

Master Pages Una Master Page permite definir una plantilla o "template" para página web donde se pueden ubicar múltiples Web Panels, los objetos utilizados en GeneXus para el desarrollo de aplicaciones Web. La Master Page está dividida en contenedores donde se insertan los Web Panels y donde cada Web Panel puede llamar a otros Web Panels, independientemente de lo que sucede en el resto de la página. Las Master Pages facilitan el desarrollo incremental de aplicaciones Web, separando la estructura de la interfase de usuario de la lógica de la aplicación: "Write once, use may times". Asimismo, simplifican grandemente el mantenimiento, ya que cambiando una Master Page, cambian todas las páginas asociadas a la misma.

Componentes El objeto transacción es uno de los principales objetos GeneXus para el diseño de aplicaciones. Cada transacción tiene una estructura de datos, fórmulas, reglas de negocio, y un formulario Win y otro Web asociados. A partir de ese objeto se genera un programa de altas, bajas y modificaciones sobre un conjunto de tablas. De este modo se tiene una interface de usuario a través de la cual se modifican datos controlando la integridad (tanto de la base de datos como de las reglas de negocio). Los Componentes permiten utilizar la integridad referencial, fórmulas, y reglas de negocio de una transacción sin utilizar su formulario. El Componente puede ser utilizado por otros objetos de la Base de Conocimiento para actualizar los datos o

inclusive por programas externos (vía web services). Su uso dentro de la Base de Conocimiento es mediante propiedades y métodos que expone el Componente. El principal beneficio de los Componentes es re-utilizar el conocimiento almacenado en un objeto transacción sin necesidad de repetir el mismo en otros objetos con la consiguiente ganancia de productividad e integridad. Asimismo, facilita la interacción e integración de las aplicaciones GeneXus con programas externos.

Otras funcionalidades destacables El **Theme Editor** se hizo extensible, permitiéndole al usuario crear sus propias clases. También se le agregaron más funcionalidades de CSS (Cascade Style Sheets).

Se incorporaron varias funcionalidades para mejorar la **experiencia del desarrollador** y al mismo tiempo hacer el entorno de desarrollo más productivo.

Desarrollo y mantenimiento de aplicaciones

GeneXus 9.0 Desarrollo y mantenimiento de aplicaciones: GeneXus 9.0 continúa con la tradición de permitirle desarrollar y mantener aplicaciones de nivel corporativo sin la necesidad de dominar cada una de las tecnologías involucradas. Como en cada nueva versión se incorporan las últimas tecnologías, haciendo de GeneXus una herramienta cada vez más poderosa. La **Interfase de usuario más interactiva** es una de las tendencias más importantes en las aplicaciones modernas, principalmente en las aplicaciones Web, es la búsqueda de una experiencia más interactiva para el usuario final. GeneXus 9.0 incorpora la arquitectura Ajax para la generación de su interfase de usuario. GeneXus 9.0 incorpora cuatro nuevas funcionalidades en este sentido (todas ellas aplicables en entornos Win y Web):

Web Client-Side Validation: permite que las fórmulas y reglas en una aplicación Web sea evaluadas del lado del cliente, es decir sin tener que hacer un viaje hasta el servidor.

Input Type: Permite ingresar un registro, ingresando su descripción, no su ID. Por ejemplo: ingresar un país por su nombre y no por su ID.

Suggest: Propiedad asociada los registros cuya descripción comienza con los caracteres ingresados hasta el momento. Por ejemplo: si el usuario ingresa la letra "A" en el campo país, se desplegarán como opciones todos los países que comienzan con la letra "A".

Combos dinámicos: Si tengo en mi KB una estructura con países y ciudades, esta propiedad me permitirá que en el combo de ciudades solo se muestren las ciudades

pertenecientes al país seleccionado en el combo de países.

Ingeniería Inversa de Bases de Datos Cada vez son más raras las aplicaciones dónde se parte de una base de datos en blanco, dónde ni el diseño y de las base de datos, ni sus datos existen aún. La nueva Database Reverse Engineering Tool incluida en GeneXus 9.0 es la poderosa sucesora del Data View Generator y permite capturar el conocimiento contenido en bases de datos preexistentes e importarlos a sus Bases de Conocimiento.

Propiedad Null La nueva propiedad Null permite definir si un atributo puede tomar valores "null" o no. Esta definición explícita redunda en mayor seguridad de los datos, mejoras en los controles de integridad referencial, y permite crear Joins con mejor performance.

Monitoreo y Administracion de aplicaciones GeneXus permite crear aplicaciones corporativas realmente grandes y altamente concurrentes. Por tanto, es necesario poder conocer qué está pasando los servidores de aplicaciones y de base de datos. GeneXus 9.0 incorpora una herramienta que permite monitorear aplicaciones Web y 3 capas. La misma permite no solo monitorear, sino realizar cambios en tiempo de ejecución a la configuración de algunas propiedades del servidor de aplicaciones.

Protección de su inversión: más plataformas y mejor preparación para el futuro Como siempre, GeneXus 9.0 le permite desarrollar y mantener aplicaciones de negocios en todas las plataformas líderes del mercado y al mismo tiempo comenzar a desarrollar hoy las aplicaciones que necesitará mañana.

Soporte para MySQL GeneXus 9.0 incorpora el soporte para el DBMS Open Source más popular en la actualidad. MySQL se puede usar con todos los Generadores GeneXus. Todas las bases de datos soportadas por GeneXus 9.0 son: IBM DB2, Informix, Microsoft SQL Server, MySQL, Oracle, y PostgreSQL.

Miles de empresas de mediano y gran tamaño y casas de software en todo el mundo utilizan GeneXus para desarrollar complejos sistemas de misión crítica con grandes bases de datos que comprenden desde sistemas centralizados, distribuidos hasta aplicaciones Web. Estos sistemas integran módulos de data warehouse, web services, portales corporativos y mucho más.

GeneXus es una herramienta inteligente de desarrollo para construir y mantener sistemas, de una manera simple.

Le permitirá a usted y a su equipo de desarrollo crear sistemas fácilmente, permitiéndole trabajar en múltiples plataformas, ya sea de sistemas operativos, lenguajes de programación o motores de bases de datos.

