PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

DESARROLLO DE UN PATRÓN GENEXUS PARA LA CONSTRUCCIÓN DE APLICACIONES SOA

ALEJANDRO EMILIO ARAUS HUERTA

TESIS DE GRADO

MAGÍSTER EN INGENIERÍA INFORMÁTICA

Diciembre de 2013

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO/

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

DESARROLLO DE UN PATRÓN GENEXUS PARA LA CONSTRUCCIÓN DE APLICACIONES SOA

ALEJANDRO EMILIO ARAUS HUERTA

Director de tesis: José Miguel Rubio León

Programa: Magíster en Ingeniería Informática

Diciembre de 2013

Con todo cariño a quienes hicieron posible este trabajo, mi familia, pareja y maestros.

Resumen

Genexus es una herramienta de desarrollo de software, la cual permite generar aplicaciones para diferentes plataformas, lenguajes y opera sobre diferentes motores de bases de datos. Además, genera código automáticamente, una de las últimas funcionalidades de Genexus es el uso de patrones, funcionalidad con la cual se crean objetos Genexus de forma declarativa, logrando con ellos construir las opciones completas de un sistema.

Este trabajo está enfocado en crear un patrón Genexus, que permita desarrollar servicios web, para su uso en una arquitectura orientada a servicios (SOA), esto teniendo como objetivo el rápido desarrollo de aplicaciones, la reutilización y el bajo acoplamiento de los componentes, todos ellos beneficios de SOA y Genexus.

Palabras claves

Mejoras del proceso de desarrollo de software – Componente de software – Genexus – Arquitectura orientada a servicios - patrones.

Abstract

Genexus is a software development tool, that allows generate applications for different platforms, languages and works over multiples database management systems and additionally generating code automatically. One of the last functionalities of Genexus is the use of patterns, functionality which allow creating Genexus objects in a declarative way, achieving with this building the entire options of a system.

This work is focused in create a Genexus pattern, that allows develop web services, for their use in a service oriented architecture (SOA), this having as a goal the quick development of application, the reutilization and a low linkage of the component, all benefits of SOA and Genexus.

Keywords

Genexus – service oriented architecture - Component– Patterns - Software development Improvement.

Contenido

Índice de figuras

Índice de tablas

Capítulo 1 - Introducción

1.1 Introducción

La independencia de la plataforma es una meta que la ingeniería de software ha buscado constantemente desde hace mucho tiempo, el nivel de abstracción de la plataforma no había podido llegar a este nivel, especialmente porque no existía un "puente" que uniera las distintas plataformas y lenguajes de desarrollo, con la llegada de los servicios web esa brecha de abstracción fue superada y es en base a ellos que nace la arquitectura orientada a servicios, una arquitectura cuyos principales beneficios son el bajo acoplamiento, su alto grado de interoperabilidad y la reutilización de componentes.

En el presente trabajo se estudiarán los conceptos relacionados con las arquitecturas de software, específicamente la arquitectura orientada a servicios (SOA) y su aplicación en el desarrollo de aplicaciones con la herramienta de ingeniería de software asistida (CASE) Genexus.

La Arquitectura orientada a servicios es una arquitectura de software para la construcción de aplicaciones de negocios, compuesta por un conjunto de servicios débilmente acoplados. Los componentes en esta arquitectura son llamados "Servicios", un servicio representa una función de negocios bien definida, reutilizable y fácil de mantener, los servicios pueden ser invocados remotamente mediante protocolos de comunicación estándar.

Genexus es una herramienta de desarrollo de software multiplataforma, desarrollada por Artech, cuyo objetivo es asistir al analista y a los usuarios en todo el ciclo de vida de las aplicaciones. La idea básica de Genexus es automatizar funcionalidades como la normalización de los datos y diseño, generación y mantenimiento de la base de datos y de los programas de aplicación.

Dentro de las funcionalidades de las últimas versiones de Genexus están los patrones, los cuales son una funcionalidad que permite la creación de objetos Genexus de forma declarativa.

En este trabajo se hará uso combinado de la arquitectura orientada a servicios y la aplicación de patrones Genexus para desarrollar componentes uniformes, multiplataforma, de forma declarativa, En este trabajo se crearán componentes que podrán ser invocados desde distintas aplicaciones y además, podrán ser construidas en distintos lenguajes, evitando con esto tener que desarrollarlos de nuevo cada vez que se necesiten, esto ahorrará esfuerzos ya sea para la empresa que desarrolla el software, como por la empresa que los dispondrá para que las empresas proveedoras de software las usen.

La estructura de este documento es la siguiente: primero se abordarán temas como la problemática actual de

las empresas respecto al desarrollo de software y la cantidad de sistemas y luego algunos problemas del desarrollo de software con Genexus, para luego continuar con la solución que se plantea en este estudio para solucionar estos problemas, posteriormente en el estado del arte, se analizará la situación actual, la información relevante de este estudio y las soluciones similares a la que se plantea en el presente trabajo, para luego abordar la base de este estudio que consiste en la creación del patrón Genexus para la creación de servicios, para luego analizar sus resultados en base a otras soluciones similares.

1.2 Motivaciones y problemas

Las aplicaciones desarrolladas con Genexus por lo general si bien son desarrolladas rápidamente dado que genera ciertas funcionalidades automáticamente, estas no presentan un buen grado de reutilización, esto se debe a que el concepto de componentes no está presente en la forma que Artech indica que se debe desarrollar con Genexus, siguiendo las instrucciones del desarrollo sugerido por Artech, la lógica queda toda asociada en un sólo objeto y muy ligada a los datos.

Somerville dice lo siguiente sobre las herramientas case en su libro de ingeniería de software [11]:

"Conjuntos de herramientas CASE no soportan el desarrollo con reutilización. Puede ser difícil o imposible integrar estas herramientas con un sistema de librería de componentes. El proceso de software asumido por estas herramientas puede no tener en cuenta la reutilización."

En el caso de esta herramienta esta cita se cumple cabalmente, la propuesta de Genexus para crear aplicaciones rápidamente es la generación automática de código fuente y no se considera la reutilización de componentes o código como algo clave, siendo que para algunos autores es un aspecto fundamental.

La importancia de la reutilización radica en la mejora de los tiempos de construcción y el aumento en la calidad del producto, esto se ejemplifica en la siguiente cita de Halafb Mili et al[13]:

"La reutilización es la única aproximación realista para llegar a los índices de productividad y calidad que la industria del software necesita."

Desde el punto de vista de las empresas consumidoras de software, existe una problemática común, que es el crecimiento desmedido de sistemas, muchos sistemas creados a lo largo del tiempo con funcionalidades repetidas, sistemas inaccesibles y otros con una lógica no conocida por el personal informático, en el libro de adopción de SOA [4], se aborda este tema y clasifican los problemas en tres tipos:

- **Antiguos sistemas de información**: Sistemas antiguos construidos con tecnología de la época, en los cuales dado la arquitectura de aquellos tiempos, las capas no se encuentran claramente definidas, además dado el

tiempo que tienen, puede no existir personal que tenga conocimiento de todos estos sistemas, es por lo mismo que el costo de mantenimiento puede ser muy elevado.

- **Sistemas redundantes e inaccesibles**: También denominados islas de información, estos sistemas no fueron diseñados para intercambiar información con otros sistemas. En esta clase de aplicaciones aparece con frecuencia funciones redundantes. Ejemplo una empresa tiene más de una tabla de clientes dada diferentes áreas. El desarrollo de opciones redundantes por parte de la empresa es un derroche de tiempo, dinero y esfuerzo.

- **Laberinto de integraciones punto a punto**: Aplicaciones y funcionalidades que se comunican una a una, estas aplicaciones pueden crear problemas severos, ya que ante la caída de uno es probable que otros sistemas caigan.

1.3 Solución propuesta

En la herramienta Genexus existen la funcionalidad llamada patrones, los cuales dado un objeto con la estructura de una tabla, generan código y objetos automáticamente, la idea de este trabajo es reforzar esa cualidad, complementándola con un enfoque más tradicional que permita la creación automática de componentes reutilizables, con esto se espera que la velocidad de desarrollo de aplicaciones aumente, ya que los componentes (servicios web) podrán ser generados automáticamente (de forma declarativa) para múltiples plataformas y con un bajo grado de acoplamiento.

Específicamente este estudio plantea el desarrollo de un componente de software que promueva la reutilización y creación de servicios de manera automática (extender la funcionalidad de Genexus), dichos componentes serán creados con la intensión de que operen en una arquitectura orientada a servicios, este enfoque ayuda al desarrollo de software de manera rápida, basada en elementos previamente construidos.

La arquitectura orientada a servicios ayuda a que los problemas relacionados con el crecimiento de los sistemas desmedidamente no ocurra, debido que su enfoque está orientado a la comunicación, abordando este aspecto desde un principio y no como un ente que se deja para el final, con esto se logra que las aplicaciones no queden aisladas y puedan ser reutilizadas constantemente [4].

Los patrones Genexus son una propiedad que se puede implementar en los objetos transacciones de la base de conocimientos (código fuente Genexus), tiene como objetivo crear de forma automática todos los objetos necesarios para proveer funcionalidades tales como buscar, ingresar, modificar, visualizar y eliminar registros de una base de datos. Dentro de la instalación inicial de Genexus se encuentra el patrón workwith el cual se aplica como una propiedad, además existen otros patrones creados por empresas externas que pueden ser instalados, estos patrones de otras empresas son extensiones al ambiente integrado de desarrollo (IDE).

La empresa que desarrolla la herramienta (Artech) asegura que más del 80% de los objetos requeridos por un sistema se puede hacer mediante la aplicación de patrones Genexus, en este estudio se creará un nuevo patrón que no sólo creará los objetos necesarios para desarrollar aplicaciones de una forma declarativa, sino que los objetos que serán creados serán multi-plataforma en su concepción y al ser servicios web, podrán ser consumidos por aplicaciones desarrolladas en distintos lenguajes, incluso para dispositivos móviles.

Con la aplicación del patrón de este estudio, se pretende mejorar la calidad y velocidad de desarrollo de las aplicaciones construidas con Genexus, ya que al aplicar el patrón la mayoría de los objetos serán creados sin necesidad de programación. Los servicios creados automáticamente servirán para administrar la información de las tablas del sistema, realizar búsquedas sobre la información y aplicar lógica de negocio, lo cual es evidentemente mucho más rápido que el desarrollo "a mano" por parte de los programadores. En cuanto a la calidad de los servicios creados, también se encuentran beneficios, ya que al ser generados los todos servicios desde una misma fuente, tendrán una consistencia entre la interfaz y código, a estos beneficios se le deben sumar los beneficios de la arquitectura SOA y la reutilización de componentes.

Dentro de las riesgos de aplicar patrones Genexus se encuentra que el uso de un patrón mal construido o especificado, creará la aplicación completa con errores, ya que el error se replicará en todos los objetos generados por el patrón, de cualquier manera, al modificar el patrón, los objetos que fueron creados por él, también cambian, pudiendo rápidamente enmendar errores relacionados por el uso de un patrón mal confeccionado.

Una de las principales motivaciones del desarrollo de este estudio, es tratar de facilitar la tarea a los desarrolladores de software, muchas veces deben construir una funcionalidad una y otra vez, además tampoco tendrán que desarrollar funcionalidades que ya existen dentro de la empresa cliente, bastará que realicen las interfaces que consumen los servicios existentes, esto permitirá tener un mayor control sobre quien actualiza la información de la empresa y la forma en que se almacena la misma.

Con el uso de funcionalidades ya construidas y con la generación de servicios automáticamente, se pretende que los desarrolladores pasen más tiempo preocupados de labores relacionadas con los requerimientos del usuario y menos que con labores propias de la programación, esta descripción es parte de la filosofía Genexus.

1.4 Análisis de Objetivos

1.4.1 Objetivo general

Crear un componente de software que permita crear servicios de forma automática y declarativa para el desarrollo de aplicaciones bajo la arquitectura orientada a servicios.

1.4.2 Objetivos específicos

Comprender y explicar los conceptos asociados a la construcción de patrones Genexus.

Comprender y explicar los conceptos relacionados con la arquitectura de software con especial énfasis en la arquitectura orientada a servicios.

Desarrollar el componente propuesto en este estudio.

Evaluar la efectividad del componente mediante su aplicación a un caso real.

1.5 Organización del documento

El presente documento se dividirá en 5 capítulos y en 3 anexos de la siguiente manera:

Capítulo 1 Introducción: Es el capítulo inicial, en él se plantea a grandes rasgos el surgimiento de este trabajo, la problemática, los objetivos y como se estructuró el trabajo realizado.

Capítulo 2 Marco teórico: En este capítulo se estudian los conceptos más importantes relacionados con el proyecto y la base del funcionamiento de la solución propuesta, como así también los trabajos relacionados.

Capítulo 3 Desarrollo: En el presente capítulo se aplicaron los conceptos presentados en el capítulo 2, se explica la construcción y el funcionamiento a grandes rasgos de la solución..

Capítulo 4 Discusión de los resultados: Se realizan pruebas con la nueva solución y se evalúan los resultados en comparación al modo de desarrollo estándar de Genexus. La comparación se realiza en dos proyectos porque la solución desarrollada impulsa la reutilización, caso que no puede ser visto en un sólo proyecto, preliminarmente se evaluará la cantidad de errores versus la velocidad de desarrollo y la calidad de la solución.

Capítulo 5 Conclusiones y trabajo futuro: En este capítulo se destacan los puntos más relevantes del desarrollo de este proyecto, como así también la manera de ampliar este trabajo (trabajos futuros).

Bibliografía: En esta sección se muestran las referencias citadas en el presente trabajo.

Glosario: En esta sección se explican los conceptos usados comúnmente durante este trabajo.

Anexo 1: En esta sección se mostrará el detalle del trabajo realizado, es la explicación detallada de la solución descrita en el capítulo 3.

Anexo 2: Estructura y explicación lenguaje que se emplea para el desarrollo de patrones Genexus.

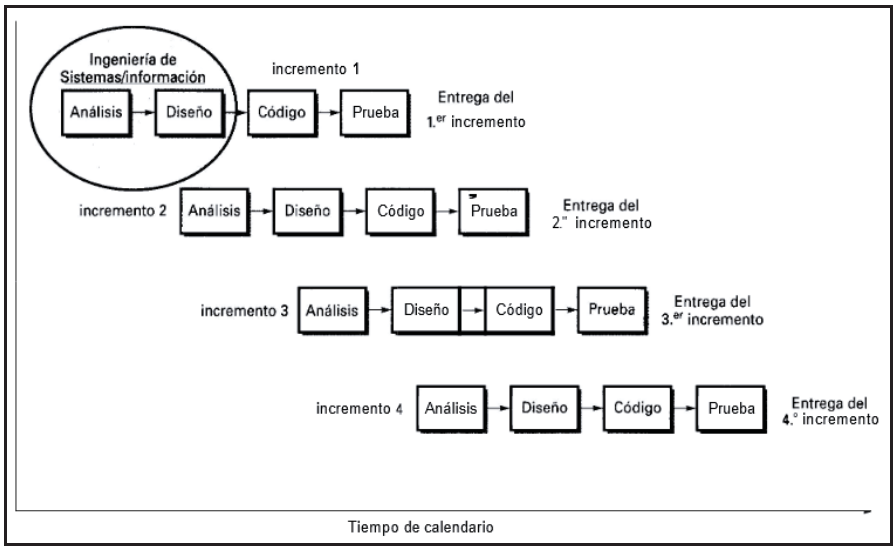
Anexo 3: Ejemplo de patrón integrado en el IDE.

1.6 Metodología de trabajo

Para el desarrollo de este proyecto, seguiremos una metodología de desarrollo de software incremental. El modelo incremental combina elementos del modelo lineal secuencial (cascada) con la filosofía interactiva de construcción de prototipos. El modelo incremental aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un «incremento» del software.

En un proceso de desarrollo incremental, los clientes identifican, a grandes rasgos, los servicios que proporcionará el sistema. Identifican qué servicios son más importantes, cuáles menos. Entonces, se definen varios incrementos en donde cada uno proporciona un subconjunto de la funcionalidad del sistema.

Una vez que un incremento se completa y entrega, los clientes pueden ponerlo en servicio. Esto significa que tienen una entrega temprana de parte de la funcionalidad del sistema. Pueden experimentar con el sistema, lo cual les ayuda a clarificar sus requerimientos para los incrementos posteriores y para las últimas versiones del incremento actual.



Este proceso de desarrollo incremental tiene varias ventajas:

1. Los clientes no tienen que esperar hasta que el sistema completo se entregue para sacar provecho de él. El primer incremento satisface los requerimientos más críticos de tal forma que pueden utilizar el software inmediatamente.

2. Los clientes pueden utilizar los incrementos iniciales como prototipos y obtener experiencia sobre los requerimientos de los incrementos posteriores del sistema.
3. Existe un bajo riesgo de un fallo total del proyecto. Aunque se pueden encontrar problemas en algunos incrementos, lo normal es que el sistema se entregue de forma satisfactoria al cliente.
4. Puesto que los servicios de más alta prioridad se entregan primero, y los incrementos posteriores se integran en ellos, es inevitable que los servicios más importantes del sistema sean a los que se les hagan más pruebas. Esto significa que es menos probable que los clientes encuentren fallos de funcionamiento del software en las partes más importantes del sistema.

En las primeras etapas del proyecto tendrán mayor relevancia las labores de análisis y diseño, y en la medida que se avance el proyecto y se defina una base de lo que se desea construir, las labores de construcción y prueba tendrán destinado un mayor tiempo.

1.6.1 Iteraciones

Para el presente trabajo se definieron 4 iteraciones:

1° Iteración: Definición del patrón a desarrollar, definición de las iteraciones a realizar, estudio bibliográfico y herramienta seleccionada, revisión del estado del arte, revisión código fuente de algunos ejemplos para evaluar factibilidad.

- **Salida**: Documento con las definiciones del patrón, documento con las iteraciones y sus tareas relacionadas.

2° Iteración: Definición del patrón a desarrollar, estudio bibliográfico, preparación ambiente para el desarrollo e inicio diseño de la solución

- **Salida**: Documento refinado con las definiciones del patrón, hitos relacionados con la documentación del proyecto y ambiente de desarrollo preparado para la construcción de los patrones y documento inicial con el diseño de la solución.

3° Iteración: Definición patrón a desarrollar, diseño de la solución, inicio construcción del patrón, primeras pruebas, elementos relacionados con la documentación del proyecto.

- **Salida**: Documento final con las definiciones del patrón, diseño de la solución en cuanto a componentes a construir, elementos relacionados con la documentación del proyecto y avances respecto a la construcción del patrón.

4° Iteración: Diseño de la solución, construcción del patrón, pruebas, conclusiones del trabajo.

- **Salida**: Patrón Genexus construido, documento con la evidencia de las pruebas, elementos relacionados con la documentación del proyecto.

1.6.2 Detalle de las tareas

Las tareas dentro de las iteraciones tienen los siguientes objetivos:

- Definición del patrón a desarrollar: Se establecen las directrices de lo que el patrón Genexus va a realizar, alcance, funciones.
- Estudio bibliográfico y herramienta seleccionada: Se estudian los conceptos relacionados con los patrones Genexus y Arquitecturas de software, componente y trabajos relacionados del área.
- Preparación ambiente de desarrollo: Se configura el pc de trabajo con las herramientas necesarias (Genexus, visual studio, visual webstudio) para continuar con la construcción de trabajo.
- Diseño de la solución: Actividad complementaria a la definición del patrón, con orientación más técnica, se establece cómo se va a construir lo que ha sido definido.
- Construcción del patrón: Programación del patrón en visual studio y pruebas en Genexus.
- Pruebas: Se realizan las pruebas en Genexus (al aplicarlo) y posteriormente se revisan los servicios web creados después de la aplicación.
- Conclusiones del trabajo: Se describen las tareas futuras y la experiencia sobre el desarrollo del trabajo.

Capítulo 2 – Marco teórico

En esta instancia definiremos los conceptos sobre los que se basa esta tesis, se centrará en definir la herramienta del estudio, la importancia de las arquitecturas de software y los conceptos de componentes de software, además se mostrará los trabajos relacionados con la tesis y como estas se complementan con el trabajo acá realizado.

2.1 Genexus

GeneXus es una herramienta inteligente para crear, desarrollar y mantener, en forma automática, aplicaciones multiplataforma de misión crítica que se adaptan fácilmente a los cambios del negocio y a las nuevas posibilidades brindadas por la evolución tecnológica [27].

Capturando los procesos y describiendo la realidad, permite crear modelos que perduran en el tiempo (bases de conocimiento), que utiliza para generar y mantener las aplicaciones. [27].

Genexus crea aplicaciones mayormente de forma declarativa, a partir de la declaración hecha por un analista, dada esta declaración Genexus genera código para las diferentes plataformas junto con ello Genexus infiere la base de datos normalizada. Todo el código es generado para diferentes plataformas destino a partir de una misma especificación básica (Lenguaje Genexus), pudiendo cambiar de proveedor de base de datos y de lenguaje de programación con un par de clics.

Genexus a partir de las visiones de los usuarios; captura su conocimiento y lo sistematiza en una base de conocimiento (Knowledge Base). Mediante procesos de inferencia, Genexus es capaz de diseñar, generar y mantener de manera automática la estructura de la base de datos y los programas de la aplicación, es decir, los programas necesarios para que los usuarios puedan operar con sus visiones [20].

La base de conocimiento de Genexus tiene una gran capacidad de inferencia lógica, es capaz de proporcionar cualquier conocimiento que se ha almacenado en ella o que puede inferirse lógicamente de aquellos almacenados en ella. Basado en esta capacidad de inferencia es capaz de proyectar, generar y mantener, en forma 100% automática, la base de datos y los programas necesarios para satisfacer todas las visiones de usuarios conocidas en un determinado momento [12].

2.1.1 Desarrollo de una aplicación con Genexus

Para desarrollar la aplicación, Genexus utiliza la información almacenada en la Base de Conocimiento, convierte este conocimiento en un metalenguaje propio para el cual realiza todas las verificaciones de integridad y consistencia; a este proceso se lo denomina especificación y genera un archivo interno con la representación correspondiente.

Posteriormente, tomando como entrada el resultado de la especificación, se genera código para el lenguaje configurado (java, .net, visual basic, entre otros); a este proceso se lo denomina generación. En caso de existir cambios a nivel de estructura de datos se generan los programas para reorganizar las estructuras físicas correspondientes. Finalmente, dependiendo de la plataforma seleccionada se ejecutará el proceso de compilación y empaquetado de la aplicación.

El Proceso de desarrollo Genexus, desde el modelado hasta la compilación de una aplicación se puede apreciar en la siguiente imagen:

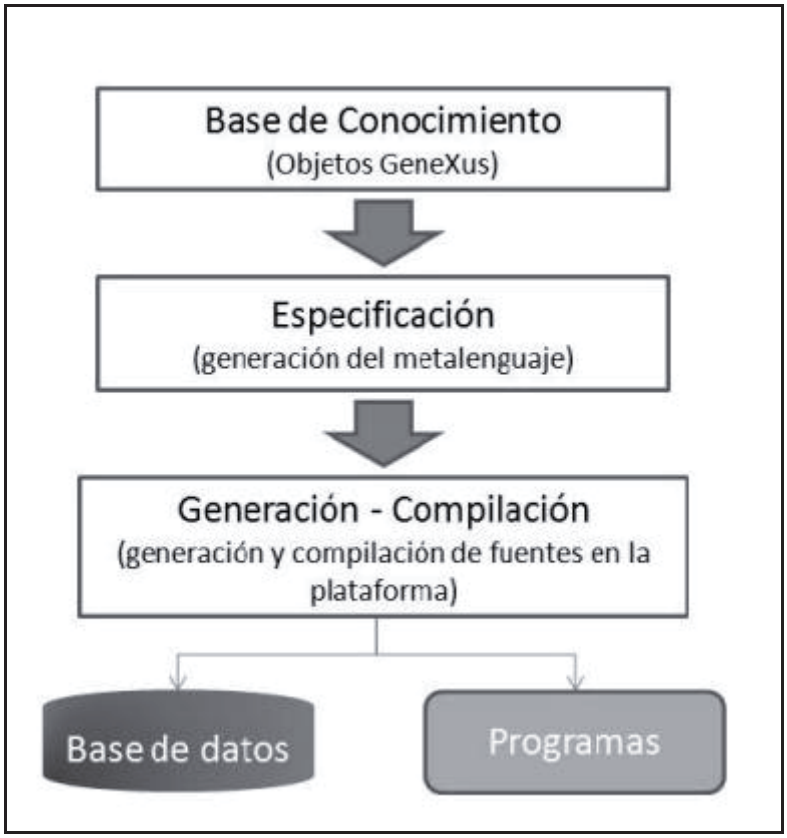


Fig. 1-Proceso de desarrollo Genexus [22]

El desarrollo en Genexus se trata de un enfoque en donde se independiza de la tecnología trabajando a un nivel de abstracción mayor que los lenguajes de propósito general, al que los desarrolladores de la herramienta denominan: desarrollo basado en conocimiento.

Dado un conjunto de objetos del usuario existe una única base de datos relacional mínima que lo satisface. Las bases de datos que diseña GeneXus están en tercera forma normal y tienen los índices estrictamente necesarios. Sin embargo, a veces, es muy adecuado (por razones de performance) introducir ciertas redundancias de datos y ciertos índices adicionales. GeneXus da al analista indicaciones de que redundancias o índices puede ser adecuado definir para optimizar las consultas.

Partiendo del conocimiento almacenado en la base de conocimiento, GeneXus genera automáticamente la aplicación (base de datos y programas) para la plataforma que se haya escogido. Desde un punto de vista lógico, lo que debe hacerse es independiente de la plataforma (configuración de la base de conocimiento). Físicamente, sin embargo, no lo es: cada lenguaje, cada sistema de administración de base de datos, cada sistema operativo, cada arquitectura, tiene comportamientos diferentes. GeneXus resuelve este problema dividiendo la generación en dos: lógica que es común a todas las plataformas, y física, que se construye plataforma a plataforma de modo de optimizar los programas generados para cada una de ellas.

El Entorno de Desarrollo de Genexus es a partir de la versión X un IDE genérico y extensible. Esto significa que es posible agregar módulos denominados “paquetes” con el fin de incorporar una funcionalidad específica. La propia versión X de Genexus está implementada como una colección de paquetes integrados a este IDE genérico.

2.1.2 Historia de Genexus

Artech consultores desarrolla la primera versión de Genexus el año 1988, el primer propósito de Genexus era establecer un modelo de datos robustos. Una vez construido dicho modelo, se realizaron ampliaciones del mismo incorporándole elementos declarativos como reglas, fórmulas, pantallas, etc., Esto con el fin de abarcar el conocimiento de los sistemas de negocios.