ANEXO 2

INSTALACION DE GENEXUS 9 Trial

GeneXus Trial Versión

Restricciones Funcionales

La GeneXus Trial Versión es completamente funcional y todos los generadores disponibles son autorizados mediante una única Site Key (sin fecha de expiración). No obstante, se aplican algunas restricciones respecto al número máximo de objetos GeneXus que se pueden crear para una Base de Conocimiento dada:

• Transacciones: 30
• Work Panels (incluyendo GeneXus selection prompts): 50
• Web Panels (incluyendo GeneXus selection prompts): 50
• Procedimientos: 20
• Reportes: 20

Otras restricciones importantes son:

• Las bases de conocimiento generadas con la GeneXus Trial Versión no pueden abrirse con una versión estándar de GeneXus y viceversa.

• La opción "Distribution" de la Base del Conocimiento está deshabilitada. Los archivos con extensión xpz creados con versiones GeneXus Trial pueden consolidarse (siempre que no excedan los límites mencionados más arriba). Es decir que se pueden consolidar proyectos pero no distribuirlos.

• GeneXus Patterns no puede ser utilizado junto con la GeneXus Trial Versión porque no existe una versión trial de GXpublic.

Restricciones de Licencia

La GeneXus Trial Versión puede ser utilizada exclusivamente para evaluación y capacitación. Su instalación es local y para un solo usuario. La instalación en red no está habilitada.

Soporte Técnico

Si necesita ayuda para la instalación y autorización de su versión trial, contáctese con:

xavusarmiento@yahoo.es

Instalación y Configuración

1. Ejecute el archivo de setup de la GeneXus Trial Versión (GX90Trial.exe) desde el menú de Inicio de Windows o seleccionando la opción apropiada en el CD de su Versión Trial.

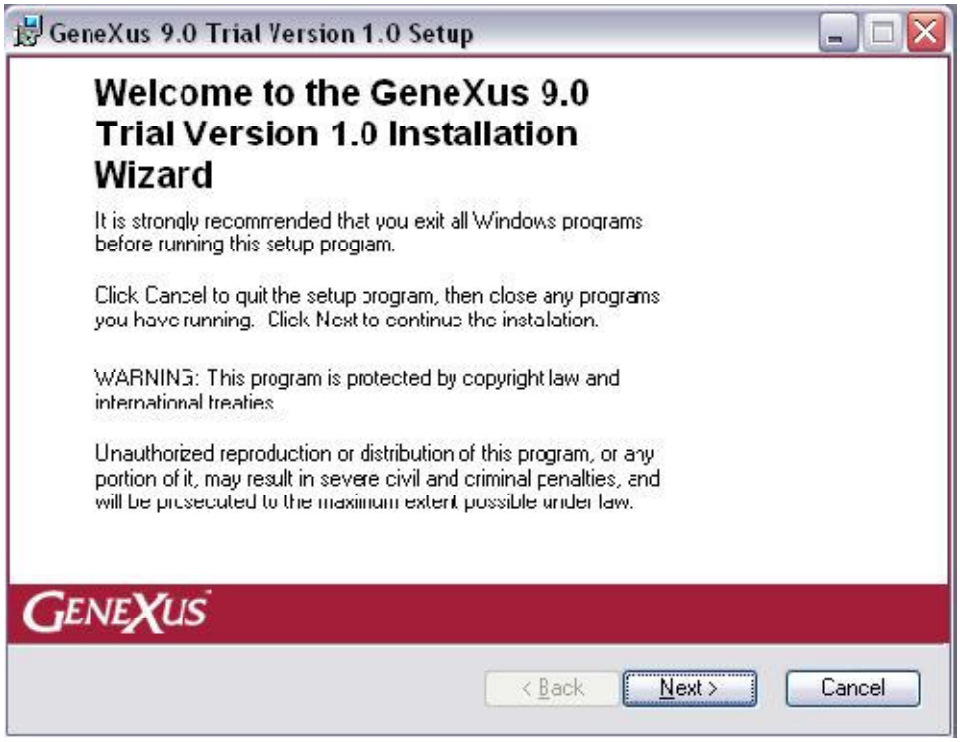


Figura 1 Asistente de Configuración de la GeneXus Trial Versión

2. Siga las instrucciones del GeneXus Trial Versión Setup Wizard. Debe instalar como mínimo el GeneXus Development Environment y uno de los Generadores GeneXus.

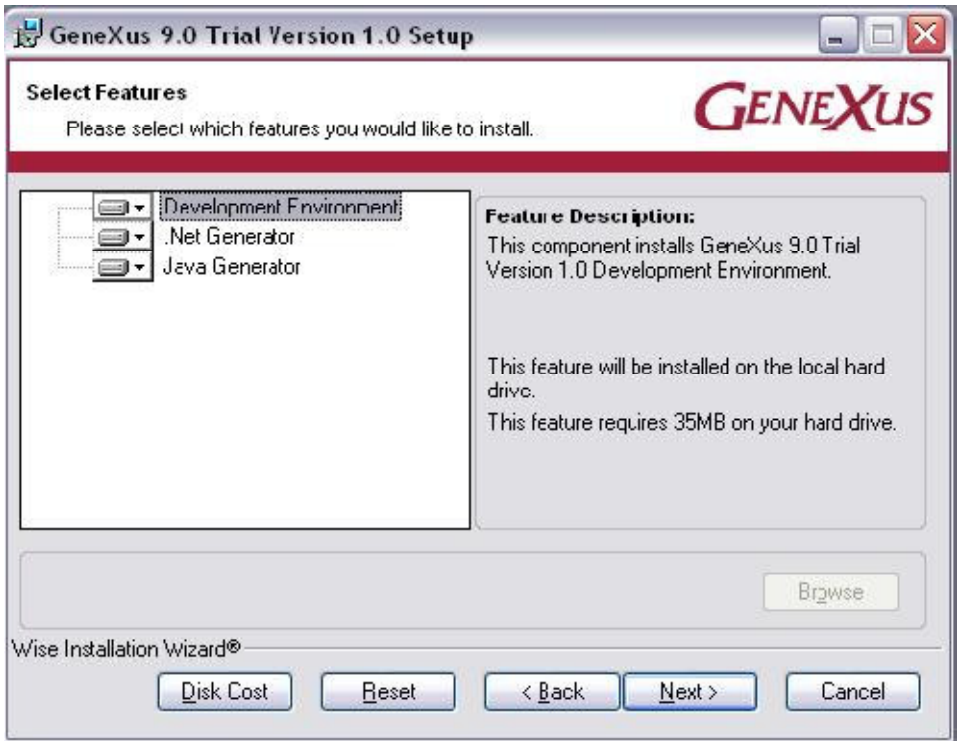


Figura 2 Cuadro de dialogo de Selección de Componentes GeneXus

Autorización de la GeneXus Trial Versión

Debe autorizar su GeneXus Trial Versión la primera vez que la use. Para ello:

1. Ejecute la GeneXus Trial Versión desde el acceso directo del escritorio o desde el menú de Inicio.

2. Escriba o copie su Site Code desde la ventana GeneXus Trial Versión Registración que se desplegará. CONSEJO: No cierre esta ventana por ahora.

Figura 3 Cuadro de dialogo para el Registro de la GeneXus Trial Versión

3. Luego vaya a http://www.genexus.com/trial/authorize. Con esto se abrirá un formulario web de actualización de la GeneXus Trial Versión donde deberá registrarse. Use su usuario y contraseña de GXtechnical, es decir, el mismo nombre de usuario y contraseña que uso para bajar la GeneXus Trial Versión.

4. Pegue el Site Code en el campo apropiado y haga clic en Submit.

5. Recibirá su Site Key vía e-mail.

6. Escriba o pegue su Site Key en el campo de Site Key la ventana GeneXus Trial Versión Registración y haga clic en OK. Aparecerá un mensaje aceptando su Site Key.

Figura 4 Mensaje de aceptación de la Site Key

7. Haga clic en OK nuevamente. Usted ya está listo para comenzar a usar su GeneXus Trial Versión.

http://www.genexus.com/portal/hgxpp001.aspx?2,32,583,O,S,0,MNU;E;130;1;MNU;,&gclid=CMjAmcir-5ECFRwgkgoddQE_oQ

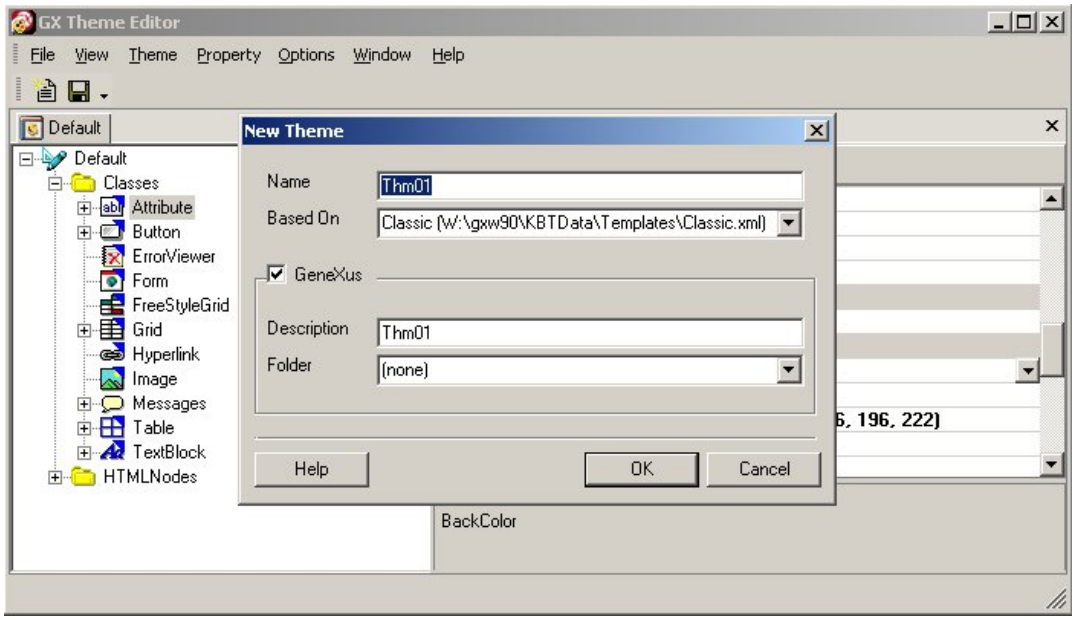
ANEXO 3

Ejemplo de uso de Themes

El siguiente es un ejemplo básico de uso de Themes.

1. Crear un nuevo Theme:

Por el menú de GeneXus, ir a Object->New Object, y crear un objeto GeneXus Theme.

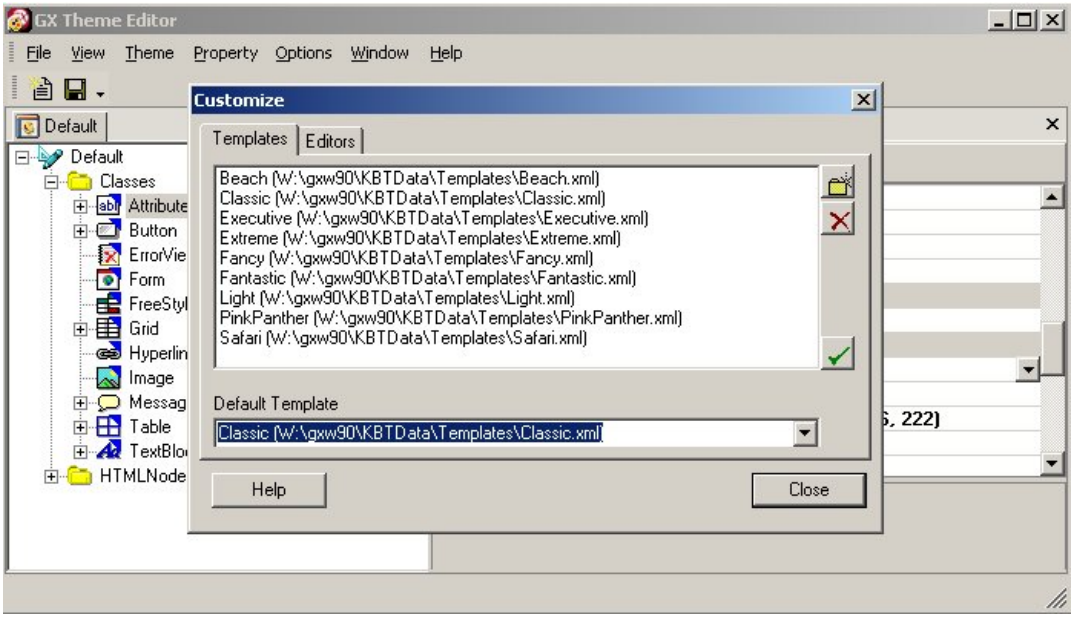


2. Realizar los cambios requeridos y salvarlo.

Para este ejemplo, cambiar el Backcolor de la clase “Attribute”. Salvar los cambios.