Para poder proyectar, generar y mantener en forma 100% automática los sistemas computacionales de una empresa cualquiera Genexus trató de abarcar todo el conocimiento de los sistemas de negocios. La primera versión de Genexus cien por ciento declarativa pretendía resolver el problema de generación y mantenimiento del 70% de los programas. De ellos se ocupaba el sistema en forma automática, el 30% restante debía ser programado y mantenido manualmente. Al año del lanzamiento del proyecto de Genexus, los clientes demandaban un mayor grado de generación automática, es debido a eso que la filosofía de Genexus cambia a conseguir un tratamiento automático del conocimiento de los sistemas de negocio.

Los aspectos en los que Genexus actualmente enfoca sus esfuerzos son los siguientes:

- Generación y mantenimiento automáticos de la base de datos y los programas de aplicación necesarios para la empresa.
- Generación a partir del conocimiento para múltiples plataformas.
- Integración del conocimiento de diversas fuentes para atender necesidades complejas con costos en tiempo y dinero inferiores a los habituales.
- Generación de aplicaciones en tecnologías futuras con la definición que existe hoy. Genexus debe trabajar con el mismo modelo existente hoy cuando vengan nuevas tecnologías.

Los hitos en la historia de Artech y Genexus en los últimos años son los siguientes:

Año	**Evento**
1989	· Se crea Artech en Uruguay
	· Genexus 1.0 se lanza al mercado con su primer generador: RPG y Cobol para AS/400
	· Se libera el generador FoxPro
1996	· Se libera el primer generador Cliente/Servidor
1999	· Se libera GXplorer, solución para Data Warehouse
2000	· Se libera el primer generador JAVA
2002	· Se libera el primer generador .NET
	· Se libera GXQuery, solución para Reporting
	· Se libera GXFlow, solución para flujos de trabajo (Workflow)
	· Se libera GXPortal, solución para crear y gestionar portales dinámicos
2005	· Se libera GX9
2008	· Se lanza Genexus X con importantes aumentos de productividad y desarrollo para web.
2009	· Se libera GXX EV1 enfocado en la experiencia del desarrollador. · Se libera GXserver, que simplifica el desarrollo en equipo. Se comienza a comercializar GXtest.
2010	· La comunidad de usuarios supera los 70 mil miembros.
2012	· Se libera GX X EV2

Tabla 1 - Historia Genexus

2.1.3 Características de Genexus

Genexus soporta distintos lenguajes de programación, DBMS y plataformas, dentro de estas podemos nombrar los siguientes.

Lenguajes de programación:

- Cobol
- RPG
- Visual Basic
- Visual FoxPro
- Ruby
- C#
- Java

Plataformas:

- Aplicaciones windows,
- Aplicaciones Web,
- Aplicaciones iseries
- Plataformas móviles, incluyendo Android o Blackberry, y Objective-C para aparatos Apple.

Motores de base de datos:

- Microsoft SQL Server
- Oracle
- IBM DB2
- Informix
- PostgreSQL
- MySQL.

Objetos Genexus

Genexus representa la realidad en una base de conocimiento mediante objetos Genexus:

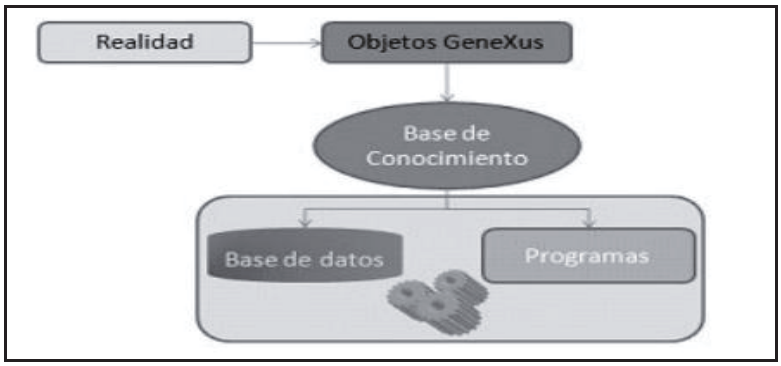


Fig. 2- Representación Genexus [22]

El desarrollador Genexus cuenta con distintos artefactos para crear una aplicación, los cuales se les denominan objetos Genexus, estos objetos tienen relación con el desarrollo de interfaces, almacenamiento en base de datos, estructuras de datos y reportes.

2.1.4 Objetos Genexus

Los principales objetos Genexus se detallan en la siguiente figura:

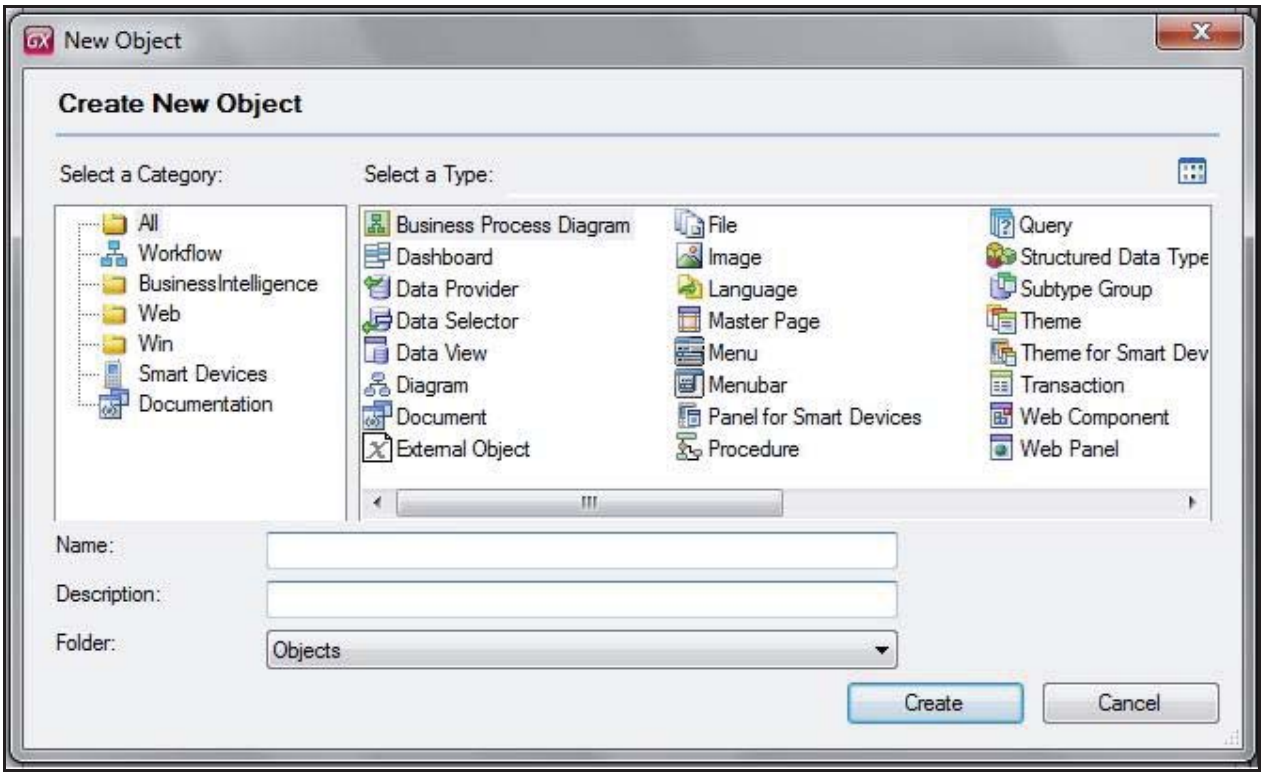


Fig. 3 – Objetos Genexus [IDE Genexus]

- **Transacciones**: Permiten definir objetos de la realidad que el usuario manipula (ej: clientes, productos, proveedores, facturas). Son los primeros objetos que se definen cuando se necesita tener persistencia de datos, ya que a través de las transacciones Genexus infiere el diseño de la base de datos.

  Además de generar una respectiva tabla o tablas, entrega la posibilidad de mediante algunos ajustes crear aplicaciones tanto web, win o iseries para las tareas principales en esas tablas (ingreso, actualización y eliminación).

- **Reportes**: Permiten recuperar información de la base de datos, y desplegarla ya sea en la pantalla, en un archivo o enviar directamente a la impresora. Son los típicos listados o informes. No permiten actualizar la información de la base de datos.

- **Procedimientos**: Tienen las mismas características que los reportes, pero a diferencia de éstos, permiten además la actualización de la información de la base de datos.

- **Work Panels**: Permiten al usuario realizar interactivamente consultas a la base de datos, a través de una interfaz, además son las encargadas del ingreso de datos para que luego puedan ser ingresadas en la base de

datos mediante un procedimiento. No permiten la actualización de la base de datos, sino solo la consulta y eventualmente llamar a los procedimientos para las operaciones de modificación de información.

- **Web Panels**: Son similares a los work panels, salvo que son usados a través de navegadores en ambiente Internet/Intranet/Extranet.

- **Data Views:** Permiten manejar archivos externos como si fueran archivos pertenecientes a la base de conocimiento. Útiles cuando se desea incluir dentro de la base de conocimiento tablas existentes y manejarlas internamente. También pueden ser archivos iseries.

- **Estructura de datos (SDT):** El objeto GeneXus Structured Data Type (SDT), permite definir estructuras de datos. Estas representan, de una forma simple, datos cuya estructura está compuesta por varios elementos.

  Los SDT tienen múltiples usos posibles:

  - Facilitan el pasaje de parámetros (en particular permite proveer/consumir información estructurada en el uso de webservices)

  - Simplifican la lectura y escritura automática de archivos de lenguaje de marcado extendible (XML) (con funciones de alto nivel),

  - Permite mejorar la legibilidad del código,

  - Permite el manejo de listas de largo variables de elementos.

- **Los objetos externos (External Objects):** Permiten la interoperabilidad de las aplicaciones Genexus con el mundo exterior a través del consumo de Web Services, acceso a base de datos remotas utilizando procedimientos almacenados y uso de bibliotecas de terceros, como por ejemplo uso de DLL en C#, archivos JAR para Java y gemas para Ruby.

Otro elemento útil de mencionar que no es un objeto son las formulas, estas se definen a nivel de las transacciones y permiten el cálculo de algún determinado campo, por ejemplo, supongamos que necesitemos obtener el sueldo líquido y en la base de datos tengamos el sueldo bruto y los descuentos, la fórmula para obtener el sueldo liquido quedaría de la siguiente manera:

SueldoLiquido = (SueldoBruto-Descuentos)

El campo SueldoLiquido debe ser definido en la transacción, este campo podrá ser usado como si existiera en la base de datos, pero físicamente no se encontrará.

2.1.5 Patrones Genexus

Artech incluye junto con Genexus X un kit de desarrollo de software (SDK por sus siglas en inglés) con el cual se pueden desarrollar extensiones a la herramienta. Las extensiones construidas para Genexus son desarrolladas en una mezcla de XML y C# por lo cual es necesario además contar con visual studio 2005 o 2008. Para su desarrollo.

El concepto original de patrón viene de Christopher Alexander el cual la define de la siguiente forma [24]:

"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno y luego describe la base de la solución a dicho problema, de tal manera que puedas usar esta solución un millón de veces, sin repetirla de la misma manera dos veces."

Después de esta definición algunos "Gurues" de la programación orientada a objetos adaptaron el concepto a lo siguiente:

"Un patrón de diseño sistemáticamente nombra, motiva y explica un diseño general que se ocupa de un problema recurrente de diseño en sistemas orientados a objetos. Él describe el problema, la solución, cuando aplicar la solución y sus consecuencias, además da concejos y ejemplos sobre la implementación. La solución es un acuerdo general de objetos y clases que resuelven el problema". La solución es personalizada e implementada para resolver un problema en un contexto particular" [14].

Estas ideas pueden ser usadas fueras de la programación orientada a objetos, adaptada a Genexus esto sería algo como lo siguiente:

Un patrón de diseño sistemáticamente nombra, motiva y explica un diseño general que se ocupa de un problema recurrente de diseño en aplicaciones Genexus. Él describe el problema, la solución, cuando aplicar la solución y sus consecuencias, además da concejos y ejemplos sobre la implementación. La solución es un acuerdo general de objetos Genexus que resuelven el problema". La solución es personalizada e implementada para resolver un problema en un contexto particular.

Los patrones Genexus son una propiedad/sistema externo que se pueden implementar en los objetos transacciones de la base de conocimientos, tiene como objetivo crear de forma automática todos los objetos necesarios para proveer funcionalidades tales como retornar información, ingresar, modificar, visualizar y eliminar registros. Dentro de la instalación inicial de Genexus se encuentra el patrón workwith el cual se aplica como una propiedad, además existen otros patrones creados por empresas externas que pueden ser instalados, estos patrones de otras empresas son extensiones al IDE.

La definición de Artech para un patrón es la siguiente: Los patrones son es una funcionalidad generadora de conocimiento a partir de conocimiento, también se le puede definir como una funcionalidad que permite automatizar

lo automatizado, esto tiene sus fundamentos en que el patrón agrega más funcionalidad a la transacción, el cual es un objeto que en su forma por defecto permite el ingreso, visualización y eliminación de información, con la aplicación del patrón se pueden hacer validaciones extras, mejorar el aspecto de las transacciones, creación de webpanels de búsquedas, crear programas para hacer testing, etc.

Los patrones son una propiedad muy potente que evita la creación de objetos de forma manual, con los tiempos y recursos que todo ello implica, se pueden crear patrones de acuerdo a la necesidad de cada empresa adaptándolos a las necesidades del negocio, esto se logra mediante la creación de patrones propios (extensiones). Los beneficios más palpables de usar patrones son la creación de interfaces uniformes y una lógica consistente entre los distintos objetos lo cual hace facilita la mantención de los sistemas, además de la velocidad de construcción de opciones.

El patrón Genexus se puede considerar como un ente procesador, el cual recibe como entrada una transacción, luego el patrón procesa la transacción para poder realizar las tareas básicas en ella y de salida entrega un conjunto de objetos Genexus adicionales, más la transacción modificada. El resultado de este procesamiento es por lo general un webpanel para realizar las búsquedas sobre la base de datos y la transacción modificada para las acciones de procesamiento de registros.

Los patrones Genexus justifican su uso en lo común que es tener en las aplicaciones formularios que sean muy similares entre sí, aunque no sean totalmente iguales, generalmente comparten muchas funcionalidades en común, como son tener variables para filtrar los datos en un listado, ordenaciones en el listado, menús, ventanas de mantenimiento; entonces en vez de estarlos desarrollando todas las veces, es más conveniente hacer uso del patrón y generar todo este conjunto de funcionalidades automáticamente.

Otro aspecto importante es que los objetos generados automáticamente pueden ser modificados como si fuera un objeto generado a mano, de igual manera se debe tener cuidado al modificar el patrón que lo generó ya que esta acción modificará todos los otros objetos generados por él.

Características de los patrones

- Personalización: Las instancias de un patrón pueden ser modificada de acuerdo a los requerimientos del negocio, se pueden agregar nuevos filtros, ordenamientos, ocultar o visualizar atributos en las transacciones o en los listados.
- Dinamismo entre la transacción y los patrones: Cuando una propiedad es cambiada en el patrón, ejemplo agregar un nuevo filtro u ordenamiento, todos los objetos que fueron generados por el patrón se actualizarán automáticamente, logrando con esto que no sea necesario aplicar el patrón a los objetos nuevamente.
- Configuración: Los patrones tienen un archivo de configuración global en el que se pueden cambiar algunas propiedades de los objetos.
- Adaptabilidad: Debido a que el funcionamiento de los patrones se basa en plantillas, se puede acceder al código de los objetos generados para modificar o agregar nuevas funcionalidades.

- Inclusión de otras KB: Los patrones permiten que en su contenido se puedan incluir bases de conocimientos completas que pasaran a formar parte de la KB en la que se está trabajando.

Los patrones deben ser construidos con el lenguaje C# y se generan a partir de las funcionalidades que proporciona Genexus mediante el SDK, Este kit entrega funcionalidades para recuperar la información de la transacción, además provee librerías que dotan los tipos de objetos, tipos de datos que se definen en Genexus.

Ventajas del uso de patrones Genexus:

Las ventajas más inmediatas del uso de patrones Genexus son las siguientes:

- Tiempo de programación corto, dado que se generan objetos y código automáticamente: el tiempo de desarrollo se debería reducir notoriamente si se desarrolla un patrón que implemente todas las funcionalidades necesarias para el funcionamiento deseado de aplicación.

- Unificación de estándares y formatos, al ser los objetos creados por los patrones, estos tendrán una lógica similar en la que variará principalmente la tabla sobre la que se trabajará.

Desventajas:

- El desarrollo de patrones requiere de un análisis de las librerías y los patrones Genexus existentes, gracias a estos ejemplos se puede desarrollar los patrones, lamentablemente la información relacionada con el desarrollo de patrones provista por Artech es escasa, por lo cual el interesado en estos temas se debe adentrar mucho en los trabajos existentes explorando los códigos fuentes más que la documentación existente.

Aspectos generales de los patrones Genexus

En el desarrollo de aplicaciones, por lo general existen algunas partes de la aplicación que son bastante similares, pero no exactamente lo mismo, con pequeñas variaciones. Por ejemplo, en una aplicación que implica clientes y productos, es natural tener un formulario para anotar los clientes, y otra forma de hacer lo mismo para los productos. A pesar de que los clientes y los productos son totalmente diferentes, estas dos formas tienen muchas cosas en común: un listado, un campo creado para filtrar los datos, pedidos, acciones.

Dentro de los patrones podemos encontrar los siguientes tipos:

- Pasivos: Ellos existen sólo en los libros, la única manera de utilizarlas es leer los libros y el uso de estas ideas mientras se codifica.

- A nivel de codificación: A pesar de que los patrones son generalmente referidos como "patrones de diseño", están principalmente destinados a facilitar la codificación

Desde GeneXus se trabaja en el nivel del conocimiento en lugar del nivel de codificación, se desea utilizar patrones como una forma de permitir la reutilización del conocimiento, no la reutilización de código. Las características de los patrones Genexus son las siguientes:

- Un patrón de Activo: no es sólo una guía, se puede "instanciar", lo que significa que el marco genera todos los objetos GeneXus necesarios para implementar una instancia de patrón.

- Reutilización del conocimiento: GeneXus ha tratado de fomentar la reutilización del conocimiento, pero con el uso de patrones que puede ofrecer un nuevo nivel de reutilización del conocimiento: el mismo conocimiento se puede reutilizar en muchas situaciones diferentes. Por ejemplo, un "patrón especializado" se pueden crear instancias de los productos, sino también a los clientes.

- Abierto: Cualquier persona puede generar su / sus propios patrones (o modificar uno ya existente), Artech provee un SDK para visual studio, además de los códigos fuentes de los patrones category y work with.

- Fácil de usar: Genexus ha realizado esfuerzos en reducir la complejidad: no es necesario ser un experto para utilizar un patrón.

Se espera que con el uso de los patrones un desarrollador GeneXus logre un impulso de su productividad y de la calidad de sus aplicaciones. Por ejemplo, un escenario de uso típico es cuando los desarrolladores necesitan migrar una base de conocimientos desde iseries o formularios de Windows a formularios Web. En este caso, el patrón “work with” se puede aplicar para generar la mayor parte de los objetos web necesarios, evitando con esto el desarrollo manual de múltiples opciones, los patrones tomarán el conocimiento almacenado en esas KB desarrolladas para otras plataformas y las trasformará gracias a la aplicación de patrones a la nueva plataforma (web).

Aplicación de un patrón

A continuación se entrega una pequeña guía de cómo se aplican en general los patrones, el patrón que se desarrolla en este trabajo se aplicará de la misma manera, la forma es la siguiente:

1. Abrir el objeto transacción al cual se le desea aplicar el patrón.

2. En el selector de patrones, elegir el patrón que se desea aplicar, a este punto se puede ver la instancia por defecto de la transacción.

3. Personalizar la instancia, haciendo las modificaciones que sean necesarias en las propiedades.

4. Marcar la opción “Apply this pattern on safe” (aplicar este patrón al guardar).

5. Guardar la transacción.

Los mensajes son mostrados en la ventana de “output”, si un error ocurre será mostrado en esta ventana también.

Se puede seleccionar un conjunto de transacciones para aplicarles el patrón, para esto se debe seleccionar las transacciones con control, shift y el botón del mouse.

Existen dos formas de aplicar el patrón de forma grupal al mismo tiempo:

1. Seleccionar todas las transacciones en la vista de carpetas (Folder view), clic derecho en ellas y luego en el menú contextual seleccionar “Apply pattern” y posteriormente seleccionar el patrón a aplicar.

2. Ir a View, workwith objects, seleccionar las transacciones filtrando, seleccionarlas con control, shift y mouse y luego seleccionar el patrón a aplicar.

Aplicaciones generales para todas las instancias pueden ser hechas en “PatternSettings”.

Opciones de los patrones Genexus

En Preferences/patterns están las propiedades para cada patrón. En los ajustes del patrón se puede configurar las propiedades generales.

Las propiedades del patrón determinan algunos aspectos generales que se aplican a todos los objetos, como por ejemplo cual es la “master page” que se utilizará en los objetos Web Panel, Acciones estándar para cada trabajo, con nombres generales para atributos, filtros de la pantalla de búsqueda generada, campos de la grilla del objeto de búsqueda, etc.

Se debe recordar que estas propiedades se definen por cada objeto por lo cual se puede tener distintos comportamientos con el mismo patrón.

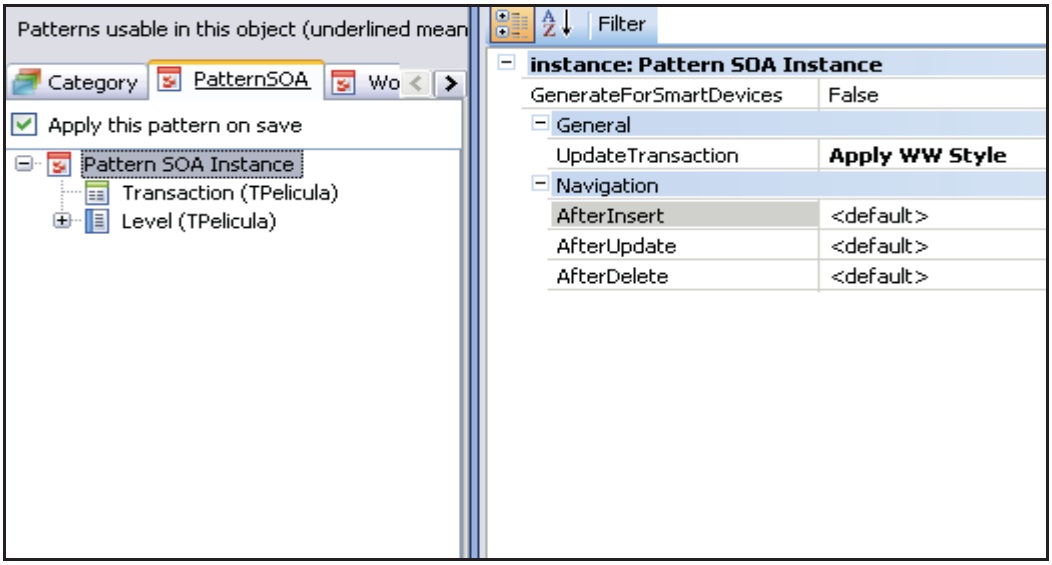


Fig. 4- Instancia patrón SOA [IDE Genexus]

Respecto de la aplicación del patrón, cada parte de cada objeto puede tener un valor por defecto, podemos tener formas predeterminadas de Web Panel, eventos, web form, etc. El valor predeterminado para cada parte puede ser diferente para cada objeto específico.

Todos los objetos generados por los patrones tienen partes por defecto, es decir que no han sido modificadas manualmente, estas se muestran de la siguiente manera en el IDE (notar el icono verde):

Cuando se modifica alguna parte del objeto generado, el icono cambia, en este ejemplo fijarse que en eventos el icono verde cambia por uno rojo:

La implementación basada en partes por defecto provee dinamismo entre la transacción y el patrón, esto significa que un cambio en alguna propiedad en la definición del patrón, agregar un nuevo filtro en la instancia o algún cambio en la transacción, todos los objetos generados por el patrón automáticamente reaccionarán al cambio sin la necesidad de re-aplicar el patrón.

2.2 Arquitecturas de software

En el presente trabajo se usa la arquitectura orientada a servicios para dar solución a los temas de reutilización y para el desarrollo rápido de aplicaciones la herramienta Genexus, en la siguiente sección se definirán los conceptos asociados a la arquitectura de software y algunas de las arquitecturas más usadas, esto con el fin de hacer un paralelo entre las arquitecturas más populares versus la ocupada en este trabajo (SOA).

La arquitecturas de software nacen con el crecimiento de los sistemas que debían ser desarrollados, estos pasaron de ser sistemas pequeños a sistemas de mediana y gran envergadura, es durante este proceso que se dieron cuenta de que la base existente para el desarrollo de software por equipos no bastaba para crear sistemas más complejos, de esta necesidad es que nace la arquitectura de software, la cual se encarga de descomponer el sistema en diferentes aspectos que al relacionarse dan origen a la solución.

Otra definición reconocida es la de Clemens [15]:

> "La arquitectura de software es a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según la percibe el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y supresión del detalle".

Se entiende por componentes los bloques de construcción que conforman las partes de un sistema de software, a nivel de lenguajes de programación, pueden ser representados como módulos, clases, objetos o un conjunto de funciones relacionadas. Un componente puede ser algo tan simple como un módulo de programa, pero también puede ser algo tan complicado como incluir bases de datos y software intermedio.

Se distinguen tres tipos de componentes [16], denominados también elementos:

- Elementos de Datos: contienen la información que será transformada.
- Elementos de Proceso: transforman los elementos de datos.
- Elementos de Conexión: llamados también conectores, que bien pueden ser elementos de datos o de proceso, y mantienen unidas las diferentes piezas de la arquitectura.

Una relación es una abstracción de la forma en que los componentes interactúan. La relación se concreta mediante conectores.

Según Len Bass [17] todos los sistemas tienen asociada una arquitectura dado que los sistemas están formados por componentes y relaciones, sin embargo esta arquitectura puede no ser conocida por todos los involucrados en el desarrollo del sistema.

La importancia de la arquitectura de software radica en 3 aspectos fundamentales [17]:

1. Las representaciones de la arquitectura de software facilitan la comunicación entre todas las partes interesadas en el desarrollo de un sistema basado en computadora.

2. la arquitectura destaca decisiones tempranas de diseño que tendrán un profundo impacto en todo el trabajo de ingeniería del software que sigue, y es tan importante en el éxito final del sistema como una entidad operacional.

3. la arquitectura constituye un modelo relativamente pequeño e intelectualmente comprensible de cómo está estructurado el sistema y de cómo trabajan juntos sus componentes.