3. Asociar el Theme al objeto Web y el control a la clase.

Editar las properties del objeto, y cambiar la propiedad Theme para que se corresponda con el Theme.



Verificar que el control edit quede asociado a la clase modificada (editar las propiedades del control, y verificar la propiedad “Class”)

Editor de Themes

El editor de Themes se despliega como una estructura jerárquica; los nodos descendientes del nodo raíz son “Clases” y “HTML tags”:

Clases

A partir del nodo “Classes” se despliega un conjunto de clases predefinidas, correspondientes a controles GeneXus.

Las clases predefinidas son:

- Attribute
- Button
- Error Viewer
- FreeStyleGrid
- Grid
- Hyperlink
- Image
- Table
- TextBlock
- Form

Cada clase reúne un conjunto de propiedades configurables por el diseñador.

Un Theme puede ser asociado a un objeto GeneXus a través de la Propiedad Theme, que se encuentra a nivel de objeto, modelo y base de conocimiento.

Una vez que un Theme es asociado a un objeto, las clases definidas en el Theme pueden ser asociadas a los controles de ese objeto, tanto en diseño como en tiempo de ejecución a través de la Propiedad class.

Herencia de clases

La jerarquía de clases, define implícitamente una herencia de clases. Una clase que es hijo de otra clase heredará las propiedades de su clase padre, mientras la propiedad en cuestión no haya sido modificada en el hijo. Es decir, cambios en una propiedad a nivel de una clase padre se reflejarán en los hijos a menos que esa propiedad haya sido modificada previamente en el hijo, por lo cual pierde la calidad de valor heredado.

Esta "herencia" busca brindarle al usuario mayores facilidades en la tarea del mantenimiento.

Por ejemplo si se define una clase para los controles Textblock “Titulo”.

Si se quiere otra clase para los Textblock que sean subtitulos, probablemente se quiera que esta última clase tenga las mismas caracteristicas que la anterior, salvo algunas excepciones, por ejemplo, el tamaño del font.

Entonces, se crearía una clase hija de “Titulo” llamada “Subtitulo”.

Por lo cual, cualquier cambio en la clase “Titulo” se refleja en la clase “Subtitulo” – salvo el tamaño del font-para el cual se perdió la herencia, y ese es precisamente el comportamiento que se espera.

Si por el contrario se quiere tener una clase para los textblocks dentro de una tabla (“TextInTable”), entonces se crearía como hija de la clase predefinida “Textblock”– independientemente de las otras (“Título” y “Subtítulo”). Porque los cambios en éstas se desea que sean completamente independientes a “TextInTable”.

Qué es específicamente un Theme?

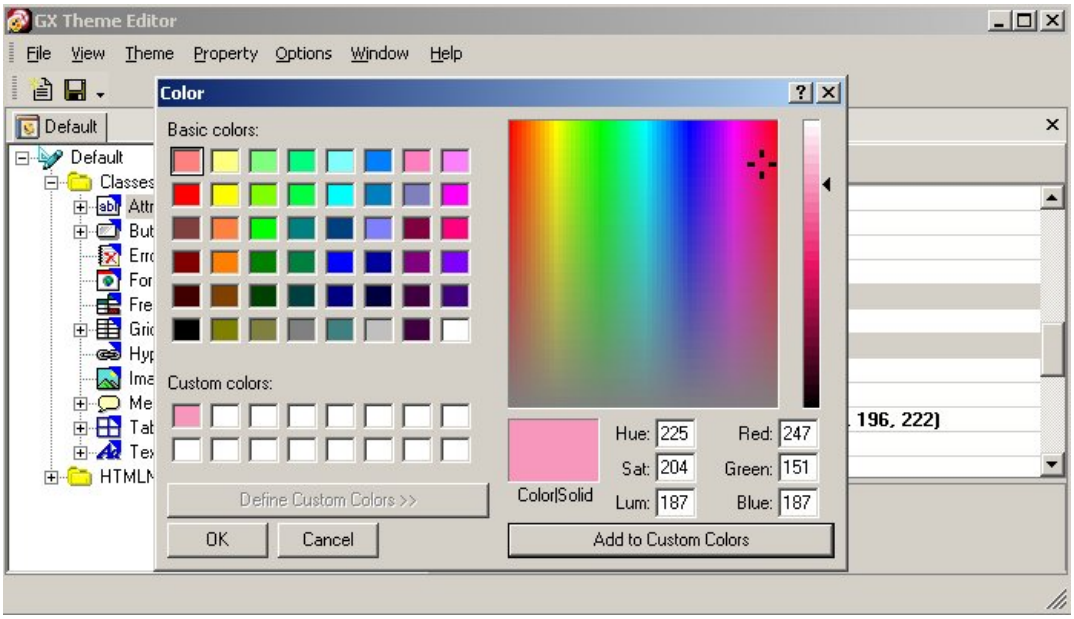
Un Theme GeneXus hoy se genera como un Cascading Style Sheet (CSS) que contiene diferentes estilos para todos los controles.

El editor de temas permite salvar los objetos Theme como CSS, que serán referenciadas en las páginas web generadas por GeneXus.

Un style sheet es un documento que contiene reglas que especifican los estilos de un HTML, y por lo general se hace referencia a él en vez de incluírlo en el HTML mismo.

Los style sheets permiten separar el contenido de un documento HTML, de su presentación (de los estilos que le aplican).

Los Themes GeneXus tienen un valor agregado que es el hecho de que se definió una Jerarquía sobre el estándar de CSS; eso le permite al usuario trabajar con los CSS en forma cómoda y totalmente transparente.



Es decir que el usuario GeneXus no tiene necesidad en absoluto de conocer la tecnología CSS para hacer uso de ella en forma 100% efectiva, a través del editor de Themes.

El hecho de incorporar Cascading Style sheets a las páginas web amplia el espectro de propiedades configurables. Por lo tanto, a través del editor de temas se podrá configurar propiedades adicionales a los controles.

Adicionalmente, se genera menos código y mejora notoriamente la performance en ambiente web.

Los estilos están incorporados en un CCS que se agrega como un link en el html generado.

Entonces, el CSS (que podría ser grande) se transfiere al cliente sólo cuando es estrictamente necesario. Esto es, cuando se accede a una página que lo referencia por primera vez y cuándo se modifica el CSS.

Si varias páginas referencian al CSS, sólo la primera debería verse afectada por la carga adicional de transferir el CSS.

ANEXO 4

MANUAL DE USO DE PATRONES PARA GENEXUS 9

Los Patterns es una herramienta parte de Genexus 9.0 (Tools / Patterns), puede ejecutarse también como aplicación independiente. Básicamente permite aplicar a una Base de Conocimiento cierto patrón (pattern), y que se generen todos los objetos Genexus necesarios para implementar el patrón, para aquellas instancias que se hayan seleccionado.

Existe un conjunto de patrones muy útiles ya definidos (al instalar la herramienta se instalan) para que el desarrollador pueda aplicarlos a sus Bases de Conocimiento rápidamente, obteniendo por resultado funcionalidades útiles generadas automáticamente. A su vez la herramienta permite crear patrones nuevos, siendo esta una tarea un poco más compleja.

Algunos patrones (patterns) existentes:

- Work With
- Bill of Materials
- OAV

Work With: Genera a partir de una transacción (o para todas aquellas transacciones que se deseen), todos los objetos necesarios para tener una aplicación web.
Por ejemplo, si se aplica el pattern "Work With" a la transacción "Customer", se generará un objeto Genexus Web Panel que permitirá al usuario final consultar interactivamente los clientes de la base de datos (se presentará una página web vistosa conteniendo una lista con los clientes de la tabla CUSTOMER).

El Web Panel se llamará "Work With Customers" y ofrecerá entre tantas cosas, un link asociado a cada cliente mostrado en la lista, para que mediante su selección se acceda a otro objeto Web Panel (de nombre "View Customer") que mostrará todos los datos del cliente (y su información relacionada como podrían ser facturas, recibos, etc.).

El Web Panel "Work With Customers" además ofrecerá para cada cliente mostrado, la posibilidad de modificar sus datos (habiéndose agregado automáticamente para

ello una invocación a la transacción “Customer”) así como la posibilidad de eliminar un registro de cliente, o de insertar un cliente nuevo (invocando a la transacción “Customer” para ello también).

Se podrán configurar variadas propiedades para agregar opciones de filtrado en el Web Panel “Work With Customers” (por ejemplo para que el usuario final pueda consultar solamente los clientes de cierto país, o aquellos cuyos nombres se encuentren incluidos en determinado rango).

Una simple propiedad se podrá incluir en el Web Panel “Work With Customers” un combo box que ofrezca distintos órdenes posibles, para que el usuario final elija si desea el resultado de la consulta ordenado por nombre de cliente, por código u otro atributo.

Si el patrón **Work With** se aplicara también a la transacción “Product” obtendríamos todas estas funcionalidades para dicha instancia también, así como para cada una de las transacciones que se deseen.

Bill of materials: Este patrón permite generar a partir de una transacción, otra que representa la relación compuesto – componente.
OAV: Objeto-Atributo-Valor; Este patrón genera a partir de una transacción otras dos transacciones que permiten extender la original, con el objetivo de permitir definir atributos en runtime.

Utilización de la herramienta (ejemplo aplicando patrón WorkWith)

Paso 1

- Contamos con una KB con algunas transacciones (se recomienda tener prototipo web creado)
- Ejecutamos la herramienta Patterns
- Desde Genexus (**Tools / Patterns**, en el modelo de Diseño) la KB se cerrará automáticamente, y se reabrirá cuando se cierre la herramienta Patterns
- En forma independiente (**GeneXusPatterns.exe**) será necesario cerrar la KB desde Genexus, ya que para utilizar una KB desde la herramienta Patterns debe haberse cerrado previamente desde Genexus y viceversa.

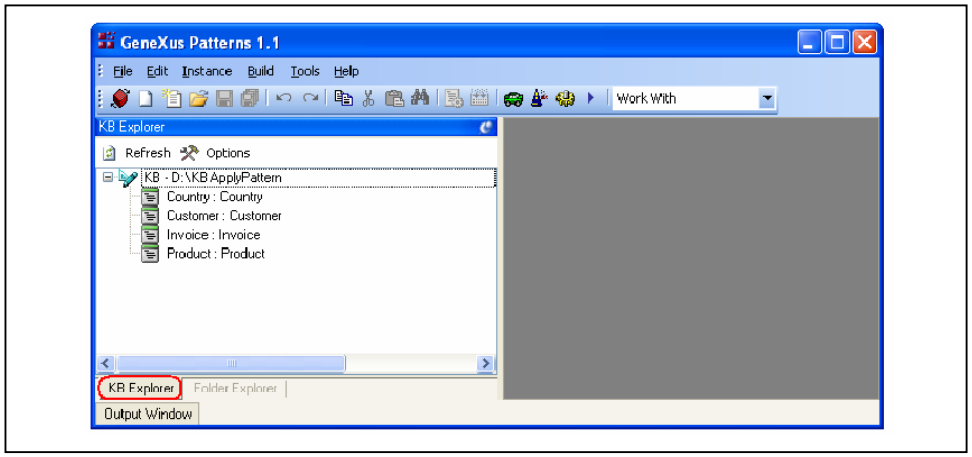
Si se ejecuta la herramienta Patterns en forma independiente de Genexus, será necesario seleccionar la KB con la cual se trabajará (para ello la herramienta Patterns ofrece el ítem: **File / Open Knowledge Base**).

Cuando se trabaja con la herramienta Patterns con una KB por primera vez, se presenta un diálogo cuyo título es "Workspace Configuration". Este diálogo permite configurar algunas opciones relacionadas a la KB, otras opciones relacionadas al pattern a ser aplicado, y otras opciones relacionadas a operaciones de Genexus (impactar, especificar, etc.) que pueden ejecutarse utilizando la herramienta Patterns.

En este punto inicial puede cerrarse estediálogo si se desea, y posteriormente es posible ingresar al mismo, mediante el ítem **Build / Configure GX Integration** (ver paso 8).