El estilo y estructura particulares elegidos para una aplicación puede, por lo tanto, depender de los requerimientos no funcionales del sistema:

1. Rendimiento. Si el rendimiento es un requerimiento crítico, la arquitectura debería diseñarse para identificar las operaciones críticas en un pequeño número de subsistemas, con tan poca comunicación como sea posible entre estos subsistemas. Esto puede significar el uso relativo de componentes de grano grueso (componentes de mayor tamaño) en vez de componentes de grano fino para reducir las comunicaciones entre ellos.

2. Protección. Si la protección es un requerimiento crítico, debería usarse una arquitectura estructurada en capas, con los recursos más críticos protegidos en las capas más internas y aplicando una validación de seguridad de alto nivel en dichas capas.

4. Seguridad. Si la seguridad es un requerimiento crítico, la arquitectura debería diseñarse para que las operaciones relacionadas con la seguridad se localizaran en un único subsistema o en un pequeño número de subsistemas. Esto reduce los costes y los problemas de validación de seguridad y hace posible crear los sistemas de protección relacionados con los de seguridad.

5. Disponibilidad. Si la disponibilidad es un requerimiento crítico, la arquitectura debería diseñarse para incluir componentes redundantes y para que sea posible reemplazar y actualizar componentes sin detener el sistema.

6. Mantenibilidad. Si la mantenibilidad es un requerimiento crítico, la arquitectura del sistema debería diseñarse usando componentes independientes de grano fino que puedan modificarse con facilidad. Los productores de los datos deberían separarse de los consumidores y deberían evitarse las estructuras de datos compartidas.

Las actividades que determinan cual será la arquitectura que se va a utilizar, pueden ubicarse en las fases tempranas del ciclo de desarrollo del sistema: luego del análisis de los requerimientos y el análisis de riesgos y justo antes del

diseño detallado, las actividades arquitectura de software varían dependiendo del tipo de software que se debe diseñar y el arquitecto de software [11].

El estudio arquitectónico es un proceso creativo en el que se establece la forma en que el sistema se organizará, en esta etapa el encargado debe responder preguntas tales como:

1. ¿Existe una arquitectura que se pueda utilizar como base?
2. ¿Qué estilo o estilos arquitectónicos son apropiados para el sistema?
3. ¿Cómo se puede descomponer el sistema?

Las responsabilidades del encargado de la arquitectura de software dentro de un proyecto son:

1. Definir los módulos principales.
2. Definir las responsabilidades que tendrá cada uno de estos módulos.
3. Definir la interacción que existirá entre dichos módulos.
4. Control y flujo de datos.
5. Secuenciación de la información.
6. Protocolos de interacción y comunicación.
7. Ubicación en el hardware.

2.3 Estilos arquitectónicos

Un estilo arquitectónico es un patrón de organización de un sistema, es una colección de decisiones de diseño a nivel arquitectónico que son aplicables a un contexto de desarrollo dado, además restringen las decisiones de diseño arquitectónico y ponen de manifiesto las calidades del sistema resultante [18].

Un estilo arquitectónico define una familia de sistemas en términos de un patrón de organización estructural, consta de:

- Un conjunto de componentes, que realiza una función requerida por el sistema.
- Un conjunto de conectores que posibilitan la comunicación, la coordinación y la cooperación entre los componentes.
- Restricciones que definen como se puede integrar los componentes que forman el sistema.

- Modelos semánticos que permiten al diseñador entender las propiedades globales de un sistema para analizar las propiedades conocidas de sus partes constituyentes.

El uso de estilos arquitectónicos brinda varias ventajas:

- Abstraen los elementos y aspectos formales de varias arquitecturas específicas.
- Sirven para sintetizar estructuras de soluciones.
- Pocos estilos abstractos encapsulan una enorme variedad de configuraciones concretas.
- Permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales.
- Coordinan arquitectos.
- Evitan el debilitamiento del software por violar la arquitectura.
- Evitan el desvío (perjudicar la coherencia y claridad por insensibilidad a la arquitectura).

Las arquitecturas de la mayoría de los sistemas grandes no utilizan un único estilo. Pueden diseñarse diferentes partes del sistema utilizando distintos estilos arquitectónicos [18].

A medida que avanza el tiempo y con ello las tecnologías, van surgiendo nuevas arquitecturas, donde la característica principal de cada una de ellas es apuntar a la abstracción.

A continuación se mencionan los estilos arquitectónicos más conocidos:

2.3.1 Monolítico

En este estilo, todo el sistema se ejecuta como un solo programa en modo kernel. El sistema se escribe como una colección de procedimientos, enlazados entre sí en un solo programa binario ejecutable extenso. Cuando se utiliza esta técnica, cada procedimiento en el sistema tiene la libertad de llamar a cualquier otro. Al tener miles de procedimientos que se pueden llamar entre sí sin restricción, con frecuencia se produce un sistema poco manejable y difícil de comprender [9].

En términos de ocultamiento de información, en esencia no hay nada: todos los procedimientos son visibles para cualquier otro procedimiento (en contraste a una estructura que contenga módulos o paquetes, en donde la mayor parte de la información se oculta dentro de módulos y sólo los puntos de entrada designados de manera oficial se pueden llamar desde el exterior del módulo).

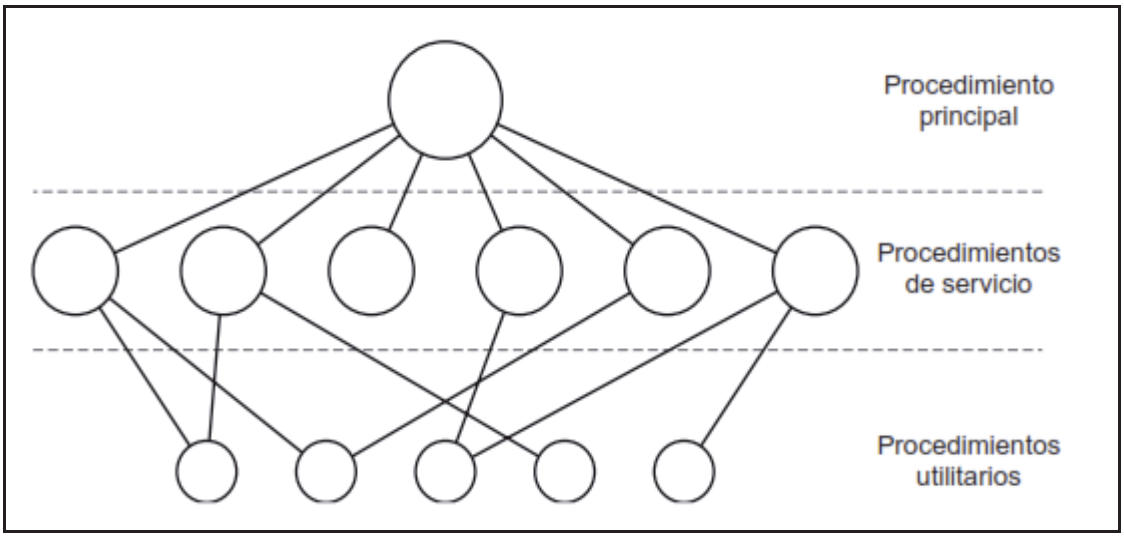


Fig. 5- Arquitectura monolítica [9]

2.3.2 Flujos de datos

Esta familia enfatiza el aseguramiento de cualidades de reutilización y modificabilidad. Este estilo arquitectónico es apropiado para sistemas que implementan transformaciones de datos en pasos sucesivos.

En esta arquitectura, las transformaciones funcionales procesan las entradas y producen salidas. Los datos fluyen de una función a otra y se transforman a medida que se mueven a través de la secuencia de funciones. Cada paso de procesamiento se implementa como una transformación.

Los datos de entrada fluyen a través de estas transformaciones hasta que se convierten en datos de salida. Las transformaciones se pueden ejecutar secuencialmente o en paralelo. Los datos pueden ser procesados por cada transformación: elemento a elemento o en un único lote [11].

Ejemplos:

- Procesamiento en lote: Los pasos de procesamiento son programas independientes; cada paso se ejecuta completamente antes de que comience el siguiente. Los pasos son programas independientes, y corren en una secuencia predefinida.

- Tubos y filtros (pipeline): Cada componente, denominado filtro, tiene un conjunto de entradas y otro de salidas. Un componente lee flujos de datos de sus entradas y produce flujos de datos en sus salidas mediante una transformación.

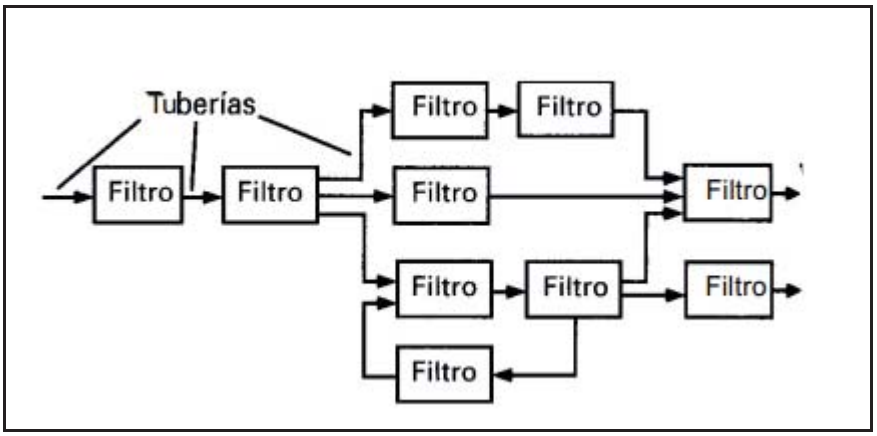


Fig. 6 - Filtros y tuberías [10]

Ventajas

- Permite la reutilización de transformaciones.

- Es intuitiva puesto que muchas personas piensan en su trabajo en términos de procesamiento de entradas y salidas.

- Generalmente se puede hacer evolucionar de forma directa el sistema añadiendo nuevas transformaciones.

- Es sencilla de implementar ya sea como un sistema concurrente o como uno secuencial.

El principal problema con este estilo es que tiene que haber un formato común para transferir los datos, de forma que puedan ser reconocidos por todas las transformaciones. Cada transformación debe estar acorde con las transformaciones con las que se comunica sobre el formato de los datos a procesar o bien se debe imponer un formato estándar para todos los datos comunicados [11].

2.3.4 Arquitectura de llamada y retorno

Este estilo arquitectónico persigue la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas de gran escala. Sub-estilos de esta arquitectura son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objetos y los sistemas jerárquicos en capas [1].

2.3.5 Programa principal y subrutinas

Estilo clásico desde los años sesenta, se basa en la descomposición en componentes/subrutinas, el objetivo es descomponer jerárquicamente en subrutinas (componentes) que solucionan una tarea o función definida. Los datos son pasados como parámetros y el manejador principal proporciona un ciclo de control sobre las subrutinas. Reflejan la estructura del lenguaje de programación. Permite al diseñador del software construir una estructura de programa relativamente fácil de modificar y ajustar a escala. Se basan en la bien conocida abstracción de procedimientos/funciones/métodos. La arquitectura de llamada de procedimiento remoto es similar con la salvedad que los componentes/subrutinas están distribuidos entre varias computadoras en red [1].

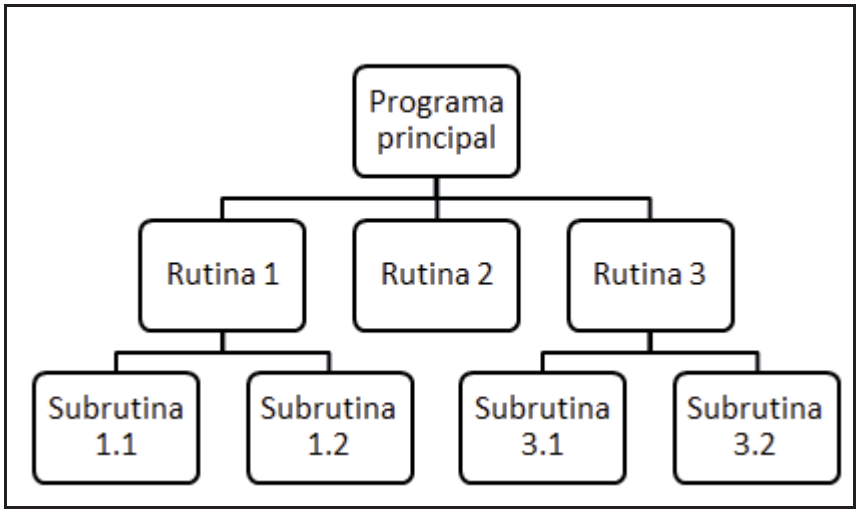


Fig. 7 – Programa principal y subrutinas [11]

Características

- Utilizados en grandes sistemas de software.
- Usa una estructura implícita de subsistemas.

- Razonamiento jerárquico, cambios en una subrutina implican cambios en las subrutinas que hacen la invocación.

- Pretenden incrementar el desempeño distribuyendo el trabajo en múltiples procesadores.

Ventajas

- La descomposición en módulos disminuye la complejidad.

- Hilo de control simple soportado por los lenguajes de programación.

2.3.6 Arquitectura Cliente-Servidor

El modelo arquitectónico cliente-servidor es un modelo de sistema en el que dicho sistema se organiza como un conjunto de servicios y servidores asociados, más unos clientes que acceden y usan los servicios. Los principales componentes de este modelo son:

- Un conjunto de servidores que ofrecen servicios a otros subsistemas. Ejemplos de servidores son servidores de impresoras que ofrecen servicios de impresión, servidores de ficheros que ofrecen servicios de gestión de ficheros y servidores de compilación, que ofrecen servicios de compilación de lenguajes de programación.

- Un conjunto de clientes que llaman a los servicios ofrecidos por los servidores. Estos son normalmente subsistemas en sí mismos. Puede haber varias instancias de un programa cliente ejecutándose concurrentemente.

- Una red que permite a los clientes acceder a estos servicios. Esto no es estrictamente necesario ya que los clientes y los servidores podrían ejecutarse sobre una única máquina. En la práctica, sin embargo, la mayoría de los sistemas cliente-servidor se implementan como sistemas distribuidos.

Los clientes pueden conocer los nombres de los servidores disponibles y los servicios que éstos proporcionan. Sin embargo, los servidores no necesitan conocer la identidad de los clientes o cuántos clientes tienen. Los clientes acceden a los servicios proporcionados por un servidor a través de llamadas a procedimientos remotos usando un protocolo de petición-respuesta tal como el protocolo http, básicamente, un cliente realiza una petición a un servidor y espera hasta que recibe una respuesta [1].

Características

- Es un estilo para sistemas distribuidos.

- Divide el sistema en una aplicación cliente, una aplicación servidora y una red que las conecta.

- Describe una relación entre el cliente y el servidor del cual el primero realiza las peticiones y el segundo envía las respuestas.

- Puede usar un amplio rango de protocolos y formatos de datos para comunicar la información.

Ventajas

- Más seguridad ya que los datos se almacenan en el servidor que generalmente ofrece más control sobre la seguridad

- Acceso centralizado a los datos que están almacenados en el servidor lo que facilita su acceso y actualización.

- Facilidad de mantenimiento ya que los roles y las responsabilidades se distribuyen entre los distintos servidores a través de la red lo que permite que un cliente no se vea afectado por un error en un servidor en particular.

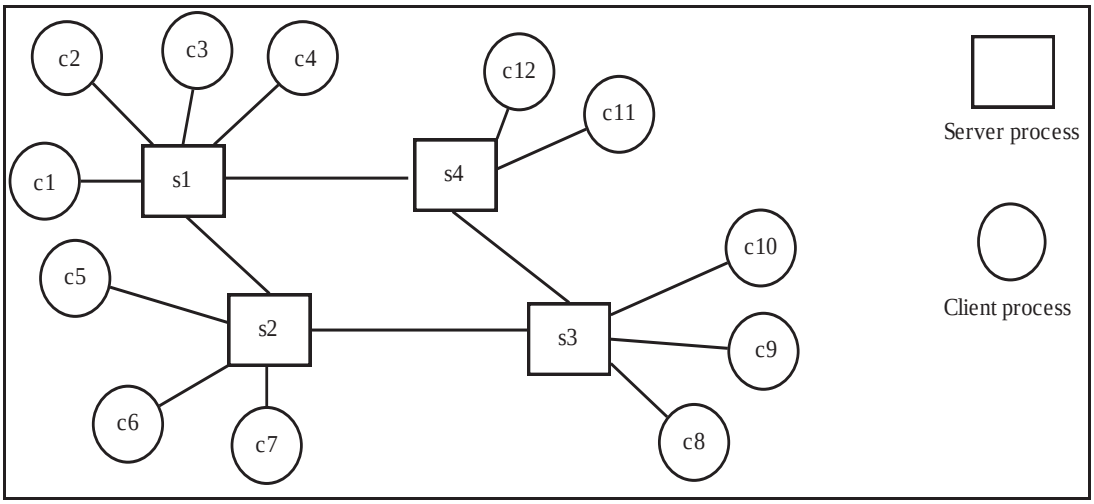


Fig. 8 - Arquitectura Cliente – servidor [11]

2.3.6 El modelo de capas

Tal como su nombre lo dice en este tipo de modelos el sistema se divide en diferentes capas que ofrecen diversos servicios. La ventaja principal de este modelo es que el desarrollo se puede realizar en varios niveles, esto es de gran utilidad a la hora de realizar mantenciones, ya que las modificaciones se hacen en el nivel requerido, sin revisar

código mezclado, otra ventaja es que el desarrollo se puede realizar por distintas personas o equipos de trabajo abstraídas de los demás niveles. Cada capa proporciona servicios a la capa superior y actúa como cliente de la capa inferior [1]**.**

Dentro estos modelos el más reconocido es el modelo de tres capas, el cual comprende las siguientes capas:

1. Capa de presentación: Es lo que el usuario ve, le comunica la información y captura la información del usuario. También es conocida como interfaz gráfica. Esta capa se comunica únicamente con la capa de negocio.

2. Capa de negocio: Esta capa se preocupa de la lógica del negocio (las reglas que se deben cumplir), es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Esta capa se comunica la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar la actualización, inserción o eliminación de datos, además de la recuperación de información.

3. Capa de datos: es donde residen los datos y es la encargada de acceder a los mismos. Reciben solicitudes de almacenamiento, eliminación o recuperación de información desde la capa de negocio.

Características

- Los componentes se organizan en capas.
- Los conectores son definidos por los protocolos que determinan cómo interactúan las capas.
- Restricciones topológicas incluyen limitar las interacciones a capas adyacentes.

La aplicabilidad de este estilo arquitectónico es en grandes sistemas caracterizados por una mezcla de elementos de alto y bajo nivel, donde los elementos de alto nivel dependen de los de bajo nivel. Cada nivel tiene asociada una funcionalidad:

- En los niveles bajos se manejan funciones simples ligadas al hardware o al entorno.
- En los niveles más altos se manejan funciones más abstractas.

Ventajas

- Facilita la migración. El acoplamiento con el entorno está localizado en las capas inferiores. Estas son las únicas a re-implementar en caso de transporte a un entorno diferente.

- Reutilización: Cada nivel implementa unas interfaces claras, las lógicas internas pueden intercambiarse sin afectar a los demás componentes.
- Mantenimiento: los cambios en una capa apenas afectan sólo a la capa superior e inferior.
- Permite trabajar en varios niveles de abstracción. Para implementar los niveles superiores no se requiere conocer el entorno subyacente, basta con las interfaces que proporcionan los niveles inferiores.

2.3.7 Arquitectura orientada a objetos

Los componentes de un sistema encapsulan los datos y las operaciones que se deben realizar para manipular los datos. La comunicación y la coordinación entre componentes se consiguen a través del paso de mensaje. La representación de los datos y sus operaciones primitivas asociadas son encapsuladas en un tipo de dato abstracto u objeto.

Un sistema orientado a objetos está compuesto de objetos que interactúan, los cuales mantienen ellos mismos su estado local y proveen operaciones sobre su estado. La representación del estado es privada y no se puede acceder a ella directamente desde fuera del objeto. El proceso de diseño orientado a objetos comprende el diseño de clases de objetos y las relaciones entre estas clases. Las clases definen los objetos del sistema y sus interacciones. Cuando el diseño se implementa como un programa ejecutable, los objetos requeridos se crean dinámicamente utilizando las definiciones de las clases [11].

Los objetos son componentes potencialmente reutilizables debido a que son encapsulamientos independientes del estado y las operaciones Los diseños se pueden desarrollar utilizando objetos creados en los diseños previos. Esto reduce los costes de diseño, programación y validación. También conduce a la utilización de objetos estándar (por lo que se mejora la comprensión del diseño) y reduce los riesgos implicados en el desarrollo de software. Sin embargo algunas veces la reutilización se implementa de una mejor forma si se utilizan componentes o marcos de trabajo en lugar de objetos individuales [11], en este estudio se opta por diseñar con estas alternativas en vez de orientación a objetos, además el lenguaje no lo permite.

Características

- En este estilo los componentes son los objetos, o instancias de tipos de datos abstractos.
- Los componentes contienen los datos y el comportamiento para trabajar con esos datos y además tienen un rol o responsabilidad distinta.
- Hace hincapié en la reutilización a través de la encapsulación, la modularidad, el polimorfismo y la herencia.

- Contrasta con el enfoque procedimental donde hay una secuencia predefinida de tareas y acciones. El enfoque orientado a objetos emplea el concepto de objetos interactuando unos con otros para realizar las tareas.

- Los objetos interactúan a través de invocaciones a procedimientos y funciones.

Ventajas

- Modificabilidad: Como un objeto oculta su representación a sus clientes, es posible cambiar su implementación sin modificar los clientes.

- La integración de un conjunto de rutinas de acceso con los datos que manipulan permite a los diseñadores descomponer los problemas en colecciones de agentes que interactúan.

2.3.8 Arquitectura orientada a servicios (SOA)

La arquitectura orientada a servicios representa un modelo arquitectónico que tiene como objetivo mejorar la agilidad y la rentabilidad de una empresa, al tiempo que reduce la carga de TI en la organización en general. Esto se logra, mediante el posicionamiento de los servicios como el principal medio a través del cual se representa la lógica de las soluciones [28].

Históricamente, el término "arquitectura orientada a servicios" se ha utilizado tan ampliamente por los medios de comunicación y dentro de la literatura de marketing, que casi se ha convertido en sinónimo de orientación a servicios. La orientación a servicios, es un paradigma de diseño destinado a la creación de unidades lógicas de solución que se forman individualmente, de manera que puedan ser utilizados colectivamente y repetidamente en apoyo de la obtención de los objetivos estratégicos específicos.

Como una arquitectura de software, SOA, puede consistir en una combinación de tecnologías, productos, APIs, extensiones de apoyo a la infraestructura y muchas otras partes.

La forma actual de una arquitectura de software desplegada en cada empresa, es única; sin embargo, se caracteriza por la introducción de nuevas tecnológicas y nuevas plataformas, las cuales específicamente soportan la creación, ejecución y evolución de las soluciones orientadas a servicios [28].

SOA es un estilo de Arquitectura de Software basado en la definición de servicios reutilizables, con interfaces públicas bien definidas, donde los proveedores y consumidores de servicios interactúan en forma desacoplada para realizar los procesos de negocio Establece un conjunto de principios y metodologías para el diseño y desarrollo de software en forma de interoperabilidad de servicios [29].

El objetivo de SOA es crear sistemas de información altamente escalables, reutilizables y brindar una forma bien definida de invocación de servicios, promoviendo la interacción entre distintos sistemas.
Cuando SOA se refiere a invocación de servicios la mayoría de las veces son servicios web (web service) pero esto no es exclusivo, ya que existen otros servicios con los cuales se puede implementar.

Las diferencias de SOA con las principales arquitecturas existentes se pueden visualizar en la siguiente tabla:

	Programación Estructurada	**Objetos**	**Componentes**	**Servicios**
Granularidad	Muy fina	Fina	Intermedia	Gruesa
Contrato	Definido	Privado/Publico	Publico	Publicado
Reusabilidad	Baja	Baja	Intermedia	Alta
Acoplamiento	Fuerte	Fuerte	Débil	Muy débil
Dependencias	Tiempo de Compilación	Tiempo de Compilación	Tiempo de Compilación	Run-Time
Ámbito de Comunicación	Intra-Aplicación	Intra-Aplicación	Inter-Aplicaciones	Inter-Empresas

Tabla 2 - Diferencias entre arquitecturas [1].

Como se puede ver en esta imagen, la arquitectura orientada a servicios presente un mayor nivel de abstracción y algunas ventajas adicionales, como lo son la reutilización, el bajo acoplamiento y la forma de publicar los servicios, lo cual hace que pueda ser consumidos tanto inter-aplicaciones como inter-empresas, quizás la mayor dificultad, se presente a la hora de vincular los servicios ya que estos se revisan en tiempo de ejecución y no al momento de compilar, tal como lo hacen las otras arquitecturas.

Los tecnicismos más frecuentes de la SOA y su importancia para la empresa [4]:

- **Granularidad gruesa**: describe el tamaño de los componentes que constituyen un sistema. La SOA prefiere los componentes de mayor tamaño (de grano grueso) a los que se conoce como servicios de negocio. Generalmente estos se construyen a partir de otros servicios técnicos más pequeños (de grano fino) que ya existen. Esto es importante porque las piezas más grandes favorecen que el personal de la empresa comprenda, reutilice y maneje los servicios de la SOA.

- **Acoplamiento débil:** El modo de diseñar servicios más flexibles y menos dependientes unos de otros. Con ello se facilita el ensamblaje de los servicios y su recombinación en nuevos contextos. Es importante porque resulta más rápido agrupar soluciones de negocio a partir de piezas prefabricadas que escribir desde cero cada una de las nuevas funciones.

- **Los contratos:** definen las obligaciones entre el proveedor y el consumidor del servicio. Pueden contemplar expectativas sobre el servicio tales como disponibilidad, fiabilidad, indicadores clave de rendimiento, costes y asistencia. Son importantes porque ayudan a los usuarios del negocio a adoptar decisiones informadas sobre los servicios en los que pueden confiar.

- **Interfaz frente a implementación:** diferencia entre lo que hace un servicio de cómo lo hace. Esto es importante porque así, el usuario del negocio centra su atención sobre lo que hace el servicio y no en los tediosos detalles de funcionamiento interno de la tecnología.

Componentes de SOA:

Servicios : Entidades lógicas, contratos definidos por una o más interfaces públicas.

Proveedor del servicio : Entidad de software que implementa una especificación de servicio.

Consumidor del servicio : Entidad de software que llama a un proveedor de servicio. Tradicionalmente se lo llama "cliente". Puede ser una aplicación final u otro servicio.