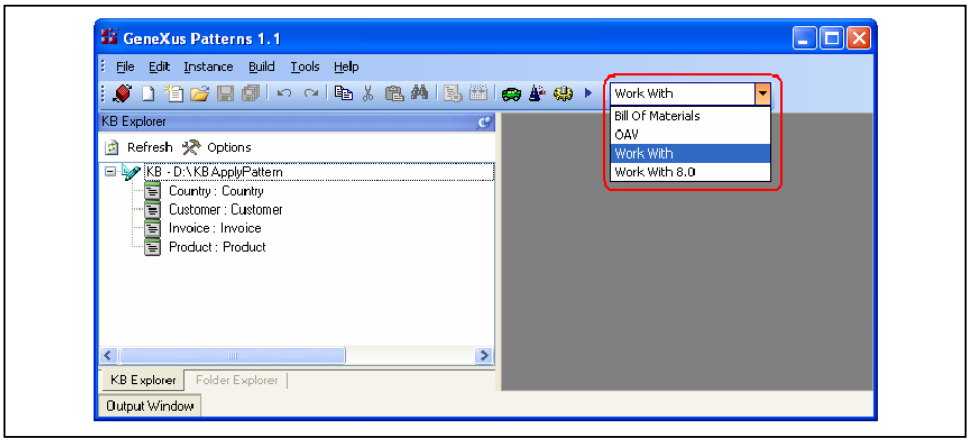
Paso 2

- Se despliega información de la KB
- Dado que muchos patterns usan TRNs, se muestra una lista de las TRNs disponibles en el tab **KB Explorer**:



Paso 3

- Seleccionar en el combo box mostrado en la toolbar, el pattern que se desea aplicar:



Se ofrece por defecto la lista de patrones predefinidos.

Paso 4

- Debemos obtener un 'instance file' por cada caso al cual queramos aplicar el pattern.

- 'Instance file' = archivo XML con los datos propios de la instancia.

- Si bien es posible crear cada 'instance file' de cero, los patterns suelen proveer una funcionalidad para crear 'instance files' por defecto, y luego poder modificarlos fácilmente.

- En el caso del pattern WorkWith, en el tab KB Explorer:

- Teniendo una TRN(o varias) seleccionada(s) botón derecho / Generate Instance File

- Doble clic en una TRN

Llamamos proceso de instanciación de un patrón al proceso de aplicar un patrón a una o varias situaciones (instancias).

En el proceso de instanciación de un patrón las entradas son:

• **Instance files:** por cada situación o instancia a la cual se quiera aplicar el patrón, habrá que crear un instance file con la información propia de esa instancia en particular (atributos a mostrar, etc.).

Cada instance file es en definitiva un archivo XML con los datos propios de la instancia, y tal como se explica en el grafico, los patterns suelen proveer una

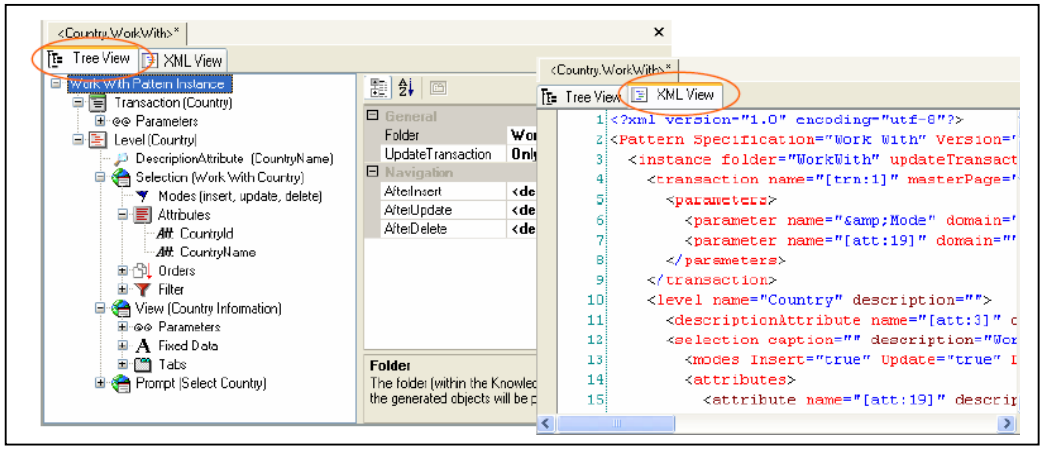
funcionalidad para crear ‘instance files’ por defecto (que luego, de considerarse necesario, se pueden modificar fácilmente).

• **Template files:** contienen la implementación del patrón y de su aplicación a las instancias.

El resultado que se obtiene del proceso de instanciación de un patrón (procesando los **instance files** y **template files**) es: un conjunto de objetos Genexus para ser consolidados en la KB.

Paso 5

- Los ‘instance files’ pueden editarse en el panel derecho
- Dos opciones para hacerlo: Tree View / XML View:



Cada ‘instance file’ es un archivo XML con estructura jerárquica, conteniendo cada uno de sus nodos un conjunto de propiedades.

La herramienta patterns ofrece 2 editores para editar cada ‘instance file’ en el panel derecho: el editor **XML View** y el editor **Tree View**.

El editor **XML View** permite editar los instance file directamente en su formato XML. Por su parte el editor **Tree View** es mucho más amigable, sencillo de usar, y con interfaz en alto nivel que provee mayor funcionalidad para ayudar en el proceso de edición.

Por todo esto el editor **Tree View** es el más usado y es el recomendado para usuarios no avanzados.

Paso 6

- Los 'instance files' se deben grabar. Para ello, bajo el ítem **Instance** se ofrecen las opciones **Save** y **Save All**.

- Los 'instance files' que no se han salvado aún se visualizan con nombre: <TRN Name.Pattern>*.

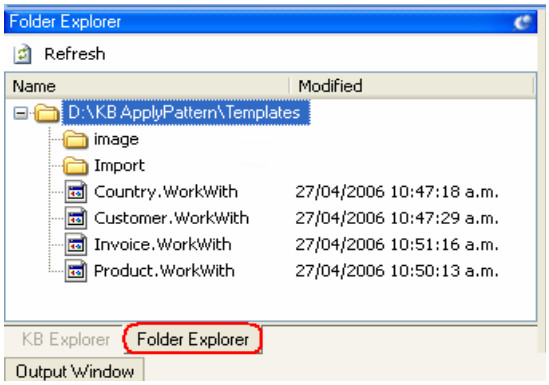
- Una vez salvados se visualizan con el nombre:

TRN Name.Pattern (ej: Country.WorkWith).

Save – salva el 'instance file' con el que se esté trabajando.