Localizador de servicio :Tipo específico de proveedor de servicio que actúa como registro y permite buscar interfaces de proveedor de servicios y sus ubicaciones, se le puede considerar un catálogo de las ubicaciones servicios.

Service bróker : Tipo específico de proveedor de servicios que puede pasar requerimientos de servicios a otro proveedor de servicios, describe los parámetros que necesita un servicio, este servicio es útil para establecer los protocolos de seguridad además, funciona como un puente, entregando características de portabilidad a los demás servicios.

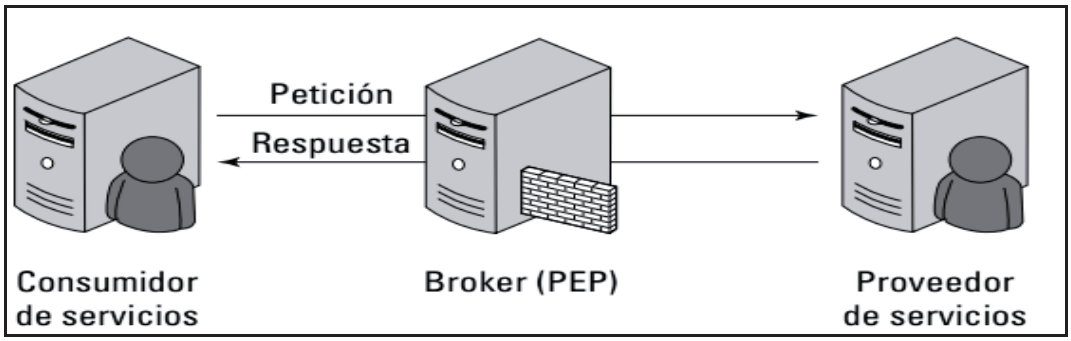


Fig. 9- Componentes SOA [4]

La estrategia de orientación a servicios permite la creación de servicios y aplicaciones compuestas que pueden existir con independencia de las tecnologías subyacentes. En lugar de exigir que todos los datos y lógica de negocio residan en un mismo ordenador, el modelo de servicios facilita el acceso y consumo de los recursos de IT a través de la red. Puesto que los servicios están diseñados para ser independientes, autónomos y para interconectarse adecuadamente, pueden combinarse y recombinarse con suma facilidad en aplicaciones complejas que respondan a las necesidades de cada momento en el seno de una organización [3].

Servicios web

Los servicios web son la forma más habitual de implementar SOA, Los servicios Web son aplicaciones que utilizan estándares para el transporte, codificación y protocolo de intercambio de información.

Los servicios Web permiten la intercomunicación entre sistemas de cualquier plataforma y se utilizan en una gran variedad de escenarios de integración, tanto dentro de las organizaciones como con partners de negocios (empresas externas) [3].

Los servicios Web se basan en un conjunto de estándares de comunicación, como son XML para la representación de datos, SOAP (Simple Object Access Protocol) para el intercambio de datos y el lenguaje WSDL (Web Services Description Language) para describir las funcionalidades de un servicio Web. Existen más especificaciones, a las que se denomina genéricamente como la arquitectura WS, que definen distintas funcionalidades para la gestión de eventos, el descubrimiento de servicios Web, archivos adjuntos, seguridad, gestión y fiabilidad en el intercambio de mensajes y transacciones.

Básicamente un servicio SOA consistiría en [4]:

- Lo que el servicio hace: Un servicio proporciona una capacidad para su consumidor, como por ejemplo, procesar el cambio de dirección de un cliente de un banco.

- Cómo se utiliza: Un servicio cuenta con un método específico para poder usarlo, lo que se llama invocación. Presenta una interfaz bien definida para poder acceder a sus prestaciones.

Lo que no se define explícitamente en un servicio de SOA es:

- Dónde está ubicado el servicio. Se puede acceder a los servicios de forma remota, es decir, que puede llamarlos desde cualquier punto de una red.

- Cómo funciona. Los servicios son opacos, lo que significa que ni se sabe, ni importa, cómo realizan su trabajo.

Los servicios de SOA pueden acoplarse para construir otros nuevos, y ensamblarse en secuencias para construir procesos.

Beneficios de SOA

Los beneficios de SOA para una organización se plasman a dos niveles distintos: al del usuario corporativo y a nivel de la organización de IT. Desde el punto de vista de la empresa, SOA permite el desarrollo de una nueva generación de aplicaciones dinámicas que resuelven una gran cantidad de problemas de alto nivel, fundamentales para el crecimiento y la competitividad. Las soluciones SOA permiten entre otras cosas [3]:

- Mejorar la toma de decisiones. Al integrar el acceso a los servicios e información de negocio dentro de un conjunto de aplicaciones dinámicas compuestas, los directivos disponen de más información y de mejor calidad (más exacta y actualizada). Las personas, procesos y sistemas que abarcan múltiples departamentos pueden introducirse de forma más directa en una panorámica unificada, lo que permite conocer mejor los balances de costes y beneficios que se producen en las operaciones de negocio que se realizan a diario. Y al disponer de mejor información en un tiempo menor, las organizaciones pueden reaccionar de manera más ágil y rápida cuando surgen problemas o cambios.

- Mejorar la productividad de los empleados. Un acceso óptimo a los sistemas y la información y la posibilidad de mejorar los procesos permiten a las empresas aumentar la productividad individual de los empleados. Estos pueden dedicar sus energías a los procesos importantes, los que generan valor añadido y a actividades de colaboración, semi-estructuradas, en vez de aceptar las limitaciones y restricciones impuestas por los sistemas de IT rígidos y monolíticos. Más aún: puesto que los usuarios pueden acceder a la información en los formatos y modalidades de presentación (web, cliente avanzado, dispositivo móvil), que necesitan, su productividad se multiplica en una gran cantidad de escenarios de uso, habituales o nuevos.

- Potenciar las relaciones con clientes y proveedores. Las ventajas de SOA trascienden las fronteras de la organización. Los beneficios que ofrece SOA trascienden los límites de la propia organización. Los procesos de fusión y compra de empresas se hacen más rentables al ser más sencilla la integración de sistemas y aplicaciones diferentes. La integración con partners comerciales y la optimización de los procesos de la cadena de suministro son, bajo esta perspectiva, objetivos perfectamente asequibles. Con SOA se puede conseguir mejorar la capacidad de respuesta a los clientes, habilitando por ejemplo portales

unificados de servicios. Si los clientes y proveedores externos pueden disponer de acceso a aplicaciones y servicios de negocio dinámicos, no solamente se permite una colaboración avanzada, sino que se aumenta la satisfacción de clientes y proveedores. SOA permite flexibilizar los procesos críticos de compras y gestión de pedidos –habilitando modalidades como la subcontratación de ciertas actividades internas- superando las restricciones impuestas por las arquitecturas de IT subyacentes, y con ello consiguiendo un mejor alineamiento de los procesos con la estrategia corporativa.

SOA contribuye también a documentar el modelo de negocio de la empresa y a utilizar el modelo de negocio documentado para integrar en él y dar respuesta a las dinámicas de cambio que se produzcan y optimizarlo de acuerdo con ellas.

Desde el punto de vista de los departamentos de IT, la orientación a servicios supone un marco conceptual mediante el cual se puede simplificar la creación y mantenimiento de sistemas y aplicaciones integradas, y una fórmula para alinear los recursos de IT con el modelo de negocio y las necesidades y dinámicas de cambio que le afectan.

- Aplicaciones más productivas y flexibles. La estrategia de orientación a servicios permite a IT conseguir una mayor productividad de los recursos de IT existentes, como pueden ser las aplicaciones y sistemas ya instalados e incluso los más antiguos y obtener mayor valor de ellos de cara a la organización sin necesidad de aplicar soluciones de integración desarrolladas ex profeso para este fin. La orientación a servicios permite además el desarrollo de una nueva generación de aplicaciones compuestas que ofrecen capacidades avanzadas y multifuncionales para la organización con independencia de las plataformas y lenguajes de programación que soportan los procesos de base. Más aún: puesto que los servicios son entidades independientes de la infraestructura subyacente, una de sus características más importantes es su flexibilidad a la hora del diseño de cualquier solución.

- Desarrollo de aplicaciones más rápido y económico. El diseño de servicios basado en estándares facilita la creación de un repositorio de servicios reutilizables que se pueden combinar en servicios de mayor nivel y aplicaciones compuestas en respuesta a nuevas necesidades de la empresa. Con ello se reduce el coste del desarrollo de soluciones y de los ciclos de prueba, se eliminan redundancias y se consigue su puesta en valor en menos tiempo. Y el uso de un entorno y un modelo de desarrollo unificados simplifica y homogeniza la creación de aplicaciones, desde su diseño y prueba hasta su puesta en marcha y mantenimiento.

- Aplicaciones más seguras y manejables Las soluciones orientadas a servicios proporcionan una infraestructura común (y una documentación común también) para desarrollar servicios seguros, predecibles y gestionables. Conforme van evolucionando las necesidades de negocio, SOA facilita la posibilidad de añadir nuevos servicios y funcionalidades para gestionar los procesos de negocio críticos. Se accede a los servicios y no a las aplicaciones, y gracias a ello la arquitectura orientada a servicios optimiza las inversiones realizadas en IT potenciando la capacidad de introducir nuevas capacidades y mejoras, y

además, puesto que se utilizan mecanismos de autenticación y autorización robustos en todos los servicios y puesto que los servicios existen de forma independiente unos de otros y no se interfieren entre ellos.

Consejos para la correcta implementación de SOA

Tanto a nivel organizativo como técnico embarcarse en un proyecto de SOA supone tener que resolver una serie de retos, estos pueden convertirse en verdaderas barreras insuperables si se ha partido de la idea de que SOA es el remedio para toda clase de males y problemas.

Para que las iniciativas de adopción de SOA tengan un fin satisfactorio, hay que asegurarse de que se cumplen una serie de condiciones indispensables [3]:

- **Definir claramente los objetivos de negocio**. El primer paso a la hora de adoptar SOA es identificar con claridad los problemas o retos empresariales más prioritarios. Cuando más precisa sea esa formulación, más fácilmente se podrá delimitar el alcance de cualquier proyecto SOA. Disponer de una visión y un rumbo claro desde el principio hará mucho más fácil la ejecución de procesos cuya esencia es la integración de múltiples funciones.

- **Definir claramente el alcance del proyecto SOA.** El objetivo de cualquier proyecto SOA no debe consistir en renovar de forma indiscriminada y masiva toda la infraestructura de IT. Este tipo de megaproyectos fracasan a la hora de implementarlos porque cuando por fin se ha conseguido crear la solución, las condiciones del negocio suelen haber cambiado tanto que los problemas que ahora deben resolverse ya no tienen mucho que ver con aquellos que se pretendían resolver cuando se inició el proyecto. El objetivo real de cada iniciativa SOA debe ser responder a necesidades concretas de negocio y crear soluciones en pasos discretos, incrementales e iterativos.

- **Evitar introducir SOA sin motives reales que lo justifiquen**: La adopción de SOA no debe considerarse una necesidad tecnológica, sino organizativa: debe responder a las necesidades de la organización. Si la introducción de SOA solamente responde al puro gusto por disponer de SOA y se empiezan a crear servicios sin un significado de negocio claro, sin la granularidad adecuada o con demasiadas interconexiones, el resultado será una implementación excesivamente compleja, inmanejable y tremendamente costosa.

- **Gestionar el proceso:** Los servicios y aplicaciones se corresponden con procesos y los outputs de información deseados a través de las diversas áreas funcionales de la organización. Puesto que representan procesos compartidos, es necesario que se les asigne un propietario para que puedan inventariarse y gestionarse a fin de garantizar que cumplen en todo momento con las directivas corporativas y responden adecuadamente a las necesidades que los justifican.

La adopción de una arquitectura basada en ser vicios requiere de una infraestructura de comunicaciones escalable y segura entre los componentes. Esto es lo que se conoce como Enterprise Service Bus.

El Bus de Servicios Empresariales es el concepto que se refiere a la infraestructura de transporte de mensajes entre el motor de procesos y los servicios de los que dispone la empresa.

El bus requiere una infraestructura sólida que ofrezca a los desarrolladores la seguridad de que los mensajes sean entregados siempre independientemente de los problemas que puedan afectar a un servicio determinado en un momento dado. Esto se logra utilizando colas de mensajes para conectar de manera asíncrona los distintos componentes. Sin embargo, en algunos casos, por ejemplo consultas, puede no ser viable usar una comunicación asíncrona y HTTP puede ser entonces una alternativa

Las colas de mensaje se utilizan para comunicar distintas aplicaciones, garantizando la entrega e integridad de los mensajes, reduciendo la complejidad del desarrollo. Se suelen utilizar para mejorar la disponibilidad y escalabilidad de los sistemas. Las colas pueden ser persistentes o no y se utilizan en esquemas punto a punto o publicación y suscripción.

El bus debe ofrecer seguridad total y conectividad hacia todo el software de infraestructura de la empresa

El bus debe ser totalmente monitoreable por que se convierte en la columna vertebral de todos los sistemas de la empresa.

2.4 Componentes de software

Los Patrones genexus son componentes de software que extienden la funcionalidad de la herramienta. El concepto de componente de software, es otro de los términos en que la comunidad informática no ha logrado ponerse de acuerdo, una discusión sobre este tema se puede leer en [7] [19].

Algunas definiciones de componentes de software son las siguientes:

- Según Philippe Krutchen de Rational Rose [19], un componente es una parte no trivial, casi independiente y reemplazable de un sistema que cumple una función dentro del contexto de una arquitectura bien definida. Un componente cumple con un conjunto de interfaces y provee la realización física de ellas.

- Según Clemens Szyperski [33], un componente de software es una unidad de composición con interfaces especificadas contractualmente y solamente dependencias explícitas de contexto. Un componente de software puede ser desplegado independientemente y está sujeto a composición por terceros.

- Según Herzum y Sims [5], un componente es un artefacto de software autocontenido y claramente identificable que describe o ejecuta funciones específicas; que tiene, además, una interfaz claramente establecida, una documentación apropiada y un status de uso recurrente bien definido.

Para efectos de este trabajo la definición que tomaremos de componente de software será la siguiente [11]:

Un componente es un proveedor de servicios independiente. Cuando un sistema necesita algún servicio, llama a un componente para proporcionar dicho servicio, sin tener cuenta dónde se está ejecutando el componente o qué lenguaje se ha utilizado para desarrollarlo.

Las tres características principales de los componentes son las siguientes [19]:

1. Un componente es una independiente y reemplazable parte de un sistema, el cual provee una clara función.
2. El componente trabaja en el contexto de una arquitectura bien definida.
3. El componente se comunica con otros componentes mediante interfaces.

En específico el término que más se relaciona con el patrón Genexus, es el concepto de complemento, el cual es un tipo de componente de software, la definición de complemento es la siguiente [6]:

Los complementos permiten la extensión de una aplicación base con nuevas características. Son implementadas como componentes que están conectadas al núcleo en tiempo de carga o incluso en tiempo de ejecución.

….. El enfoque de complementos, fomenta la creación de aplicaciones bases compactas que pueden ser extendidas mediante complementos de acuerdo a las necesidades de los usuarios.

Los conceptos claves de esta definición son la existencia de una aplicación base la cual provee una funcionalidad, y el componente (complemento) que provee una nueva funcionalidad que la aplicación inicialmente no tiene.

Recientemente el concepto de complementos a emergido como una prometedora vía de desarrollo de aplicaciones que son inherentemente extensibles y personalizables a las necesidades de los usuarios específicos. Varios enfoques basados en complementos han encontrado su camino en la práctica [7]. Algunos ejemplos son herramienta de desarrollo Eclipse y el navegador Firefox.

El concepto de componente de software y patrón Genexus están estrechamente relacionados, básicamente ambas son librerías con funcionalidad e interfaz de usuario, que ayudan a que el programador no realice repetitivamente una tarea que puede ser automatizada, en este caso la creación de servicios, los cuales tienen la intención de ayudar a una empresa en su camino a una implementación SOA, mediante la generación rápida de servicios.

2.5 Reutilización de software

El presente trabajo tiene como objetivo crear componentes reutilizables, por lo mismo se debe definir que es lo que se considera reutilización a nivel de desarrollo de software.

La reutilización de componentes de software es un proceso inspirado en la manera que se producen y ensamblan componentes en la ingeniería de sistemas físicos. La aplicación de este concepto al desarrollo de software no es nueva. Las librerías de subrutinas especializadas comúnmente utilizadas desde la década de los setenta representan uno de los primeros intentos por reutilizar software [25].

Existen variadas definiciones del termino reutilización de software. Alguna de estas definiciones son las siguientes:

- Según Somerville [11] La reutilización es un enfoque de desarrollo de software que trata de maximizar el uso recurrente de componentes de software existentes.
- Según Sametinger [31], La reutilización de software es el proceso de crear sistemas de software a partir de software existente, en vez de partir desde el comienzo.
- Según Sodhi et al. [32], La reutilización de software es el procedo de implementar o actualizar sistemas de software usando activos de software existentes.

Los tres conceptos no difieren mucho del otro, básicamente defienden que la reutilización es el uso de componentes de software previamente construido en el desarrollo de nuevo software o mantención de existentes.

La reutilización es cada vez un aspecto más importante en la construcción de software; Durante el diseño se toman las decisiones sobre la reutilización. Los marcos, componentes y clases ya están implementados y por tanto, se pueden incluir directamente dentro del software a la hora de implementarlo [30].

Para demostrar la efectividad de la reutilización de software, se pueden extraer algunas estadísticas del libro de Sametinger [31]:

- Entre el 40 y el 60% del código fuente de una aplicación es reutilizable de otra similar.
- Aproximadamente el 60% del diseño y del código de aplicaciones administrativas es reutilizable.
- Aproximadamente el 75% de las funciones son comunes a más de un programa.
- Solo el 15% del código encontrado en muchos sistemas es único y novedoso a una aplicación específica. El rango general de uso recurrente potencial está entre: 15% y el 85%.

A partir de estos números es fácil darse cuenta de la importancia de la reutilización de software.

2.6 Trabajos relacionados

Existen pocos trabajos sobre la mejora de procesos relacionados con la construcción de software con Genexus, destaca la tesis de magister hecha por Nicolás Castagnet sobre Software Factories, para construir sistemas de Información con Genexus [2], en este trabajo se muestran los conceptos de las software factories y conceptos de Genexus sobre la construcción de extensiones, la solución que se entrega en este trabajo es la creación de una extensión para el IDE de Genexus la cual permite crear objetos con mayor abstracción del lenguaje gracias al uso de plantillas, permite orientar el desarrollo de software con Genexus a la metodología basada en modelos.

El tema principal de esta tesis es "Software Factories para construir Sistemas de Información con GeneXus", Las Software Factories, surgen como una propuesta que busca ayudar a reducir la alta tasa de fracasos que hay en la Industria de Software y mejorar su eficiencia, su definición es la siguiente:

Una Software Factory es una Línea de Productos de software que configura herramientas extensibles, procesos, y contenidos utilizando una plantilla de Software Factory, basada en un esquema de Software Factory, para automatizar el desarrollo y el mantenimiento de variantes de un producto arquetípico adaptando, ensamblando, y configurando componentes basados en frameworks.

Los problemas que intenta resolver la Software Factory construida en esta tesis de magister es:

1. Definición de interfaces de ingreso de datos y consultas.
2. Integración con Sistemas de Workflow para definir los procesos de negocio de la empresa.
3. Invocación de programas de cálculos y transformaciones desde la interfaz de usuario.
4. Administración de usuarios en el sistema.
5. Creación de menús de acceso por perfil de usuario.

La diferencia del trabajo presente, es que a pesar de ser una software factory para sistemas con Genexus, el trabajo se centra mucho en lo que es la construcción basada en modelos, dejando de lado aspectos que optimizaban el proceso de construcción con software factory, como lo son la reutilización sistemática y la generación de componentes, aspectos claves de este estudio.

Un caso similar es el que implementan los patrones Genexus, tanto Artech como empresas asociadas entregan herramientas para generar aplicaciones de manera más rápida, como se mencionó Genexus maneja una estructura para las base de datos las cuales proveen funcionalidad, estas son las llamadas transacciones, es sobre estas estructuras de datos que se puede aplicar un patrón Genexus, el cual se encarga de generar los programas necesarios

para la mantención de la tabla relacionada, es decir generan las pantallas de búsqueda y mejoran la transacción para facilitar al usuario las labores de ingreso, visualización, modificación y eliminación.

Los filtros de búsqueda, los campos a mostrar y algunas validaciones son modificados en las propiedades del patrón, evitando la escritura de código por parte del usuario.

La diferencia entre los patrones que entrega Artech y los que entregan las otras empresas varían básicamente en el aspecto y la cantidad de funcionalidad que generan automáticamente.

El enfoque que pretende perseguir Artech con estos patrones es el desarrollo rápido de aplicaciones pero no mediante la reutilización del código, sino la reutilización del conocimiento, es decir un analista Genexus se encargará de definir la estructura de la base de datos y la herramienta se encargará de generar los componentes necesarios para dejar la aplicación con las funcionalidades necesarias asociadas a esa estructura.

Dentro de los patrones Genexus más populares podemos nombrar los siguientes:

- WorkWith: Es el entregado dentro de la aplicación Genexus, este patrón permite generar los objetos básicos para el manejo de la base de datos, de uso gratuito con la licencia de Genexus.
- Workwith plus: orientado a realizar las acciones del workwith anterior pero con funcionalidades web 2.0, permite aplicar el patrón a 20 transacciones y luego se debe comprar.
- K2BTools pattern: Similar a workwith plus, la empresa K2BTool provee otras herramientas adicionales para personalizar otros objetos complementando este patrón.
- PxTools: Provee patrones con web 2.0 además de otras extensiones para la personalización de pantallas. El uso de esta herramienta es restringido a una herramienta trial.

En la página de Artech se pueden encontrar otros patrones menos populares que ayudan a resolver algunos problemas comunes del desarrollo de software, estas soluciones fueron implementadas con la herramienta para desarrollar patrones de Genexus (SDK) y son llamadas patrones de negocio:

Patrones de interfaz de usuarios: Conjunto de patrones que crea objetos relacionados con las interfaces de usuarios pero no transacciones (no modifica la información en la base de datos.)

- Patrón de asignación de múltiples valores: Dada una relación M-R de dos entidades, a veces es necesario mostrar conjunto de valores de cada una de las entidades. Con este patrón se manejan dos listados.
- Generación de archivos Excel (work with Excel): Crea un webpanel que permite descargar la estructura los registros de una tabla en un archivo Excel.

- Resumido por patrón: Genera un webpanel que muestra la data de una tabla más un gráfico, la información a mostrar se modifica en las propiedades.

- Patrón RSS Feed: Devuelve en formato de RSS la información de una transacción, para que pueda ser leida por cualquier lector de RSS.

Patrones de modelamiento de datos: Usualmente crea nueva transacciones, crea objetos involucrados en la modificación de registros de la base de datos.

- Lista de materiales: dada una transacción de "productos", genera otra transacción con los productos componentes, y los listados y funciones necesarias para saber, dado un producto, la lista de sus componentes.

- OAV Pattern: Su nombre viene de las iniciales Objeto atributo valor, permite modificar los tipos de los atributos en tiempo de ejecución, sin modificar la base de datos.

- Patrón del último valor: Va registrando el cambio del atributo en el tiempo, útil cuando es necesario rastrear el valor de un atributo para históricos. Por ej, El dolar que va cambiando y se realizan contratos en base al monto en un cierto mes.

- La diferencia entre los patrones Genexus versus el patrón que se desea desarrollar en este trabajo radica en la arquitectura ocupada, los patrones Genexus tienen relacionadas en una capa toda la operación, quedando muy atadas sus funcionalidades a la capa de datos, además la arquitectura crea lógica y objetos que solo pueden ser usados dentro de Genexus, con la cual no comparte la información con diferentes plataformas u otros sistemas, es decir no opera como creación de componentes reutilizables por diferentes aplicaciones.

Lo que se desea realizar es un patrón que ayude a la integración de aplicaciones y evitarlos problemas relacionados con el crecimiento desmedido de los sistemas, como así también crear una herramienta que permita fácilmente proveer esos servicios.

Capítulo 3 Desarrollo

3.1 Desarrollo de la solución

En los capítulos anteriores se ha mostrado los distintos puntos en los que se basa este trabajo, enfocándose principalmente en conceptos claves para la solución que se pretende entregar, conceptos tales cómo arquitectura de software principalmente arquitectura orientada a servicios, la herramienta Genexus, sus extensiones, y la reutilización.

En la presente etapa nos enfocaremos en la construcción de los patrones que serán aplicados a las transacciones. El patrón a diseñar debe crear los objetos necesarios para generar sistemas orientados a una arquitectura de software orientada a servicios, con esto se pretende generar sistemas con un alto grado de reusabilidad y los mencionados beneficios que conlleva el uso de una aplicación con una arquitectura orientada a servicios.

En este capítulo se mencionarán las formas para crear un nuevo patrón, se señalan los pasos y la información técnica para el desarrollo de estos, además se ahonda en temas como por ejemplo las herramientas necesarias para la creación de un nuevo patrón, la sintaxis relacionada con los patrones, el desarrollo de un ejemplo canónico para luego posteriormente enfocarnos en el desarrollo del patrón de este estudio.

3.2 Creación de un patrón Genexus

Hay dos formas posibles de crear un nuevo patrón en Genexus X, Una de ellas es definir y desarrollar todos los archivos necesarios de forma manual, otra forma de crear nuevos patrones es usar el asistente constructor de patrones (disponible a partir de Genexus X / Genexus Platform SDK Upgrade # 3) que simplifica la creación del patrón, las plantillas y los archivos XML, como base se puede tomar del patrón work with y editar las opciones, está última opción es la más fácil y sugerida por Artech.

3.3 Creación de forma manual

Si se desea desarrollar el patrón siguiendo esta opción, se debe seguir los siguientes pasos [21]:

Crear una carpeta con el nombre del patrón en la ruta: “Carpeta de instalación de Genexus”/Packages/Pattern, la nueva carpeta debe tener los siguientes archivos:

- Definición del patrón (archivo con extensión pattern.)
- Archivo de configuración
- <Empresa>.Patterns.<Nombre del patrón>.dll
- Archivos DKT

A continuación se puede ver un ejemplo paso a paso del desarrollo de un patrón de forma manual, de igual manera es necesario tener experiencia previa en el desarrollo de patrones Genexus y la construcción de DLLs si se opta por esta opción.

Primero: Escribir una descripción del patrón:

Escribir un patrón desde cero tiene mayor complejidad que tomar como base uno existente. Así que Artech recomienda que antes de empezar a desarrollar un patrón desde cero, revisar los patrones existentes y ver cual se acomoda más al patrón que se desea desarrollar. En la página de Artech existen algunos patrones que pueden ser tomados como ejemplos.

Segundo: Programar el ejemplo canónico:

Ejecutar Genexus y desarrollar el ejemplo canónico, tratar de captar la mayor cantidad de opiniones en pos de mejorar el producto.

Algunos puntos que se deben tener en cuenta al momento de desarrollar el patrón son:

- Cuando se desarrolla el ejemplo, siempre considerar que otras áreas puede el patrón cubrir.
- Tratar de usar funciones genéricas para facilitar la generación del patrón, por ejemplo en vez de usar DtoC (date to carácter) usar la función ToString() que aplica para todos los tipos de datos y no solamente fechas. Sino habría que hacer varios casos para distintos tipos de datos origen.
- Usar la mayor cantidad de componentes posibles, de esta forma habrá menos código que generar, esto simplifica el desarrollo de plantillas y además hace el código más legible a la hora de hacer upgrade al patrón.

Tercero: Definir el archivo de especificación del patrón

Una vez que se tenga el ejemplo canónico, este debe ser generalizado. En otras palabras, se necesita definir que objetos del patrón deben ser generados y que parámetros tendrá.

Existen dos tipos de parámetros:

- Parámetros de instancia: Estos son usados por el archivo de especificación de instancia.
- Parámetros generales: Existen parámetros que son aplicados a todos los archivos de instancia (ej, texto común, locación del gxchart) estos parámetros son definidos en el archivo de especificación de opciones.

En resumen esta tarea consiste en definir la estructura de estos archivos, para ello se necesita tener conocimientos en XML y amplio conocimiento del patrón a desarrollar.

Cuarto: Escribir los template (archivos DKT)

Esta es la parte más técnica del desarrollo de patrones. Los template son básicamente compuestos por salidas mezcladas con instrucciones de proceso. Las instrucciones de proceso son delimitadas por tags (<% y %>). En

particular para los template la salida es un archivo XPZ (archivo de exportación de Genexus). El archivo y las instrucciones de proceso son escritas en C#. En otras palabras esta es una mezcla de sentencias en C# con lenguaje Genexus, La parte variable del código se escribe con C#.

Quinto: Escribir el código de archivo "instance"

Es más simple crear un nuevo archivo instance con la mayoría de los valores ya existentes. Esta es también una parte altamente técnica, que por lo general requiere la lectura de la KB (usando las DLL proporcionadas por la herramienta de patrón) para llenar el archivo de la instancia.

El archivo instance es que le que tiene las opciones y las propiedades de los objetos que serán creados, tiene la estructura de un XML y al desplegarse en la herramienta se muestra como un árbol de opciones. Es uno de los archivos más importantes en la construcción de patrones.

3.4 Creación de un patrón con el SDK de Genexus

El SDK de Genexus es una herramienta desarrollada por Artech que provee las librerías necesarias para el desarrollo de patrones mediante la herramienta Visual Studio en las versiones 2005 y/o 2008 de Microsoft, también se le llama a esta herramienta PatternBuilder. Esta opción es la más recomendada, especialmente porque ayuda a compilar la dll, tiene elementos de autocompletación y los beneficios de desarrollar con un ID versus hacerlo en una herramienta como notepad.

Para la instalación del SDK es necesario tener instalado previamente Genexus evolutión 1 y alguna versión de Visual Studio ya sea 2008 o 2005, es altamente recomendable tener instalada la versión 2008 por sobre la versión 2005, básicamente porque los ejemplos que entrega Artech están desarrollados con esta versión, al tratar de correr los ejemplos en 2005, existirán problemas con la compatibilidad del framework con el cual fueron desarrollados los patrones, versus el framework que usa visual studio 2005. Respecto a cuál versión de visual studio 2008, no es necesario tener el entorno completo, con la versión xpress que es gratuita, es más que suficiente para el desarrollo de patrones.

Una vez que el SDK sea instalado exitosamente, se creará en la bandeja de visual studio un icono que permite la construcción de patrones y paquetes Genexus.

Fig. 10- SDK Genexus [IDE visual studio]

Si es que se usa el pattern builder para la construcción del patrón se puede basar en uno existente tales como el patrón workwith o Catalog que vienen el SDK de Genexus o realizar uno desde cero.

Para el caso de este estudio, se usará una opción intermedia, que significa construir desde cero y copiando desde los existentes sólo lo necesario del patrón workwith y no editar el patrón entero.

Al abrir el visual studio y crear un nuevo proyecto, aparecerá la opción “Genexus Pattern”, al seleccionarla se deben seguir los pasos que se indican a continuación:

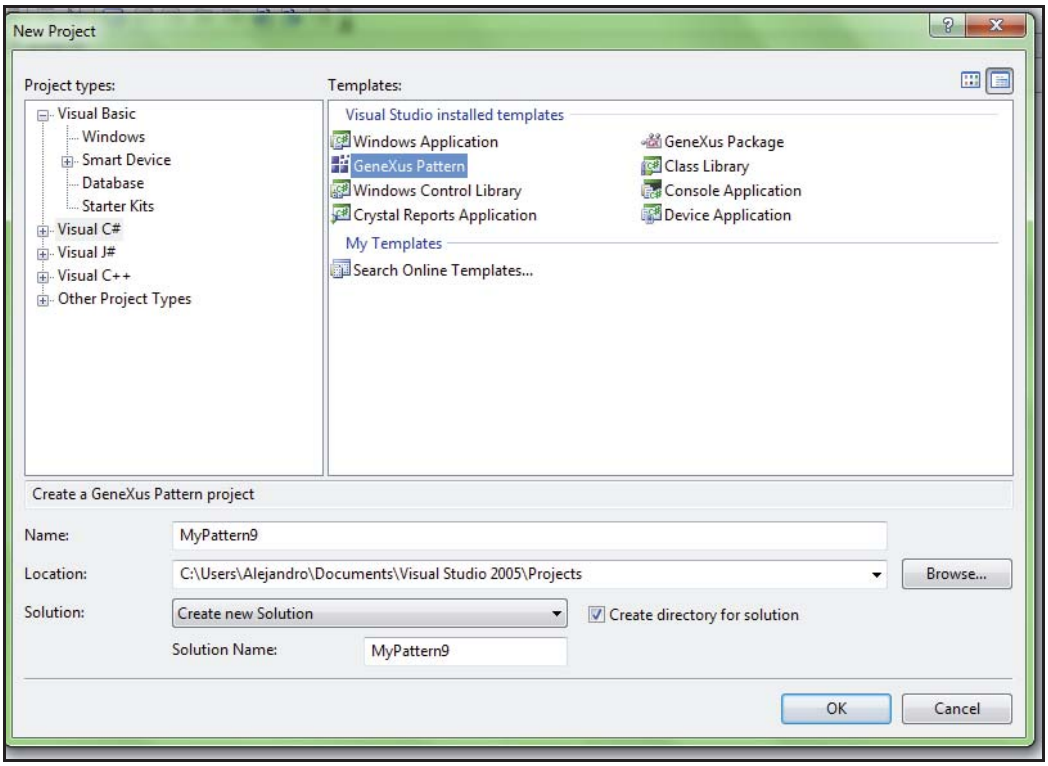
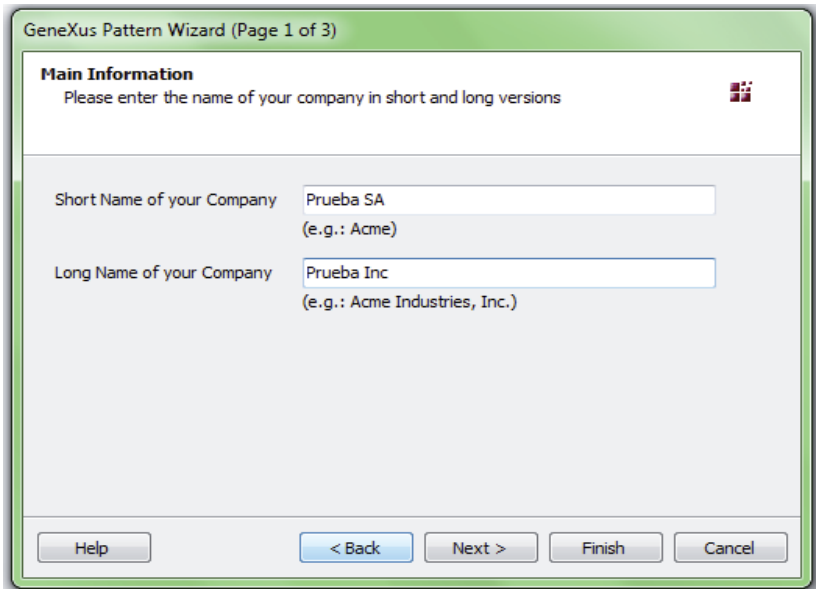


Fig. 11 - Paso 1 Creación patrón Genexus [IDE visual studio]

Al seleccionar la opción pedirá información tal como la empresa:



Seleccionar sobre que objetos se aplicará el patrón, por lo general en esta opción se selecciona la transacción al ser la que tiene los datos de la tabla en la base de datos.

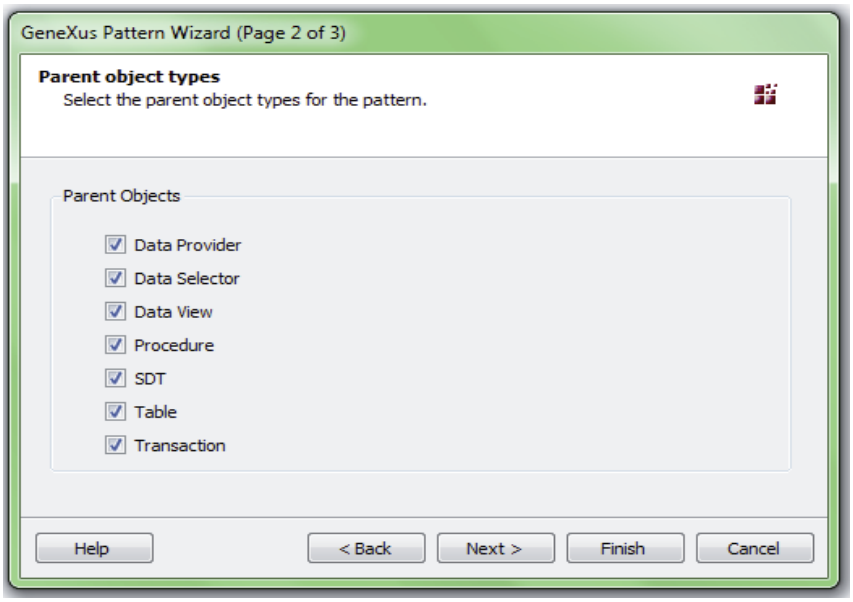


Fig. 12 - Paso 2 Creación patrón Genexus [IDE visual studio]

Seleccionar los tipos de objetos que se generarán al aplicar el patrón, con esto se crean los archivos con extensión DKT de ejemplo, los archivos DKT se explicarán adelante con más detalle.

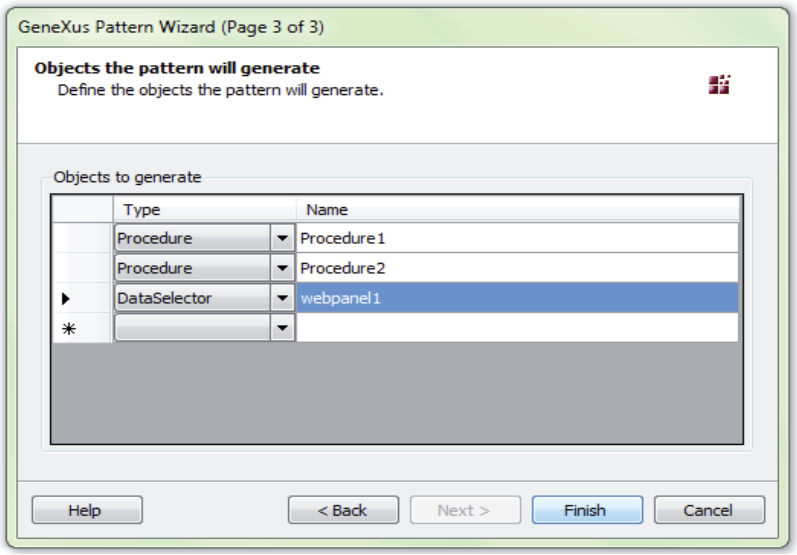


Fig. 13 - Paso 3 Creación patrón Genexus [IDE visual studio]

Luego al presionar "Finish" se creará un proyecto visual studio, con los archivos básicos para continuar la construcción del patrón.

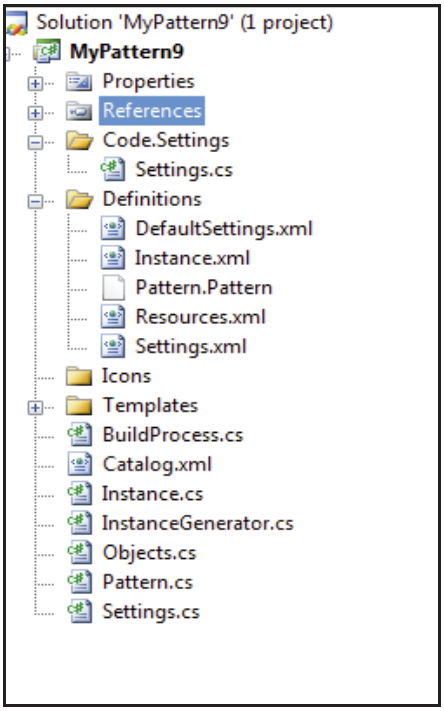


Fig. 14 - Paso 4 Creación patrón Genexus [IDE visual studio]

En la carpeta “Template” se encuentran los archivos DKT, cada uno de ellos representa el objeto y la parte del objeto, en Genexus la lógica de los objetos se divide en 5 partes:

- WebForm: Información del formulario web. Los procedimientos Genexus tienen un elemento layout para la creación de informes en remplazo de los webform.
- Rules: Se definen las reglas, se pueden hacer validaciones, pero principalmente se usa para la recepción de parámetros.
- Procedure: Código fuente de la aplicación.
- Conditions: Establece las condiciones de acceso a las tablas, esta lógica puede ser remplazada
- Variables: Se definen las variables que se usan en la aplicación, estas pueden ir en las pantallas, las reglas, el código fuente y en las condiciones.

Estos archivos tienen información básica que debe ser rellenada con la lógica para que cumpla con los requisitos del objetivo de este estudio.

Cuando se le da construir al proyecto, la utilidad “CodeGen” es ejecutada, esta se encarga de generar las clases necesarias para acceder a las instancias, esto puede ser visto en el output del visual studio, luego todos los archivos necesarios son copiados al directorio de Genexus, la ruta donde queda el resultado del patrón es la siguiente:

C:\Program Files\Artech\GeneXus\GeneXusXEv1\Packages\Patterns

La ruta variará en parte dependiendo de donde se instaló Genexus.

Posteriormente se ejecuta Genexus y el nuevo patrón aparecerá al seleccionar el objeto para el cual se asoció en las opciones (transacción), en “preferences/pattern” existirán las opciones del patrón, asociadas a la instancia por defecto del nuevo patrón.

A continuación se muestra una breve explicación sobre los archivos involucrados en la creación de un patrón Genexus:

Archivo pattern

Este archivo tiene la configuración del patrón, lleva por extensión “.pattern”, en él se definen los objetos que creará el patrón. La estructura de este archivo es muy similar a la de un XML, en él se manejan secciones, la sección que engloba todo el archivo es la sección “<Definition>”.

El nombre y el GUID del patrón debe ser únicos, estos se definen al principio del archivo.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<Pattern xmlns="http://schemas.Genexus.com/Patterns/Definition/v1.0"
                         Publisher="Araus"
                         Id="78CECEFE-BE7D-4980-9999-8D6E91FBA04B"
                         Name="PatternSOA"
                         Description="PatternSOA"
                         Version="1.1"
                         Icon="icons\PatternSOA.ico"
```

Luego sigue el nombre de la instancia del patrón, el cual es siempre el nombre del patrón seguido de los caracteres {0}:

```
<InstanceName>MyPattern9{0}</InstanceName>
```

En las siguientes secciones se ingresan los nombres de los archivos que ayudan a definir el patrón.

```
<InstanceName>PatternSOA{0}</InstanceName>
<InstanceSpecification>PatternSOAInstance.xml</InstanceSpecification>
<SettingsSpecification>PatternSOASettings.xml</SettingsSpecification>
<Implementation>Patterns.PatternSOA.dll</Implementation>
```

La función de estos 3 archivos se explicará más adelante.

El objeto asociado al patrón también queda definido en este archivo, de la siguiente manera:

```
<ParentObjects>
      <ParentObject Type="Transaction" />
</ParentObjects>
```

En este ejemplo se dice que el patrón será aplicado en los objetos transacción.

En la sección object se indican cuáles son los archivos DKT que se usaran y su ubicación.

```
<Object Type="Procedure" Id="busqueda" Name="Busqueda{Parent.Name}"
Element="instance/busqueda">
   <Part Type="Procedure" Template="templates\BusquedaProcedure.DKT" />
   <Part Type="Layout" Template="templates\BusquedaLayout.DKT" />
   <Part Type="Rules" Template="templates\BusquedaRules.DKT" />
   <Part Type="Conditions" Template="templates\BusquedaConditions.DKT" />
   <Part Type="Variables" Template="templates\BusquedaVariables.DKT" />
</Object>
```

Estos objetos son los que quedaran asociados a la transacción al aplicar el patrón, en el código de estos archivos DKT detalla la lógica de cada uno de los objetos generados, estos archivos no tienen sólo Tags como los XML, sino que también tienen código C# nativo.

Los atributos posibles son los siguientes:

- Tipo: El tipo de objeto que se va a generar pueden ser procedimientos, webpanel, transacciones, estructure data type, etc.
- Id: Identificador del elemento.
- Name: Nombre del objeto generado, por lo general es el nombre de la transacción más algún prefijo que tiene relación con el patrón.
- Description : Descripción del objeto generado.
- Element : El elemento que puede ser visualizado en la instancia.
- Part : Partes que conforman el objeto (events, condition, etc) además de la declaración del template usado en la creación de la parte.

La primera vez que se aplica el patrón se puede determinar que objetos importar, para eso se usa un archivo XML (el cual viene internamente en el archivo XPZ), esto es útil por ejemplo para importar el tema (archivo CSS) que se le aplicará al patrón o a una master page.

```
<Resources>
      <Resource Id="Resources" Version="0.8" File=
"Resources\WorkWithResources.xml"/>
      <Resource Id="ResourcesSDTheme" Version="0.5"
File="Resources\WorkWithResourcesSDTheme.xml"/>
            <Resource Id="ResourcesSD" Version="0.3"
File="Resources\WorkWithResourcesSD.xml"/></Resources>
```

Los atributos de esta instancia son los siguientes:

- Id: Identificador del recurso:
- Versión: Sirve para llevar un orden respecto de los avances en los recursos importados.
- File: Indica el archivo que se importa

Archivo instance

Es un archivo XML, el cual es referenciado en el archivo pattern, en él se definen los nodos que tendrá el patrón, estos nodos son las opciones personalizables del patrón, es un listado con forma de árbol que se despliega al seleccionar el patrón.

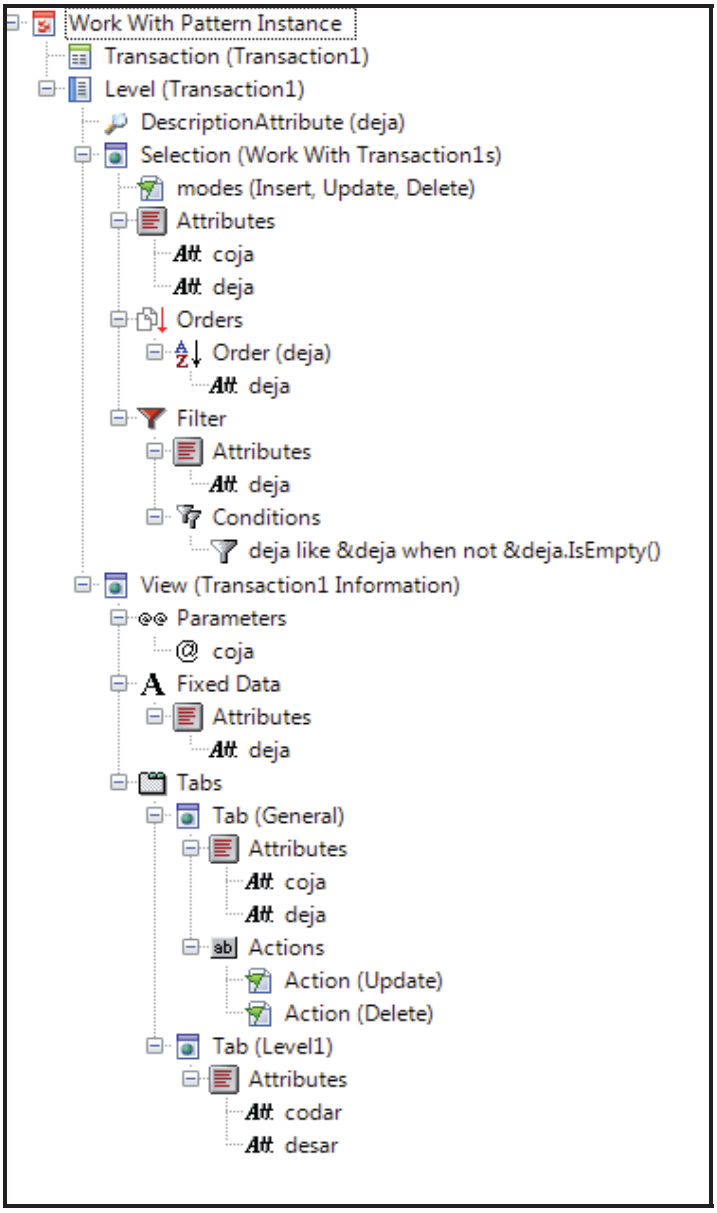


Fig. 15 - Despliegue archivo instance [IDE visual studio]

El nodo raíz en este ejemplo es “Work with pattern instance”, este es el primer nodo del patrón y es el padre de los demás nodos que se crean, la forma de definir el elemento raíz es la siguiente:

```
<Pattern
xmlns="http://schemas.Genexus.com/Patterns/InstanceSpecification/v1.0"
Name="Work With" Version="1.0" RootElement="instance" RootType="Instance">
```

Los atributos de esta definición son los siguientes:

- Name: Es el nombre del patrón, tal como en el archivo .pattern, debe ser un nombre único.

- Versión: Versión del patrón, se usa para mantener un orden respecto al manejo de versiones del patrón.
- RootElement: Indica cual es elemento “Root”, debe corresponder con uno de los elementos que se definirán más adelante en este archivo.
- RootType: El tipo del “RootElement”, debe corresponder con un elementType de los que se definirán más adelante en este archivo.
- HelperAssembly: Assembly .net que contiene las clases necesarias para generar y validar las instancias del patrón.
- Default generator: El nombre completo de la clase que se utiliza para generar una instancia predeterminada del patrón, Si no se especifica el usuario deberá rellenarlos manualmente.
- VersionAdaptar: Es opcional, se utiliza para convertir instancias de un patrón entre las versiones.
- Metadatamanager: Es opcional, Usada para mantener los metadatos asociados a la instancia.

Nodos ElementType

Estos describen un tipo de elemento que puede estar presente en la instancia. Cada elemento corresponde a un nodo de elemento XML en los archivos de instancia y a un nodo en la vista de árbol durante la edición de las propiedades de los patrones:

- Name : El nombre del ElementType, debe ser único.
- Caption y CaptionParameters: Controlan cómo se muestra el nodo en el editor.
- KeyAttribute: Un atributo puede ser seleccionado como indicador de "clave principal" de nodos de este tipo. Este valor es opcional.
- Validator: Esta clase se utiliza para validar el contenido de los nodos de este tipo. Debe estar presente en la Helper Assembly.
- Icono: Icono que se mostrará en el editor, al lado del caption.

Cada ElementType además incluye:

- Una lista de nodos “Atribute”, las cuales definen las propiedades de los elementos.
- Una lista de nodos hijos (“ChildElement”), corresponden a elementos subordinados del actual, se despliegan dentro del nodo padre (vista del tipo árbol).

- Una lista de nodos de acción, describen acciones personalizadas aplicables a los elementos de este tipo en el menú de árbol.

En la siguiente imagen se puede apreciar un Element Type que contiene Atributos y elementos hijos:

```
    <ElementType Name="busqueda" Caption="WebService de Busqueda"
Icon="icons\modes.ico">
      <Attributes>
        <Attribute Name="SDTOne" Type="string" Category="General"
Description="SDT tabla" DefaultValue="SDT({0})" />
      </Attributes>
      <ChildrenElements>
        <ChildElement Name="filter"     ElementType="Filter"
Multiple="false" Optional="true" />
        <ChildElement Name="attributes" ElementType="Attributes"
Multiple="false" Optional="true" />
        <ChildElement Name="parameters" ElementType="Parameters"
Multiple="false" Optional="true" />
      </ChildrenElements>
    </ElementType>
```

Nodos atributos

Los nodos atributos describen al elemento que aparece en el treeview del pattern, los atributos se visualizan en la sección de propiedades:

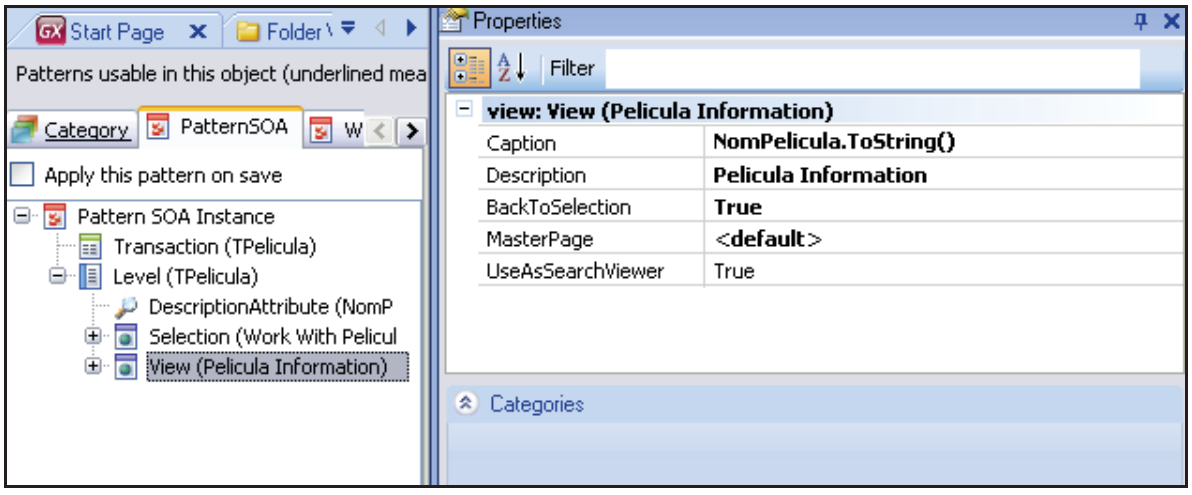


Fig. 16 - Propiedades objeto Genexus [IDE Genexus]

Los valores de las etiquetas de los treeview se pueden ver a continuación:

- Name: Nombre del atributo.

- Tipo: DataType del atributo, los valores posibles son string, text, int, bool, and enum (esto se presentan en forma de combo para selección por parte del usuario). Si un asterisco es incluido como una opción, entonces el usuario podrá ingresar un valor cualquiera. Text es similar a un string, sin embargo el usuario puede ser capaz de abrir un editor para ingresar múltiples líneas o texto largo.

- Category, Descriptión, PrettyName, Visible (Opcionales): Determina como el atributo se mostrará en la cuadricula de propiedades. Si PrettyName no es especificado, el atributo "name" es que el que aparecerá en la grilla.

- DefaultValue: Un valor por defecto, cuando el elemento que contiene el atributo se crea en blanco, el atributo tiene este valor.

- SerializationType: Establece como valor del atributo será almacenado en el archivo XML, los valores posibles son default, atribute, element, innertext, CDATA.

- ValidValues: El nombre de la clase que se usa para mostrar un combo box para permitir la selección de valores.

- GXLink: Indica que el atributo almacenará una referencia a un objeto Genexus, atributo o dominio.

- Not Null: Indica que un valor para el atributo es necesario.

Nodo ChildElement

Indica cuales elementos aparecerán como hijos del elemento, los valores que tiene este nodo son los siguientes.

- Nombre: Nombre del hijo.

- ElementType: Debe corresponder a un ElementType definido en el archivo Pattern.

- Multiple: Puede haber varios elementos de este tipo, es decir, corresponde a un elemento de una colección.

- Optional: El elemento es opcional, si es true: no se crea de forma predeterminada y puede ser eliminado. Los elementos marcados como múltiples se consideran siempre opcionales.

Nodo Action

Define una acción personalizada (en el menú de contexto) que puede ser ejecutado sobre elementos del type actual.

- Name: El nombre de la acción presentado en el contexto del menú.

- Class: La clase para ejecutar la acción debe estar presente en el HelperAssembly.

Archivo settings

Este archivo al igual que el archivo instance tiene una extensión XML, tiene la siguiente estructura de ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<Pattern Name="MyPattern9" Version="0.1.0" RootElement="Config"
         RootType="Config" xmlns="http://schemas.genexus.com/Patterns/InstanceSpecification/v1.0">
  <ElementTypes>
    <ElementType Name="Config" Caption="MyPattern9 Configuration" KeyAttribute=""  Icon="" ChildrenOrdered="Default">
      <Attributes />
      <ChildrenElements>
        <ChildElement Name="Names" ElementType="Names" Multiple="false" Optional="false" />
      </ChildrenElements>
    </ElementType>
    <ElementType Name="Names" Caption="Names" KeyAttribute="" Icon="" ChildrenOrdered="Default">
      <Attributes>
        <Attribute Name="procedure1" Type="string" Category="General" Description="" DefaultValue="&lt;Object&gt;Procedure1"  />
        <Attribute Name="procedure2" Type="string" Category="General" Description="" DefaultValue="&lt;Object&gt;Procedure2"  />
        <Attribute Name="webpanel1" Type="string" Category="General" Description="" DefaultValue="&lt;Object&gt;webpanel1"  />
      </Attributes>
      <ChildrenElements />
    </ElementType>
  </ElementTypes>
</Pattern>
```

Dentro de este se configura cada uno de los nodos que aparecerán en el Pattern settings de Genexus y sus correspondientes valores. Las Pattern settings son propiedades generales que se aplicarán para todas las instancias.