¿Dónde se almacenan físicamente los 'instance files'? En el subdirectorio Templates bajo el directorio de la KB.

Seleccionando el tab **Folder Explorer** se pueden visualizar estos archivos:



Save All – Salva todos los 'instance files' (si ya existen, pregunta si reemplazar).

Es importante tener en cuenta que si se generan los 'instance files' por defecto nuevamente a partir de las transacciones, serán sobrescritos.

Paso 7

- Una vez creados y editados los 'instance files', el siguiente paso es que la herramienta genere los objetos Genexus que implementan el pattern para las instancias.
- Opciones:
- Build / Apply Pattern
- Build / Apply and Consolidate
- Posibilidad de que se consoliden automáticamente a continuación o que lo haga después (desde la KB) el desarrollador Genexus.

Mediante botón derecho también es posible ejecutar estas acciones: es decir, estando posicionado en el tab Folder Explorer, luego de haber seleccionado los instance files (.workwith files), se pueden ejecutar las opciones Apply Pattern o Apply and Consolidate.

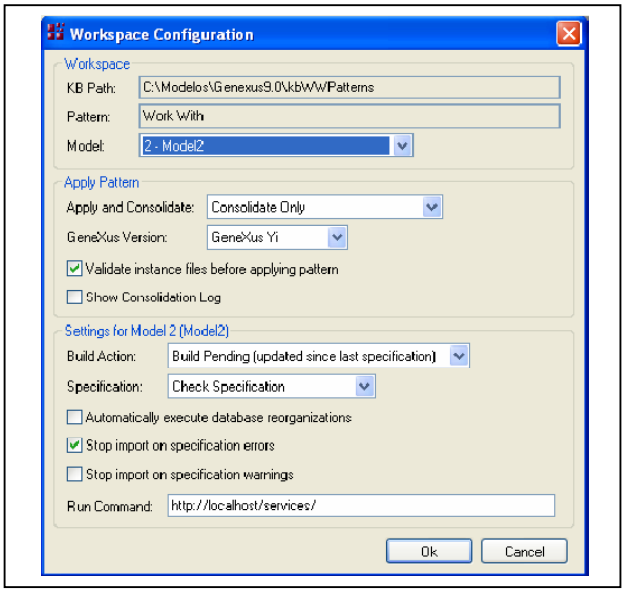
También es posible seleccionar una TRN o grupo de TRNs estando posicionado en el tab KB Explorer, y mediante botón derecho ejecutar la opción Generate, Apply and Consolidate. Pero es importante entender que seleccionando esta opción, se generarán los instance files por defecto nuevamente para las TRNs seleccionadas, y luego de ello, se generarán los archivos <TRNName>.xpz.xml correspondientes y se consolidarán en la KB.

Si se está seguro que los instance files por defecto no necesitan ser editados, es posible seleccionar esta opción directamente.
A su vez es importante saber que si se selecciona la opción Apply and Consolidate se efectuarán también las siguientes acciones:

- Se configurará como default theme de la aplicación al theme Fantastic.
- El directorio Images será copiado automáticamente bajo el directorio KBPath\Templates.
- La model property "Base image path" del modelo de diseño se configurará con el valor anterior.

Paso 8

- Desde la herramienta **Patterns**, es posible ejecutar las acciones que se realizan también desde Genexus: Impactar el modelo, Especificar, Generar, Compilar y Ejecutar.
- Diálogo donde se configura:



Para hacer más fácil todo el proceso, desde la herramienta Patterns, es posible ejecutar las acciones que se realizan con Genexus: Impactar el modelo, Especificar, Generar, Compilar y Ejecutar.

Estas acciones se configuran en el diálogo “Workspace Configuration” que se abre en el momento de abrir la KB con la herramienta Patterns, o seleccionando luego la opción Build / Configure GX Integration.

Opciones disponibles en el diálogo

Model – Muestra los modelos definidos en la KB, y permite seleccionar uno de ellos (se requiere tener los modelos creados y la creación de sus tablas hechas).

Apply and Consolidate – Al seleccionar la opción Apply and Consolidate, es posible ejecutar más acciones además de la generación de objetos y consolidación. Este combo ofrece las siguientes posibilidades:

• Consolidate Only (Apply and Consolidate)
• Impact Model (Apply and Consolidate + Impact Model)
• Impact, Specify (Apply and Consolidate + Impact + Specify)
• Impact, Specify, Compile (Apply and Consolidate + Impact + Specify + Compile)
• Impact, Specify, Compile, Run (Apply and Consolidate + Impact + Specify + Compile + Run).

GeneXus Version - La versión de GeneXus correspondiente a la KB se detecta automáticamente. El modelo se especificará / generará con dicha versión.

Build Actions- Permite seleccionar qué objetos deben especificarse y generarse al seleccionar Specify and Generate. Las opciones disponibles son:
- Build All
- Build Pending (updated since last specification)
− Specify Consolidated Objects

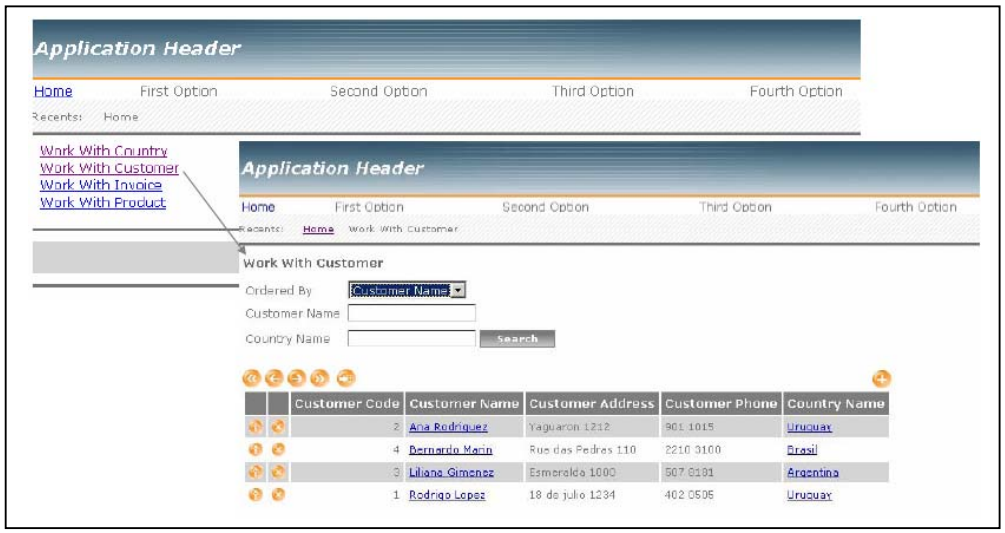
Specification – Permite seleccionar el tipo de especificación / generación a ser ejecutado al seleccionar