De la misma forma que el archivo instance.xml, está formado por el root node y los element type, donde cada uno de ellos tendrán definidos sus atributos y childElements (elementos hijos).

Lo que genera este archivo puede ser visto en preferences/patterns, el árbol que genera es similar al siguiente:

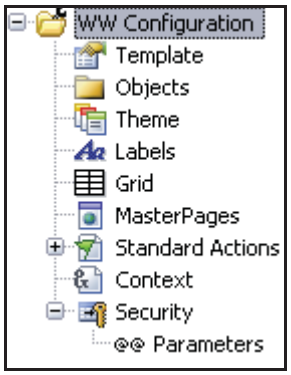


Fig. 17 - Preferencias patrón [IDE Genexus]

Pattern template (DKT)

Estos archivos son los que contienen el código que se ingresará en los distintos objetos generados por el patrón, tales como webpanels, grillas, procedimientos, sdt, dataproviders, etc. Estos son de mucha importancia, básicamente contienen el código fuente de los objetos Genexus.

Los template tienen código nativo C# mezclado con el código que se utiliza en GX:

```
<%@ Template Language="C#" TargetLanguage="GX" Description="View Rules" %>
<%@ Include Name="Constants.cs" %>
<%@ Assembly Name="Artech.Patterns.WorkWith" %>
<%@ Import Namespace="Artech.Patterns.WorkWith" %>
<%@ Property Name="Object" Type="Artech.Architecture.Common.Objects.KBObject" %>
<%@ Property Name="Part" Type="Artech.Architecture.Common.Objects.KBObjectPart" %>
<%@ Property Name="Instance" Type="Artech.Packages.Patterns.Objects.PatternInstance" %>
<%@ Property Name="Element" Type="Artech.Packages.Patterns.Objects.PatternInstanceElement" %>

<Part type="<%= PartType.Rules %>">
    <Source><![CDATA[<%

    WorkWithInstance wwInstance = WorkWithInstance.Load(Instance);
    ViewElement view = wwInstance.GetElement<ViewElement>(Element);
%>
<%= view.Parameters.RuleForView() %>
]]>
    </Source>
</Part>
```

Los templates son para generar el contenido de las partes de los objetos (event, rules, conditions, etc.), por lo mismo por estándar se denominan con el nombre del objeto y la parte del objeto.

En la sección de anexos se explican los archivos DKT con mayor detalle.

Pattern a desarrollar

Tal como se mencionó anteriormente para el patrón de este trabajo se desarrollará desde cero y se tomará como ejemplo el workwith para sacar ciertas partes de su código fuente, esto debido a la escasa información disponible sobre la construcción de patrones.

El patrón que será construido será aplicado a la transacción, pero no se usará la transacción para el manejo de información, la mayor complejidad que esto trae es el hecho que la lógica que traen intrínsecas las transacciones (valores por defecto, validaciones, cálculos de las formulas, manejo de información) debe ser apartada y desarrollada mediante webpanel y procedimientos.

Se debe recordar que la mayor parte de inteligencia que tiene Genexus es gracias a las transacciones, la cual se considera el objeto principal, este objeto es el encargado de definir la tabla de la base de datos y además contiene la lógica de negocio para el manejo de información. Las transacciones no se podrán usar en este trabajo básicamente porque toda la lógica viene incluida en un solo objeto Genexus (acceso y modificación) y el objetivo de este trabajo es el desarrollo de servicios web consumibles por aplicaciones desarrolladas en otros lenguajes, es decir no debe quedar atada sólo a Genexus, sino que a cualquier lenguaje que pueda consumir servicios web.

No se debe malentender el uso de las transacciones que se hará en este trabajo, el hecho que no se use como insumo para el cliente, no tiene relación con que no se use para la definición de los atributos, tablas y formulas, en ese

sentido las transacciones son irremplazables, lo que se hará es no usar la transacción como opción de uso para el usuario final, pero si para la definición de las tablas y sus atributos.

Los objetos que serán creados en este proyecto serán los siguientes:

- Servicio web para modificar: Este servicio web se usará para realizar las actualizaciones sobre las tablas, hará las validaciones necesarias antes de guardar la información.
- Servicio web para crear: Este servicio web se usará para realizar los ingresos sobre las tablas, hará las validaciones necesarias antes de guardar la información.
- Servicio web para eliminar: Este servicio web se usará para realizar las eliminaciones sobre las tablas, hará las validaciones necesarias antes de eliminar la información.
- Servicio web que retorna la información de determinadas tablas, para el retorno de información desde dos o más tablas se usarán los niveles de las transacciones.
- Programa encargado de verificar el acceso, recibirá un string de largo 200 y retornará true o false, este programa se usará para establecer el módulo de seguridad, no vale la pena hacerlo con usuario y password, porque el acceso variará dependiendo de cada empresa que use el patrón, por lo mismo deberá ser editado.
- Pantalla de selección de información: Pantalla que usará el servicio que retorna información y permitirá filtrarla.
- Pantalla de ingreso de información: Pantalla para el ingreso, modificación y eliminación de registros, consumirá los otros servicios, debe considerar la validación al ingresar los campos.

3.5 Consideraciones construcción de la solución

En la presente sección se muestra el código relevante de la aplicación como así también las consideraciones que se tomaron a la hora de construir el patrón.

Como primera instancia la construcción de este patrón consta de 1 archivo patterm. 4 archivos XML, 15 archivos DKT y 8 archivos CS.

Los archivos principales con los que cuenta el patrón son los siguientes:

- PatternSOA.Pattern
- PatternSOAInstance.xml

- PatternSOASettings.xml

El archivo instance declara los siguientes nodos con sus correspondientes hijos, Los hijos de esas opciones juntos con sus propiedades son las siguientes:

- Webservice de búsqueda
    - Filtros
        - Atributos
        - Condiciones
    - Atributos
    - Parámetros
- Webservice de ingreso de información.
    - Atributos
- Webservice de modificación de información
    - Filtros
        - Atributos
        - Condiciones
    - Atributos
- Webservice de eliminación de información
    - Filtros
        - Atributos
        - Condiciones

Dentro de los archivos DKT, tenemos los siguientes relacionados con el servicio de ingreso de información:

- IngresoConditions.dkt
- IngresoLayout.dkt
- IngresoProcedure.dkt

- IngresoRules.dkt
- IngresoVariables.dkt

Dentro de los archivos DKT, tenemos los siguientes relacionados con el servicio de modificación de información:

- ModificacionConditions.dkt
- ModificacionLayout.dkt
- ModificacionProcedure.dkt
- ModificacionRules.dkt
- ModificacionVariables.dkt

Dentro de los archivos DKT, tenemos los siguientes relacionados con el servicio de eliminación de información:

- EliminacionConditions.dkt
- EliminacionLayout.dkt
- EliminacionProcedure.dkt
- EliminacionRules.dkt
- EliminacionVariables.dkt

Dentro de los archivos DKT, tenemos los siguientes relacionados con el servicio de búsqueda de información:

- BusquedaConditions.dkt
- BusquedaLayout.dkt
- BusquedaProcedure.dkt
- BusquedaRules.dkt
- BusquedaVariables.dkt

Además de los archivos DKT utilizados en el patrón, existen los archivos CS que son los encargados de fundamentalmente de manejar las propiedades de los objetos, el archivo CS más importante de este patrón es el archivo workwithbuildprocess.cs en él se marca que los programas serán programas principales y el método por el

cual serán llamados es SOAP, las propiedades definidas en este archivo se aplican al momento de compilar la solución, las líneas de código que aplican las propiedades mencionadas son las siguientes:

```
//Establecer como principal
SetObjectsProperty(instanceObjects.GetGroup(PatternSOAObject.Selection),
Properties.WBP.MainProgram, true);

SetObjectsProperty(instanceObjects.GetGroup(PatternSOAObject.Busqueda),
Properties.PRC.MainProgram, true);

SetObjectsProperty(instanceObjects.GetGroup(PatternSOAObject.Ingreso),
Properties.PRC.MainProgram, true);

SetObjectsProperty(instanceObjects.GetGroup(PatternSOAObject.Eliminacion),
Properties.PRC.MainProgram, true);

SetObjectsProperty(instanceObjects.GetGroup(PatternSOAObject.Modificacion),
Properties.PRC.MainProgram, true);

//Establecer el metodo de llamada
SetObjectsProperty(instanceObjects.GetGroup(PatternSOAObject.Busqueda),
Properties.PRC.CallProtocol, Properties.PRC.CallProtocol_Values.Soap);

SetObjectsProperty(instanceObjects.GetGroup(PatternSOAObject.Ingreso),
Properties.PRC.CallProtocol, Properties.PRC.CallProtocol_Values.Soap);

SetObjectsProperty(instanceObjects.GetGroup(PatternSOAObject.Modificacion),
Properties.PRC.CallProtocol, Properties.PRC.CallProtocol_Values.Soap);

SetObjectsProperty(instanceObjects.GetGroup(PatternSOAObject.Eliminacion),
Properties.PRC.CallProtocol, Properties.PRC.CallProtocol_Values.Soap);
```

Capítulo 4 Discusión de resultados

4.1 Contexto de los resultados

En el presente trabajo se intentó entregar las bases para el desarrollo rápido de aplicaciones bajo una arquitectura orientada a servicios, para eso se mezclaron dos conceptos claves de dos formas de desarrollo distintas, como lo son la reutilización de conocimiento, pensamiento proveniente del desarrollo con Genexus y la arquitectura orientada a servicios.

La estructura de este trabajo se dividió en- tres partes fundamentales, el estudio de las arquitecturas de software, especialmente SOA, su aplicación en el contexto Genexus y la evaluación de los resultados.

Arquitecturas de software y mejoras del proceso de desarrollo de software

En esta fase se estudiaron las arquitecturas de software existentes en el mercado, especialmente para sentar las bases de la arquitectura orientada a servicios, para el conocimiento de este tema se revisaron papers y libros de la arquitectura orientada a servicios, en estos textos se plantean algunas definiciones de SOA muy similares, para los efectos de este trabajo nos quedaremos con la siguiente:

SOA es una arquitectura de aplicación en la cual todas las funciones se definen como servicios independientes con interfaces invocables bien definidos, que pueden ser llamadas en secuencias definidas para formar procesos de negocios.

Por servicios en la anterior definición, se refiere por lo general a servicios web, aunque este no es exclusivo, los servicios Web son aplicaciones que utilizan estándares para el transporte, codificación y protocolo de intercambio de información. Los servicios Web permiten la intercomunicación entre sistemas de cualquier plataforma y se utilizan en una gran variedad de escenarios de integración, tanto dentro de las organizaciones como con partners de negocios.

Otros elementos importantes dentro de la arquitectura SOA son los siguientes:

Broker: Su función es tomar mensajes de una cola, transformarlos y rutearlos hacia uno u varios sistemas dentro de una transacción de negocio. En él se encuentra el UDDI (Universal description discovery and integration), el cual es un directorio para encontrar servicios. Las ventajas de usar un bróker son las siguientes:

- Descubrimiento automático de los servicios existentes y su publicación en el repositorio.
- Detección de dependencias en la fase de ejecución entre los servicios y otros activos, y su publicación automática en el repositorio.

- Detección de consumidores de servicios, y su publicación automática en el repositorio.
- Publicación de información de alto nivel sobre el rendimiento de los servicios para ayudar a tomar decisiones de gobierno.
- Localización y notificación de diferencias detectadas en la información contenida en el repositorio y lo que existe realmente durante la ejecución (por ejemplo, notificación al administrador si el WSDL del servicio que se está ejecutando en el contenedor no coincide con el que está almacenado en el repositorio).
- Publicación en el repositorio de información sobre eventos relevantes para el gobierno (como son las violaciones de políticas o de los acuerdos a nivel de servicio).

Ventajas de SOA

La arquitectura SOA es una arquitectura que permite organizar mucho mejor los sistemas IT de una compañía. Esta arquitectura aporta ventajas muy destacables como:

- Escalabilidad
- Robustez
- Homogeneidad
- Aplicar lógica de middleware pudiendo implementar procesos de negocio y seguridad.
- Reutilización de componentes
- Permitir una mayor integración de sistemas y lograr tiempos de desarrollo más cortos usando sistemas estándar no propietarios de comunicación
- Tolerancia a cambios de diseño e infraestructura.
- Interacción dinámica entre socios de negocio.
- Sopor te para sistemas heterogéneos.
- Mejorar la flexibilidad de los sistemas.

Genexus

Genexus es una herramienta case desarrollado por Artech, cuyo objetivo es asistir al analista y a los usuarios en todo el ciclo de vida de las aplicaciones.

La idea básica de Genexus es automatizar todo aquello que es automatizable:

- Normalización de datos y diseño.
- Generación y mantenimiento de la base de datos.
- Programas de aplicación.

Con esto se pretende que el analista evite dedicarse a tareas rutinarias y tediosas, permitiéndole poner su atención en entender los problemas del usuario.

Características de 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 soporta las plataformas y lenguajes líderes y los DBMS más populares (SQL server, my sql, DB2, Oracle, postgres)
- Genexus y los productos Genexus proporcionan un set de herramientas para el desarrollo de aplicaciones y soluciones críticas de negocios.
- Los productos Genexus constituyen una suite de herramientas complementarias que permiten emprender desarrollos frecuentes y tareas de gestión de datos como workflow (GXflow), reportes (GXquery), data warehousing (GXplorer) y construcción de portales e integración de aplicaciones online (GXportal).

Diseño basado en el conocimiento	• Diseño de aplicaciones a partir de las visiones de los usuarios. • Diseño totalmente automático del modelo de datos. • Diseño y prototipo independientes de la plataforma.
Desarrollo automático	• Generación 100% automática de las aplicaciones (base de datos y programas). • Generación y mantenimiento automático de la documentación de aplicaciones.
Mantenimiento automático	• Mantenimiento 100% automático (base de datos y programas). • Migración automática de los datos a las nuevas estructuras.
Desarrollo e implementación multiplataforma	• Java/J2EE, .NET y .NET Compact Framework, Ruby, Cliente/Servidor, AS/400, iSeries, System i, Smart Devices.

Tabla 3 - Características Genexus

Las aplicaciones en Genexus se crean a partir de sus objetos, los más relevantes son los siguientes:

- **Transacciones**: Una transacción es un proceso interactivo o pantalla (Win o Web) que permite a los usuarios crear, modificar o eliminar información de la base de datos.

- **Reportes:** Un reporte es un proceso que permite visualizar los datos de la base de datos. La salida del listado puede ser enviada a pantalla o a la impresora (y con ello tenemos un listado convencional).

- **Procedimientos:** Este objeto tiene todas las características de los Reportes, y además permite actualizar la base de datos.

- **Work Panels:** Es una pantalla Windows que permite al usuario realizar consultas interactivas a la base de datos.

- **Web Panels:** Son similares al grupo de Work Panels pero requieren un navegador de aplicaciones (Browser) para ser ejecutados en ambientes Internet/Intranet/Extranet.

Patrones Genexus:

Dentro de las funcionalidades de Genexus, existe la de crear objetos y funcionalidades a partir de una estructura básica (transacción), esta funcionalidad son los llamados patrones Genexus.

Es a partir de ellos que se puede generar la mayor parte de un proyecto informático sin tener que escribir una sola línea de código, en el mercado actual existen patrones gratis y patrones en venta, los cuales brindan diferentes funcionalidades enfocándose siempre en la actualización de información de las tablas que componen el sistema.

Trabajo realizado

Genexus provee una herramienta para ampliar el IDE y crear nuevos patrones, en el presente trabajo se estudió la escasa información sobre esta funcionalidad y se implementó un patrón que permitiera crear los objetos necesarios para enfocar el desarrollo en una arquitectura SOA, logrando con esto un alto grado de reutilización de las funcionalidades implementadas, además de la promoción de la integración.

En este estudio se realizó la creación de un patrón aprovechando las funcionalidades que brinda el SDK de Genexus, con ello se crearon cuatro servicios de manera declarativa:

- Servicio para la búsqueda de información: Se le ingresan los parámetros para la búsqueda y los campos a retornar, en la definición se pueden retornar campos que estén relacionados con la tabla gracias a las bondades de la sentencia for each de Genexus.

- Servicio de ingreso de información: Se crea un servicio que realiza el insert en la base de datos, en caso de que el registro ya existe, el servicio retorna un mensaje indicando tal situación, en una próxima versión se incluirán características como validación de campos.

- Servicio de modificación de información: Se crea un servicio que realiza el “update” en la base de datos en caso de que no encuentre registros el servicio entrega un mensaje indicando la situación.

- Servicio de modificación de información: Se crea un servicio que realiza el “update” en la base de datos en caso de que no encuentre registros el servicio entrega un mensaje indicando la situación.

Tal como se recomienda en el desarrollo de patrones de la guía de Artech, se desarrollará el ejemplo canónico, es decir se programarán los objetos que debe generar el patrón de una forma tradicional, es decir se crearán los objetos manualmente mediante programación tradicional, esto servirá para definir cuáles son los objetos y la lógica que el patrón debe generar.

El ejemplo canónico que se usará será una tabla Clientes, la cual tendrá la siguiente información:

- Código
- Nombre del cliente
- Apellido
- Nombre del país
- Nombre de la ciudad
- Dirección

Se crearán 4 servicios, estos servicios tendrán la lógica para ingresar, modificar, eliminar y el resultado de información.

El modelo de datos del ejemplo es el siguiente:

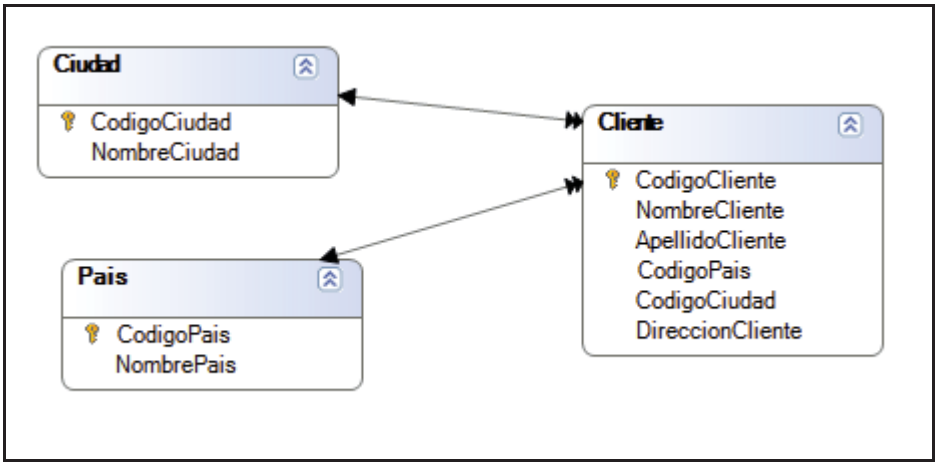


Fig. 18 - Simple modelo de datos de ejemplo

Búsqueda

El código fuente de la búsqueda se puede ver a continuación:

```
For each
      Where CodigoCliente = 1
      &SDTClienteItem.CodigoCliente = CodigoCliente
      &SDTClienteItem.NombreCliente = NombreCliente
      &SDTClienteItem.ApellidoCliente = ApellidoCliente
      &SDTClienteItem.NombrePais = NombrePais
      &SDTClienteItem.DireccionCliente = DireccionCliente

      &SDTCliente.add(&SDTClienteItem )
      &SDTClienteItem = new()
EndFor
```

Para efectos del patrón debe ser posible:

- Ingresar las condiciones de acceso a la tabla
- Retornar información de más de una tabla a la vez.
- Crear automáticamente una estructura dinámica que será la que finalmente retornará el servicio.

Ingreso

El código fuente del ingreso se puede ver a continuación:

```
New
      CodigoCliente     = &CodigoCliente
      NombreCliente     = &NombreCliente
      ApellidoCliente   = &ApellidoCliente
      CodigoPais        = &CodigoPais
      CodigoCiudad      = &CodigoCiudad
      DireccionCliente  = &DireccionCliente
EndNew
```

Lo relevante en esta opción es la recepción de parámetros, los cuales pueden ser variables y el ingreso de información a la base de datos.

Modificación

El código fuente de la modificación se puede ver a continuación:

```
      For each
            Where CodigoCliente = 1
                  CodigoCliente     =  &CodigoCliente
                  NombreCliente     =  &NombreCliente
                  ApellidoCliente   =  &ApellidoCliente
                  DireccionCliente  =  &DireccionCliente
```

```
When None
      &MensajeVal = "No se encontraron registros"
EndFor
```

Para efectos del patrón debe ser posible:

- Ingresar las condiciones de acceso a la tabla
- Ingresar los parámetros que la opción debe recibir/actualizar.

Eliminación

El código fuente de dicha eliminación es el siguiente:

```
For each
      Where CodigoCliente = 1
            Delete
EndFor
```

Lo relevante en esta opción es la recepción de parámetros y el ingreso en la base de datos, esta opción deberá recibir como parámetros las variables consideradas en las condiciones de acceso.

En el siguiente capítulo se verá cómo aplicar el patrón y como generar el código fuente que se expuso en esta sección.

4.2 Caso de estudio

En este capítulo se evaluará la aplicación del patrón Genexus en un caso de estudio relacionado a un trabajador que pertenece a una empresa y a un determinado cargo, el modelo de datos de este trabajo será el siguiente:

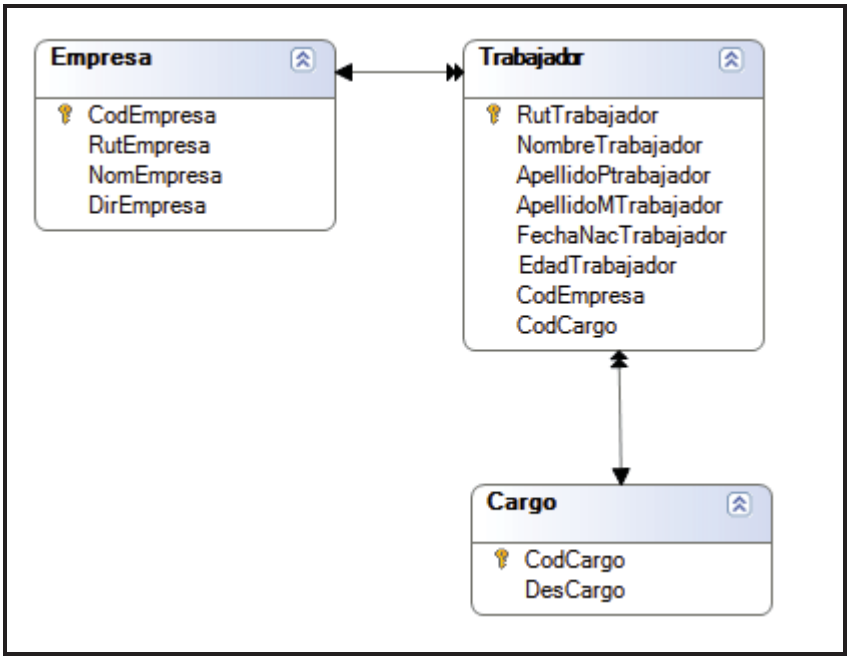


Fig. 19 - Simple modelo de datos ejemplo

Estas tablas son generadas a partir de las transacciones Genexus, para este caso de estudio es interesante retornar la información del trabajador, pero no es demasiado funcional retornar el código de la empresa o el código del cargo, por lo cual se retornará el nombre de la empresa y el nombre del cargo, adicionalmente se retornará la edad del trabajador. La transacción que retornará esta información tendrá la siguiente estructura:

Name	Type	Description	Formula
Trabajador	Trabajador	Trabajador	
RutTrabajador	Numeric(9.0)	Rut Trabajador	
NombreTrabajador	Character(20)	Nombre Trabajador	
ApellidoPtrabajador	Character(20)	Apellido Ptrabajador	
ApellidoMTrabajador	Character(20)	Apellido MTrabajador	
CodCargo	Numeric(4.0)	Cod Cargo	
DesCargo	Character(20)	Des Cargo	
CodEmpresa	Numeric(10.0)	Cod Empresa	
NomEmpresa	Character(100)	Nom Empresa	
FechaNacTrabajador	Date	Fecha Nac Trabajador	
EdadTrabajador	Numeric(4.0)	Edad Trabajador	age(FechaNacTrabajador)

Fig. 20- Transacción trabajador ejemplo [IDE Genexus]

Notar que la transacción tiene los campos “DesCargo” y “NomEmpresa” estos se usan para poder obtener la información relacionada al trabajador, no afectan la tabla, ya que Genexus infiere que esos campos pertenecen a otra tabla mediante los campos CodEmpresa y CodCargo.

Para la edad del trabajador se utilizó una fórmula que retorna los años que han pasado desde su fecha de nacimiento hasta el día de hoy.

4.2.1 Búsqueda de registros

El primer ejemplo será aplicar el patrón de búsqueda, para ello se debe aplicar el patrón creado, de la forma que se ve en la Figura 21.

Lo que hace este patrón es retornar un conjunto de registros, para este caso retornará sólo uno dado que consultaremos por la clave primaria, es decir el Rut.

La definición aplicada se traduce en lo siguiente:

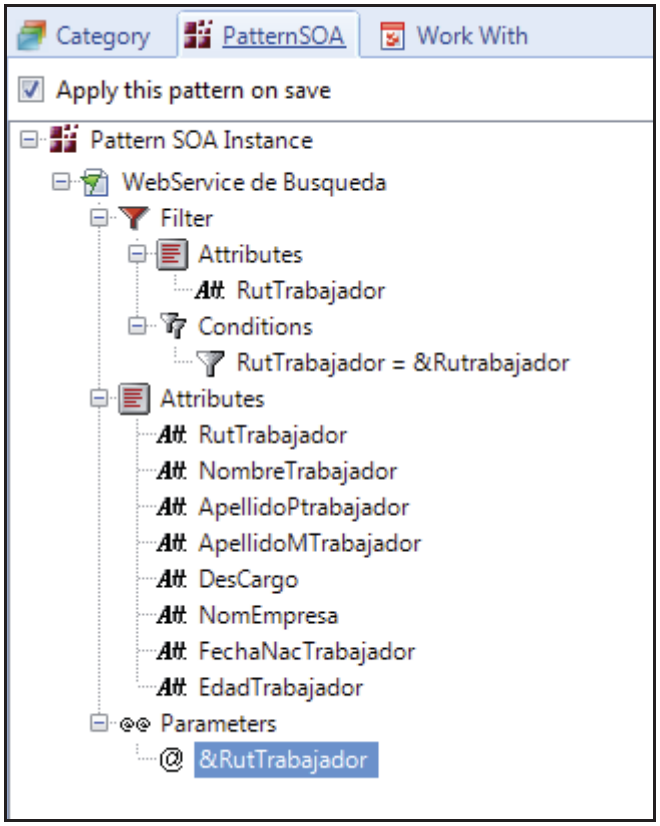


Fig. 22 - Despliegue ejemplo instance [IDE Genexus]

Filtros:

- RutTrabajador = &RutTrabajador

Parámetros:

- Variable RutTrabajador.

Campos a retornar:

- RutTrabajador
- NombreTrabajador
- ApellidoPTrabajador
- ApellidoMTrabajador
- DesCargo
- NomEmpresa
- FechaNacTrabajador
- EdadTrabajador

El programa generado por Genexus y expuesto como servicio es el siguiente:

Reglas:

```
parm(&RutTrabajador , &SDTTrabajador);
```

Condiciones:

```
RutTrabajador = &RutTrabajador;
```

Código fuente:

```
For each
      &SDTTrabajadorItem.RutTrabajador          = RutTrabajador
      &SDTTrabajadorItem.NombreTrabajador       = NombreTrabajador
      &SDTTrabajadorItem.ApellidoPtrabajador    = ApellidoPtrabajador
      &SDTTrabajadorItem.ApellidoMTrabajador    = ApellidoMTrabajador
      &SDTTrabajadorItem.DesCargo               = DesCargo
      &SDTTrabajadorItem.NomEmpresa             = NomEmpresa
      &SDTTrabajadorItem.FechaNacTrabajador     = FechaNacTrabajador
      &SDTTrabajadorItem.EdadTrabajador         = EdadTrabajador

      &SDTTrabajador.add(&SDTTrabajadorItem )
      &SDTTrabajadorItem = new()

EndFor
```

Al revisar el webservice con la herramienta de microsoft “Web service studio 2.0”, se entregan los datos de entrada y los métodos que incluye el servicio:

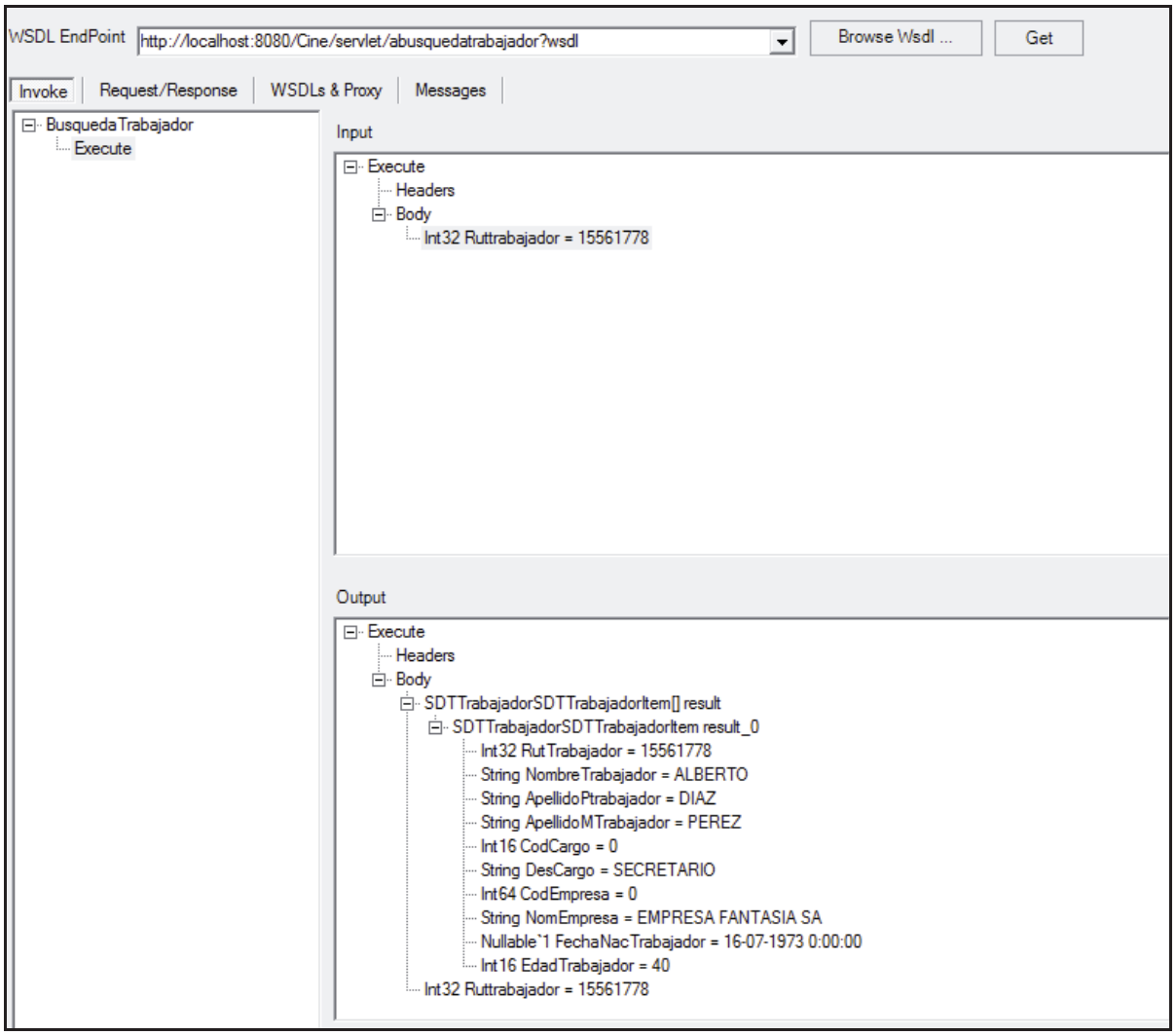


Fig. 23 - Servicio web búsqueda [WebService Studio]

Al revisar el servicio se puede ver que tiene el método “execute” recibe como parámetro el Rut del trabajador y entrega la información del trabajador, en el ejemplo se usaron datos ficticios en el que se envía como Rut 15561778, notar que para los campos de cargo y empresa retorna los datos que salen de una tabla distinta a la de trabajadores, por lo que entregar información que involucre varias tablas estaría cubierta, esto se logra gracias a que Genexus puede sacar todos los datos de una tabla si es que la tabla que estamos recorriendo tiene la clave primaria de otra.

4.2.2 Ingreso de registros

Para el ingreso de información se puede crear un servicio mediante patrón creado en este estudio, la definición de dicho servicio se puede ver en la siguiente imagen:

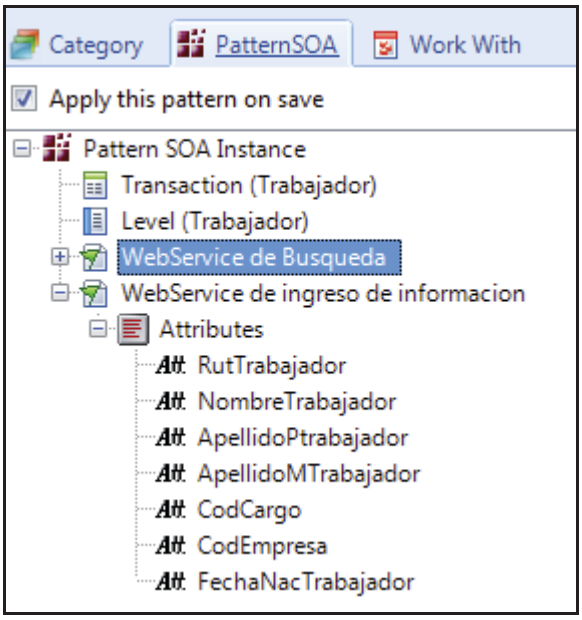


Fig. 24 - Aplicando el patrón ingreso [IDE Genexus]

Como se aprecia en la imagen, para el ingreso de información se debe seleccionar los campos que necesita que se ingresaran al insertar los registros, estos campos serán a su vez tomados como datos de entrada.

El código generado por el patrón será el siguiente:

Reglas:

```
parm(&RutTrabajador , &NombreTrabajador , &ApellidoPtrabajador ,
&ApellidoMTrabajador , &CodCargo , &CodEmpresa , &FechaNacTrabajador);
```

Condiciones:

No aplican.

Código fuente:

```
New
      RutTrabajador           = &RutTrabajador
      NombreTrabajador        = &NombreTrabajador
      ApellidoPtrabajador     = &ApellidoPtrabajador
```

```
        ApellidoMTrabajador     = &ApellidoMTrabajador
        CodCargo                = &CodCargo
        CodEmpresa              = &CodEmpresa
        FechaNacTrabajador      = &FechaNacTrabajador

EndNew
```

Al revisar el webservice con la herramienta de microsoft “Web service studio 2.0”, se entregan los datos de entrada y los métodos que incluye el servicio:

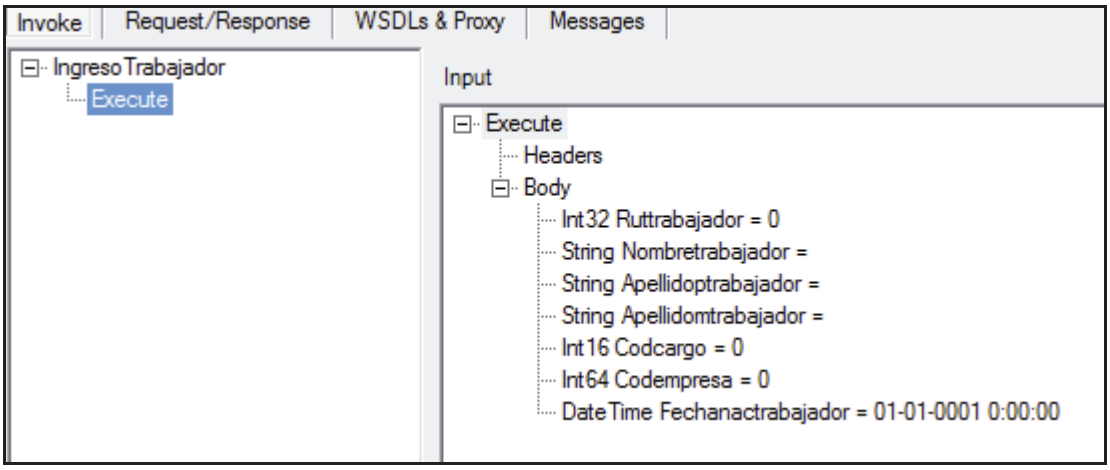


Fig. 25 - Servicio web ingreso [VisualService Studio]

Al igual que en el servicio anterior este servicio recibe como entrada los datos definidos en el patrón, el método que tiene se llama execute.

Quedará para una nueva versión la inclusión de validaciones al momento de ingresar información.

4.2.3 Eliminación de registros

La definición para el servicio de eliminación es en base a parámetros los cuales sirven para la definición de las condiciones de eliminación, a continuación se muestra un ejemplo de cómo definir la eliminación:

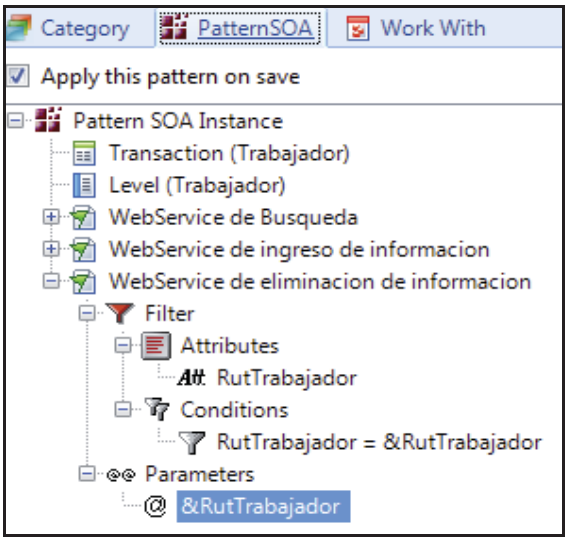


Fig. 26 - Aplicando patrón eliminar [IDE Genexus]

El código fuente generado por el patrón es el siguiente:

Reglas:

```
parm(&RutTrabajador , &MensajeVal);
```

Condiciones:

```
RutTrabajador = &RutTrabajador;
```

Procedimiento:

```
For each
      defined by RutTrabajador
            Delete
      When None
            &MensajeVal = "No se encontraron registros"
EndFor
```

Al revisar el procedimiento con la herramienta de microsoft entrega la siguiente información:

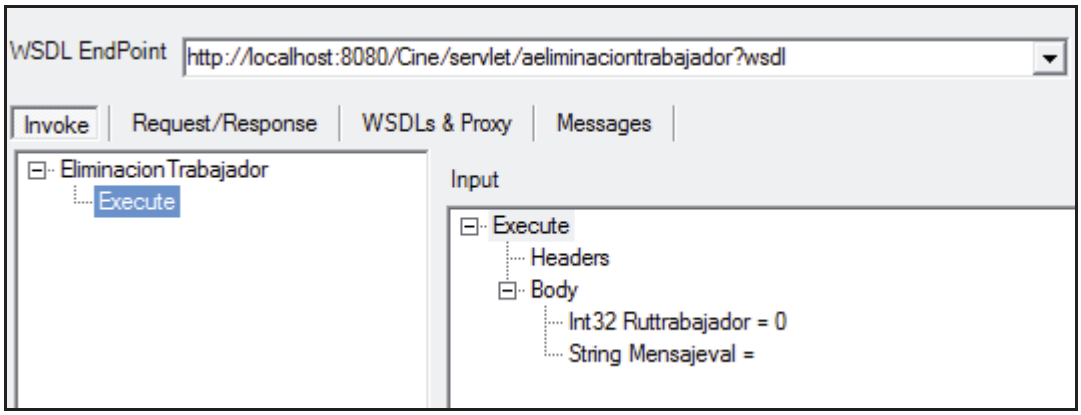


Fig. 27 - Servicio web eliminar [VisualService Studio]

Si la base de datos está con integridad referencial, no permitirá eliminar, si el servicio no encuentra ningún registro, retornará un mensaje indicando la situación.

4.2.4 Modificación de registros

Para la modificación de la información se usará patrón de modificación de información, en él se definen los parámetros, las condiciones para acceder a los registros a actualizar y los campos a actualizar, como restricción deben haber una concordancia entre lo que se recibe y los campos a actualizar, dentro de los campos que se reciben también deben ir los que se considerarán en las condiciones (si es que estos son variables).

La siguiente imagen muestra cómo se define el ejemplo típico acá descrito.

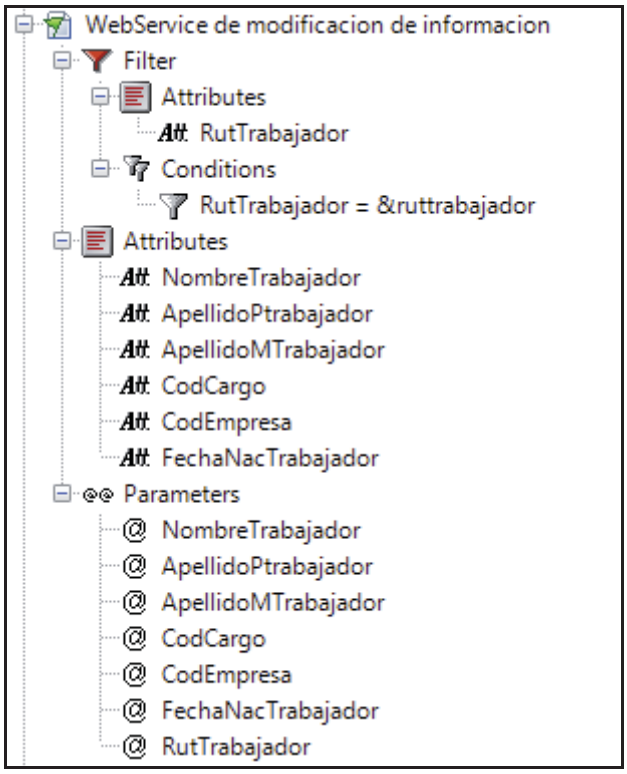


Fig. 28 - Aplicando patrón modificar [IDE Genexus]

El código generado por la aplicación es el siguiente:

Reglas:

```
parm(&NombreTrabajador , &ApellidoPtrabajador , &ApellidoMTrabajador ,
&CodCargo , &CodEmpresa , &FechaNacTrabajador , &RutTrabajador , out:
&MensajeVal);
```

Condiciones:

```
RutTrabajador = &ruttrabajador;
```

Código fuente:

```
For each
      NombreTrabajador        =  &NombreTrabajador
      ApellidoPtrabajador     =  &ApellidoPtrabajador
      ApellidoMTrabajador     =  &ApellidoMTrabajador
      CodCargo                =  &CodCargo
      CodEmpresa              =  &CodEmpresa
      FechaNacTrabajador      =  &FechaNacTrabajador
When None
      &MensajeVal = "No se encontraron registros"
```

```
EndFor
```

Al revisar el servicio con la herramienta de Microsoft podemos ver su información en cuanto a métodos y parámetros de entrada:

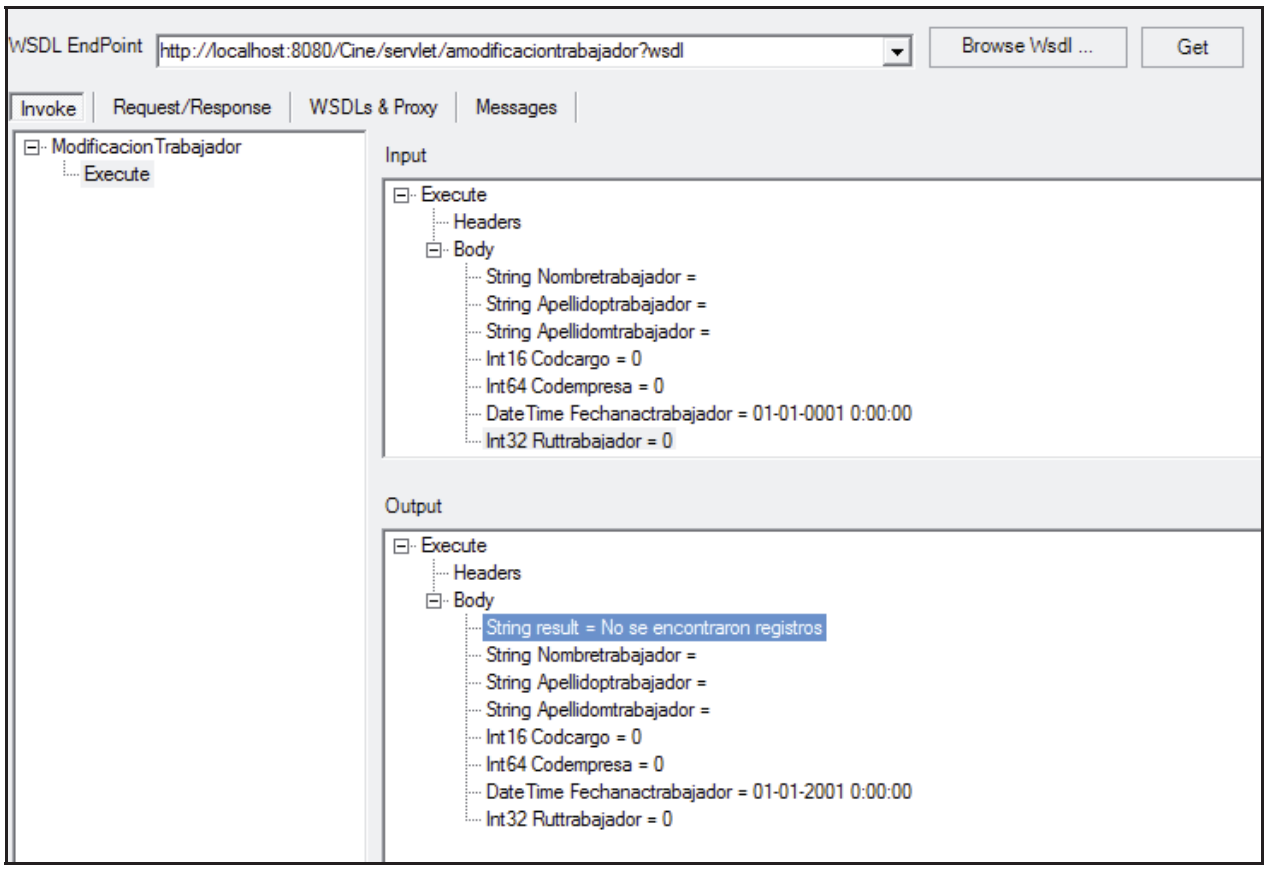


Fig. 29 - Web service modificar [VisualService Studio]

En el caso del ejemplo, el servicio fue invocado con todos sus campos en 0 y no retorno registros, por lo mismo el resultado fue “No se encontraron registros”.

4.3 Resultados

Los resultados que se muestran a continuación son extraídos de pruebas con distintos usuarios del patrón desarrollado, para estas pruebas se tomaron personas con experiencia en el uso de la herramienta Genexus.

La aplicación de los patrones una vez que ya se tienen construidas las tablas toma en promedio 20 minutos, este resultado contrarresta con el desarrollo a mano de los 4 servicios, el cual se estima en unos 120 minutos. A esto se le debe sumar la bondad de poder modificar todos los servicios creados por el componente de una sola vez, con lo cual se ven reducidos también los tiempos destinados a mantención.

Además de los beneficios nombrados, se obtienen beneficios propios del uso de patrones como son la estandarización de los objetos creados y la actualización de todos ellos en caso de no haber sido modificados a mano.

Respecto de los beneficios de haber usado este patrón versus haber ocupado otro patrón Genexus en el cual no se crean servicios, las ventajas van dadas por los beneficios propios de SOA los cuales han sido bastante nombrados en este trabajo, pero en resumen son:

- La reutilización de componentes: Los servicios pueden ser usados por distintas aplicaciones escritas en distintos lenguajes.
- Abstracción. Los servicios son autónomos y se accede a ellos a través de un contrato formal lo que provee desacople y abstracción.
- Facilita el testeo, los componentes que ya fueron probados no es necesario probarlos cada vez que se reutilizan, si ya se sabe cómo operan y que información ingresa/retorna.
- Seguridad, permite definir distintos niveles de seguridad, además de proveer un servicio que solo hace lo que tiene que hacer, a diferencia de cuando se da acceso a las bases de datos, en cuyo caso un error por parte del desarrollador podría borrar tablas completa.
- Favorece el monitoreo, al estar el acceso a los datos centrado se puede hacer un mejor monitoreo de los tiempos de carga o de solicitud del servicio.
- Fácil escalabilidad, en caso de querer hacer una modificación está se realiza en el servicio y no en todas las aplicaciones que lo usan, ejemplo obtener información de una nueva tabla.

4.3.1 Rúbrica resultados

A continuación se muestra una tabla en la cual se evalúan 5 aspectos y se indican los criterios de cumplimiento aceptable y no aceptable de cada aspecto.