Specify and Generate. Las opciones disponibles son:
− Full Specification
− Check Specification
− Force Generation

Run Command – En esta opción se debe indicar la URL que se ejecutará si se seleccionó la opción
Impact, Specify, Compile, Run.
Se debe configurar:
Para aplicaciones .NET Run Command = http://localhost/services/**hhome.aspx**
Para aplicaciones Java o http://localhost:8080/servlet/**hhome**

Nota: Vale aclarar que “home” es el nombre de un web panel generado por el patrón WorkWith; el mismo ofrece links a todos los web panels WorkWith generados (y la letra h que antecede a su nombre en la invocación, corresponde al prefijo que se agrega a todo objeto web panel main o ejecutable que se invoca).

Resultado en ejecución



¿Cómo modificar los objetos generados?

• Mediante la herramienta Patterns

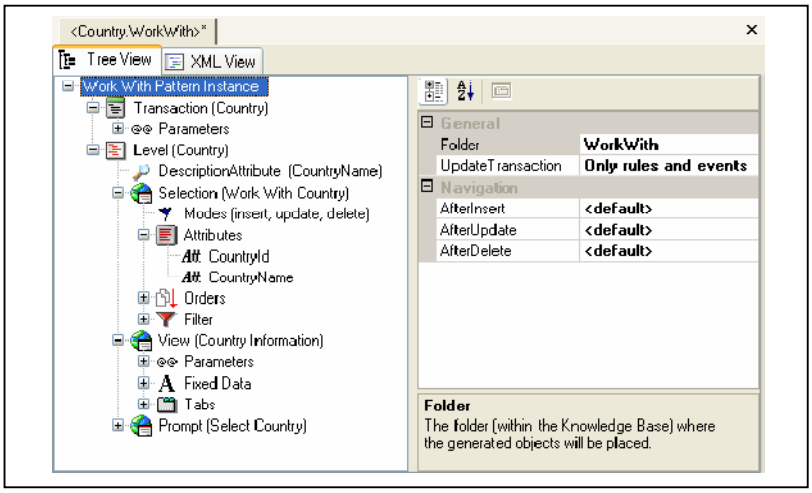
1. Modificando las propiedades de las instancias (en la medida que ofrezcan lo que se desea)

2. Modificando el patrón, personalizándolo para que ofrezca configurar propiedades que implementen las necesidades

• Mediante GeneXus

Modificando los objetos GeneXus generados (Desventaja: en caso de querer volver a generarlos con Patterns, se regeneran los objetos, perdiendo los cambios hechos a los mismos)

¿Cómo modificar las propiedades de las instancias?



Son muchas las propiedades que se ofrecen en los archivos de instancia correspondientes al patrón WorkWith para personalizar el comportamiento de los objetos que se generarán. A continuación describimos algunas de ellas.

El nodo **Selection** ofrece las propiedades relacionadas al web panel WorkWith que se generará para la instancia. Sus sub-nodos son:

Modes Este nodo permite definir en cuáles modos se ofrecerá invocar a la transacción. Las posibilidades y sus valores por defecto son:
Insert: True
Update: True
Delete: True
Display: False

Para casa modo podrá especificarse una condición. Se proveen las siguientes propiedades para ese propósito:
Insert Condition
Update Condition
Delete Condition
Display Condition

Si se define una condición asociada a un modo, la invocación para ese modo solo se habilitará si la evaluación de la condición es verdadera (Ejemplo: CountryId=10).

Attributes Este nodo permite definir cuáles atributos se desean mostrar en el grid (y para cada atributo, se pueden personalizar varias propiedades).

Orders Es posible ofrecer al usuario final varios órdenes posibles para ver el resultado de la consulta (es decir, las líneas mostrando los datos en el grid). Utilizando el botón derecho del mouse se puede definir un nuevo orden (su nombre y composición). Cada orden puede estar compuesto por varios atributos (pudiendo indicar para cada uno de ellos si se desea orden ascendente o descendente).

Se presentará un combobox en el web panel WorkWith ofreciendo todos los órdenes posibles de seleccionar, para que el usuario final elija uno y los datos se presenten en el grid ordenados por el mismo.

Filters Permiten definir condiciones de filtro para que se muestren en el grid solo los registros que cumplan con las mismas.

El nodo **View** por su parte, ofrece las propiedades relacionadas al web panel View que se generará para la instancia. El web panel View muestra toda la información de un registro, que fue seleccionado en el grid del web panel WorkWith (la información del registro es mostrada en una solapa de un tab control, y además hay una solapa con un grid por cada tabla directamente subordinada, para mostrar la información relacionada).

Observaciones

Al crear cada ‘instance file’ por default, podemos observar que las distintas propiedades del ‘instance file’ se inicializan con valores por default. Dichos valores por default para las propiedades, se especifican en un archivo denominado NombrePattern.config (en nuestro caso WorkWith.config).

El archivo NombrePattern.config se encuentra donde está instalada la herramienta Patterns, bajo el subdirectorio del Pattern particular que se esté utilizando. Para el

caso del patrón WorkWith estará bajo: Patterns\WorkWith (recordar que bajo el directorio de la herramienta Patterns, existe un subdirectorio por cada patrón predefinido (WorkWith, Bill of Materials, OAV) así como deberá haber un subdirectorio por cada patrón definido por el usuario.

Si deseamos tener un archivo de configuración NombrePattern.config por cada KB, debemos copiar este archivo al directorio Templates que se crea bajo la KB al usar la herramienta Patterns; así la herramienta Patterns utilizará dicho archivo ubicado en la KB para inicializar las propiedades de los 'instance files' que se creen.

Si la herramienta Patterns no encuentra el directorio Templates con este archivo bajo la KB, utilizará el archivo NombrePattern.Config (en nuestro ejemplo WorkWith.config) ubicado en el directorio Patterns\WorkWith.

El WorkWith.Config permite configurar algunos aspectos generales que aplicarán a todos los objetos. Por ejemplo, las master pages a ser utilizadas por los objetos web, los web components utilizados como header y footer, etc.