	No cumple	**Medianamente cumple**	**Cumple**
Usabilidad	Un usuario con experiencia en el uso de patrones, tarda más de 20 minutos en el primer intento de aplicar el	El usuario con experiencia en el uso de patrones tarda entre 15-20 minutos en el primer intento de aplicar el	El usuario con experiencia en el uso de patrones tarda menos de 15 minutos en el primer intento de aplicar el

	nuevo patrón.	nuevo patrón.	nuevo patrón.
Reutilización	Fracasa en el intento de usar el componente creado, fuera del ambiente Genexus.	Es posible usar el componente con Genexus y con el lenguaje con el que fue generado (.net o java).	Es posible usar el componente independiente del lenguaje con el que la aplicación consumidora está creada.
Facilidad de testeo	Componente nuevo, nunca antes testeado, se debe testear de nuevo.	Componente existente, modificado levemente para adaptarlo. Debe ser probado nuevamente, específicamente en las modificaciones realizadas.	Componente existente, probablemente en ambiente de producción. Utilizado sin modificaciones, no necesita que sea testeado nuevamente.
Seguridad	No aplica ninguna configuración de seguridad	Permite implementar seguridad mediante desarrollo tradicional	Incluye distintas opciones de seguridad a elección.
Escalabilidad	Modificación de los componentes uno a uno.	Corrección de todos los servicios a la vez	Ofrecer opción de seleccionar los servicios construidos, que desean modificarse.

Tabla 4 - Tabla posibles resultados

4.3.2 Resultados asociados al patrón

Dada la rúbrica con los aspectos, se aplicará en el patrón desarrollado, para ver el grado de cumplimiento de los aspectos mencionados.

Patrón desarrollado	
Usabilidad	Cumple: El patrón desarrollado, se aplica igual que los patrones Genexus existentes, las opciones tienen asociadas ventanas explicativas, en caso de que el usuario no entienda alguna función.

Reutilización	Cumple: El componente puede ser usado por cualquier lenguaje que ofrezca la posibilidad de consumir servicios web.
Facilidad de testeo	Cumple: El componente una vez que se crea, solo debe ser usado, ya está probado por la aplicación que lo puso en producción.
Seguridad	Medianamente cumple: El patrón ofrece la posibilidad de programar las opciones de seguridad de manera tradicional, Ejemplo: un procedimiento interno que retorna si es posible continuar.
Escalabilidad	Medianamente cumple: Si se modifica el patrón, se modifican todos los componentes (servicios) creados por el patrón.

Tabla 5 - Resultados del patrón

Capítulo 5 Conclusiones y trabajo futuro

5.1 Conclusiones

La principal motivación para realizar este estudio es la poca reutilización de código que Genexus brinda, esto se debe a que la herramienta opera con un enfoque orientado a la reutilización del conocimiento, con la cual se promueve el desarrollo rápido basado en generación automática de código.

Para aumentar el grado de reusabilidad de las opciones construidas con esta herramienta se optó por buscar una solución a este tema mediante un enfoque más tradicional, como lo son la arquitectura de software y la creación de un componente de software que apoye dicha arquitectura.

La baja reutilización que presenta Genexus se debe principalmente a la poca separación de las capas que existe, esto queda de manifiesto especialmente cuando se ocupan los objetos transacción, dado que la capa de presentación, la capa de datos y el controlador son mantenidos en este mismo objeto.

En una primera instancia se estudió la posibilidad de ver la mejor forma de aplicar una arquitectura de tres capas mezcladas con los beneficios y objetos de Genexus, este estudio pudo entregar beneficios al desarrollo con Genexus, básicamente porque bastaría con cambiar el origen de los datos, y con esto algunas opciones quedarían en un alto porcentaje construidas automáticamente, lo cual ayudaría a la reutilización (debido a la separación de capas) y a la disminución en el tiempo de desarrollo, cabe destacar que al ser Genexus un lenguaje orientado a evento, el uso de una capa controlador en una capa distinta era imposible o poco significativo. Investigando el tema de las arquitecturas de software, es que se decide abordar esta tesis en un contexto distinto, utilizando una arquitectura de software más reciente, más independiente y multiplataforma, considerando que hoy en día las tecnologías móviles tienen gran relevancia en el mundo de la informática y el uso de servicios web puede ser usado desde dicha plataforma.

Como se mencionó anteriormente la arquitectura de software es fundamental en el desarrollo de software, cada sistema tiene una forma en que sus componentes se organizan, pero lo relevante es tener conciencia de esa arquitectura y de la forma en la que se quieren organizar los componentes. La arquitectura debe ser consistente a lo largo del desarrollo, al seguir la arquitectura al pie de la letra se logran sistemas más escalables, lo cual a la larga ahorra tiempo en el desarrollo de mantenciones, cabe recordar que un alto porcentaje del tiempo del proyecto se ocupa en mantenciones, es por eso que este ítem no debe ser tomado a la ligera.

SOA entrega muchos beneficios a los desarrollos de sistemas, quizás la más significativa sea el hecho que es multiplataforma, el libro de introducción a SOA nombra tres problemas que existen en las empresas respecto al crecimiento de sus sistemas de software:

- Sistemas antiguos heredados.
- Sistemas redundantes e inaccesibles entre sí.
- Sistemas con múltiples integraciones “punto a punto”.

Al empezar a desarrollar sistemas con arquitectura SOA estos problemas mencionados tienden a desaparecer ya que se centraliza el acceso a los datos a través de los servicios web.

SOA está pensando para fomentar la comunicación entre sistemas, con esto se evita tener sistemas “islas” y tener integraciones punto a punto, además de centralizar el acceso a los datos, entrega beneficios en cuanto al tiempo de desarrollo de nuevas aplicaciones ya que la lógica de almacenamiento, la seguridad de acceso a los datos y el manejo de la concurrencia, estaría disponible para ser consumido si es que ya fue desarrollado para una opción previa.

Respecto de lo anterior un beneficio tangible es que entrega este tipo de proyectos, es que en caso de que una determinada empresa tenga muchos proveedores que ocupan distintas tecnologías, estos se podrán comunicar, sin tener la necesidad de construir de nuevo las aplicaciones, con esto se podría reducir el costo de los sistemas que son construidos, ya que para muchas opciones bastaría con hacer la pantalla que consuma el servicio, que fue disponibilizado.

Genexus es una gran herramienta, su lenguaje orientado a eventos no es difícil de entender, aunque a cualquiera que viene de una idea de desarrollo en la que se usa el lenguaje de consultas de consultas estructurado (SQL) puede resultarle tedioso en el principio, con el tiempo Artech ha incluido funcionalidades automáticas interesantes como el uso de Ajax, patrones y la posibilidad de incorporar extensiones y user controls, haciendo el IDE más completo y escalable.

Una de las mayores dificultades a la hora de enfrentar este tipo de proyectos, es que se tiene mucha ambición respecto a lo que se desea construir, se pretende que todo sea ajustable y parametrizable, básicamente se quiere que se configuren un par de opciones y el 80% del sistema esté construido, lamentablemente la verdad es que si bien los patrones ayudan a la construcción de sistemas, aún debe hacérsele un pequeño retoque a los programas que genera para que se comporten según lo deseado, esto es especialmente verdad cuando no se quieren usar transacciones, las cuales generan muchas funcionalidades automáticamente, tales como validaciones, cálculos e ingreso de información. En este patrón para almacenar en la base de datos se usan servicios web, lo cual hace que el sistema sea un poco más rígido y se deban configuran un par de opciones más.

Otra dificultad que vale la pena mencionar es la escasa documentación existente, Artech entrega el código fuente del patrón base y sobre ese se debe averiguar en base a prueba y error como funciona y como se podría adaptar a las necesidades de un proyecto de esta índole.

5.2 Trabajo futuro

Dentro del trabajo futuro del presente estudio está incluir más opciones a la hora de establecer las reglas del patrón, como lo son por ejemplo nuevas validaciones y valores de inicialización del patrón, con esto se logrará que el usuario pueda aplicar el patrón con el mínimo esfuerzo.

Otra tarea importante es publicar el patrón en la página de Artech junto con el código, para que pueda ser usado y mejorado por la comunidad Artech, de igual manera se debería crear un segundo patrón a partir de este, pero que en vez de crear los servicios, provea las funcionalidades de forma "local", con ello se podrán desarrollar aplicaciones con el mismo patrón sin tener que publicar los servicios, una vez que se haga esta tarea se debe poder crear webpanels a través del patrón, para ello se debe tomar como base los patrones existentes que crean llamativas interfaces.

En este trabajo se entregaron las directrices para que alguien pueda crear su propio patrón e igual se entregan las instrucciones como usar el patrón, como trabajo futuro se podría explicar el código en aún más detalle, mezclando lo que sería un manual de usuario de la aplicación del patrón, con las partes de código que permiten las funcionalidades, con esto quedará más claro cómo se desarrolló el patrón y que partes modificar en caso de que alguien quiera intervenirlo.

Referencias

[1] Reynoso, B. (2005). *Arquitectura para distribución y agregación*. Última consulta en Julio 2013, desde http://download.microsoft.com/download/4/F/F/4FF88340-43CC-4C5B-8E50-09002969D0DD/20051129-ARC-BA.ppt

[2] Castagnet, N. (2006). *Software Factories para construir Sistemas de Información con Genexus*. Tesis de maestría no publicada, Universidad de la República, Montevideo, Uruguay.

[3] Microsoft (2006). *La Arquitectura Orientada a Servicios (SOA) de Microsoft aplicada al mundo real*. Última consulta en Julio 2013, desde http://download.microsoft.com/download/c/2/c/c2ce8a3a-b4df-4a12-ba18-7e050aef3364/070717-Real_World_SOA.pdf

[4] Matsumura, M., Brauel, B., & Shah, J. (2009). *SOA adoption for dummies*. Hoboken, NJ: Wiley Publishing inc.

[5] Herzum, P., & Sims, O. (2000). *Business component factory: A comprehensive overview of component-based development for the enterprise*. New York: John Wiley.

[6] Wolfinger, R., Dhungana, D., Prähofer, H., & Mössenböck, H. (2006). *A Component Plug-In Architecture for the .NET Platform*. Lecture Notes in Computer Science, 4228, 287-305.

[7] Wolfinger, R. (2008). *A Plug-in Architecture and Design Guidelines for Customizable Enterprise Applications*. OOPSLA 2008 Doctoral Symposium. Nashville, TN, EEUU, Octubre. OOPSLA Companion '08, 893-894.

[8] Broy, M., Szyperski, C., Henn, J., Koskimies, K., Plasil, F., Pomberger, G., Pree, W. (1998). *What characterizes a (software) component?*. Última consulta Abril 2014, desde http://www.softwareresearch.net/fileadmin/src/docs/publications/J010.pdf

[9] Tanenbaum, A. S. (2009). *Sistemas operativos modernos* (3th ed.). Amsterdam, Holanda: Pearson Education.

[10] Pressman, R. S. (2001). *Software engineering: A practitioner's approach (5th ed.)*. New York: McGraw-Hill.

[11] Sommerville, I. (2002). *Ingeniería de software (7th ed.)*. Londres, Inglaterra: Addison-Wesley.

[12] Gonda, B., & Jodal, N. (2007). *Filosofía y fundamentos teóricos de Genexus*. Última consulta en Julio 2013, desde http://www.Genexus.com/files/Desarrollo-basado-en-conocimiento.pdf?es

[13] Mili, H., Mili, F. & Mili, A. (1995). *Reusing Software: Issues and Research Directions*. USA: IEEE Transactions on Software Engineering, 21, 528-561.

[14] Gamma, E. (1995). *Design patterns: Elements of reusable object-oriented software*. Reading, Mass: Addison-Wesley.

[15] Clements, P. (1996). *A survey of architecture description languages*. 8th Workshop on Software Specification and Design, Schloss Velen, Alemania. 8th International Workshop on Software Specification and Design, 16-25.

[16] Perry, D. E., & Wolf, A. L. (1992). *Foundations for the study of software architecture*. ACM Sigsoft Software Engineering Notes, 17, 40-52.

[17] Bass, L., Clements, P., & Kazman, R. (1998). *Software architecture in practice*. Reading, Mass: Addison-Wesley.

[18] Garlan, D., & Shaw, M. (1993). *An Introduction to Software Architecture*. Technical ReportCS, Carnegie Mellon University, School of Computer Science, 94-166.

[19] Brown, A. W., & Wallnau, K. C. (1998). *The Current State of CBSE*. IEEE Software, 15, 37-56.

[20] Artech (2012). *Visión general de Genexus*. Última Consulta en Julio de 2013, desde http://www.Genexus.com/files/wp-vision-general?es

[21] Artech (2012). *Documentación patrones*. Última Consulta Julio de 2013, desde http://wiki.gxtechnical.com/commwiki/servlet/hwiki?Category%3APatterns

[22] Silverira, L. (2010). *Diseño e implementación de un motor de reglas dinámicas usando especificaciones GeneXus*. Tesis de maestría no publicada, Universidad de la República, Montevideo, Uruguay.

[23] Microsoft (2005). *Introducción a visual studio*. Última Consulta en Julio 2013, desde http://msdn.microsoft.com/es-es/library/fx6bk1f4%28v=vs.80%29.aspx

[24] Alexander, C., Ishikawa, S., & Silverstein, M. (1977). *A pattern language: Towns, buildings, construction*. New York: Oxford University Press.

[25] Montilva, J., Arapé, N., Colmenares, J. (2010). *Desarrollo basado en componentes*. IV Congreso de automatización y control, Mérida, Venezuela, Noviembre, (paper).

[26] Weitzenfeld, A. (2005). *Ingeniería de software orientada a objetos con UML, Java e Internet*. México: Thomson.

[27] Artech (2012). Características de Genexus. Última Consulta en Abril de 2014, desde

http://www.genexus.com/productos/genexus-2012/genexus-home?es

[28] Earl, T. (2009). *SOA design patterns*. Upper Saddle River, NJ: Prentice Hall.

[29] Delgado, A., González, L. Piedrabuena F. (2005). *Desarrollo de aplicaciones con enfoque SOA (Service Oriented Architecture)*. Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento, Puebla, Mexico, Febrero, (paper).

[30] Campderrich, F. B. (2003). *Ingeniería del software*. Barcelona: Universitat Oberta de Catalunya.

[31] Sametinger, J. (1997). *Software engineering with reusable components*. Berlin: Springer.

[32] Sodhi, J., & Sodhi, P. (1999). *Software reuse: Domain analysis and design processes*. New York: McGraw-Hill.

[33] Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*.Boston: Addison-Wesley.

Anexos

Anexo 1: Glosario

Arquitectura de software: Definición la estructura general de un sistema y varía de acuerdo con el tipo de sistema a desarrollarse. Así puede estar basada en elementos sencillos o componentes prefabricados de mayor tamaño [26].

API: Interface de Programación de Aplicaciones (Application Programming Interface), es un conjunto de funciones, procedimientos, métodos que un sistema operativo, biblioteca o servicio provee para poder interactuar [22].

Base de Conocimiento: Base de datos que contiene la metadata GeneXus

Business Component: es una propiedad de los objetos Transacción GeneXus que permite que dichos objetos se ejecuten sin interfaz de usuario.

Case: (Ingeniería del Software Asistida por Computadora) comprende un amplio abanico de diferentes tipos de programas que se utilizan para ayudar a las actividades del proceso del software [11].

DLL: Acrónimo de Bibliotecas de Enlace Dinámico (Dynamic Linking Library), término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda del programa por parte del sistema operativo [22].

Formulario: Conjunto de elementos que componen una página, incluye controles que se utilizan para mostrar/pedir información, acciones que se le permiten realizar al usuario, definición de accesos a otras páginas y referencias a programas de inicialización.

Genexus: Es una herramienta inteligente para crear, desarrollar y mantener, en forma automática, aplicaciones multiplataforma de misión crítica que se adaptan fácilmente a los cambios del negocio y a las nuevas posibilidades brindadas por la evolución tecnológica [27].

IDE: Acrónimo de entorno de desarrollo integrado (Integrated Development Environment); aplicación que brinda facilidades a los programadores para el desarrollo de software.

Java: Lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90.

Patrón Genexus: Funcionalidad de Genexus que permite crear objetos de forma declarativa, mediante el llenado de opciones en vez de programación tradicional.

Programación orientada a Objetos: La Programación orientada a objetos (Object-Oriented Programming) es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas.

Plantillas DKT: Plantilla para generar un archivo de texto. Se utiliza para generar un XML que define una parte de un objeto GeneXus, como por ejemplo, las reglas, el código fuente o la definición de variables.

Procedimiento: Tipo de objeto en GeneXus que permite escribir código para leer o escribir información en la Base de Datos, o hacer otro tipo de procesamiento.

Software Factory: Línea de Productos de software que configura herramientas extensibles, procesos, y contenidos utilizando una plantilla de Software Factory, basada en un esquema de Software Factory, para automatizar el desarrollo y mantenimiento de variantes de un producto arquetípico adaptando, ensamblando, y configurando componentes basados en Frameworks.

SOA: Acrónimo de Arquitectura orientada a servicios (Service Oriented Architecture); Arquitectura de Software basado en la definición de servicios reutilizables, con interfaces públicas bien definidas, donde los proveedores y consumidores de servicios interactúan en forma desacoplada para realizar los procesos de negocio Establece un conjunto de principios y metodologías para el diseño y desarrollo de software en forma de interoperabilidad de servicios [29].

SQL: Acrónimo de lenguaje de consulta estructurado (Structures Query Language); Lenguaje para acceder a bases de datos relacionales.

Transacción: Objeto GeneXus que brinda información de los datos de la aplicación y cómo el usuario va a acceder a la aplicación para realizar inserciones, lecturas, modificaciones y borrado de datos.

UML: Acrónimo de Unified Modeling Language (Lenguaje Unificado de Modelado), lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

URL: Acronimo para Uniform Resource Identifier, localizador de recursos uniforme. se usa para nombrar recursos en Internet para su localización o identificación

Visual studio .Net: Plataforma de desarrollo de software respaldada y desarrollada por Microsoft

WebPanel: Objeto GeneXus utilizado para definir páginas Web.

Workflow: Flujo de Trabajo. Se refiere al flujo de trabajo a seguir para la consecución de una tarea o trabajo predeterminado.

XML: Acrónimo de lenguaje de marcado extensible (extensible Markup Language); estándar creado por la W3C, utilizado para almacenar datos en forma legible.

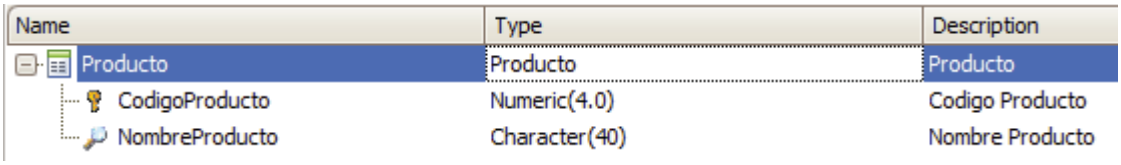
Anexo 2 Patrón category

Artech en su sitio [21] entrega directrices para el desarrollo de un patrón, tomando como ejemplo el patrón category, al ser una tecnología no muy popular a nivel mundial no existen libros para el desarrollo de patrones, por lo que estas guías son el único instrumento para el desarrollo de patrones, a continuación se entrega una explicación del desarrollo de patrones, útil para hacer un paralelo con el patrón desarrollado en este trabajo.

Este patrón sirve para agregar categorías, es muy útil cuando se desea categorizar productos, por ejemplo, tenemos un producto y deseamos saber a qué tipo o rama pertenece.

Ejemplo canónico:

Supongamos que tenemos una tabla de productos cuya transacción es la siguiente:



Al aplicar este patrón, se generarán los siguientes objetos:

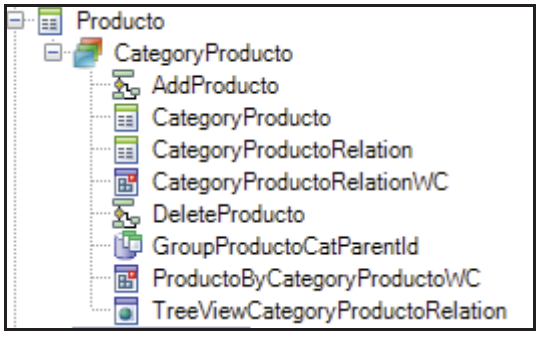
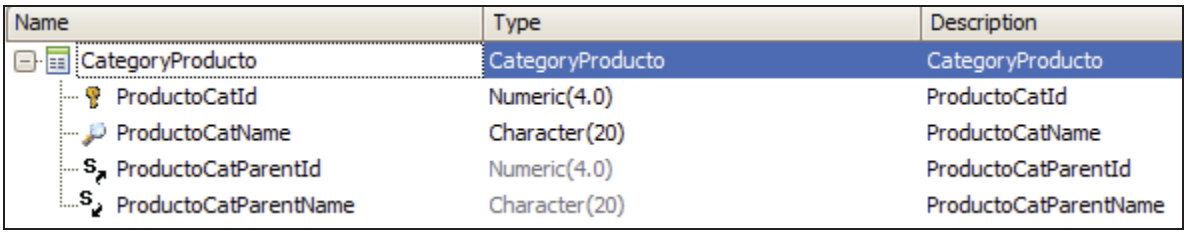


Fig. 30 - Objetos generados por el patrón Category

Dentro de ellos destacan dos tablas adicionales, encargadas de almacenar la categoría del producto:

CategoryProducto

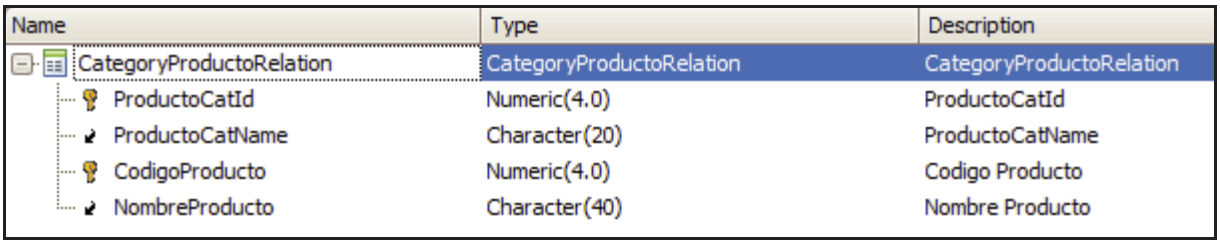


Name	Type	Description
CategoryProducto	CategoryProducto	CategoryProducto
ProductoCatId	Numeric(4.0)	ProductoCatId
ProductoCatName	Character(20)	ProductoCatName
ProductoCatParentId	Numeric(4.0)	ProductoCatParentId
ProductoCatParentName	Character(20)	ProductoCatParentName

Fig. 31 - Transacción patrón category producto

Esta primera tabla almacenará el nombre de la categoría.

CategoryProductoRelacion



Name	Type	Description
CategoryProductoRelation	CategoryProductoRelation	CategoryProductoRelation
ProductoCatId	Numeric(4.0)	ProductoCatId
ProductoCatName	Character(20)	ProductoCatName
CodigoProducto	Numeric(4.0)	Codigo Producto
NombreProducto	Character(40)	Nombre Producto

Fig. 32-Transacción patrón category

Esta segunda tabla almacenará la relación del producto con la categoría.

Los otros objetos generados son los siguientes:

WebPanel TreeViewCategoryProductoRelation: Muestra una vista de árbol de cada categoría con sus elementos asignados.

DataProvider ProductoCatalog: Implementado para cargar la vista de árbol en el evento de inicio del Web Panel TreeViewCategoryProductoRelation.

Web Component ItemByCategoryProductoWC: Este componente Web muestra los elementos asignados a una categoría en el Web Panel TreeViewCategoryProductoRelation

Web Component CategoryProductoRelationWC y Procedimientos DeleteItem, AddItem - Le permite añadir elementos a una categoría o eliminarlos de una categoría.

El despliegue en el navegador de este patrón se muestra a continuación:

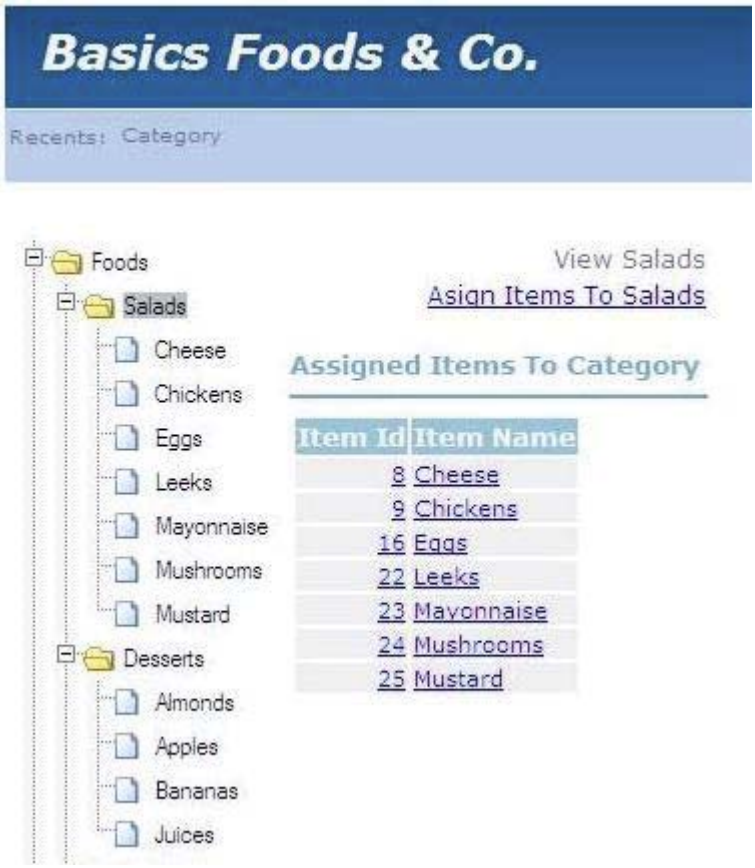


Fig. 33 - Patrón category aplicado

Al hacer clic en el link “Asing ítem to Salads” se muestra la siguiente imagen:

Fig. 34 - Patrón category aplicado

El archivo pattern es el principal archivo de configuración, en este archivo se define el patrón que se desea generar. En él se agregan los archivos instance.xml y settings.xml

Ejemplo:

```
<InstanceName>Category{0}</InstanceName>
<InstanceSpecification>CategoryInstance.xml</InstanceSpecification>
<SettingsSpecification>CategorySettings.xml</SettingsSpecification>
<Implementation>Artech.Patterns.Category.dll</Implementation>
<AutoUpdate>true</AutoUpdate>
```

Después de esto hay que definir los objetos padres, estos son los objetos donde el patrón debe ser generado.

```
<ParentObjects>
      <ParentObjectType="Transaction">
      </ParentObject>
</ParentObjects>
```

Incluso también se puede definir el "DefaultSetting.xml" donde se determina que objetos serán importados la primera vez que el patrón es aplicado.

La parte principal de este archivo consiste en definir los objetos que serán generados por el patrón;

Se puede crear cualquier tipo de objetos desde atributos, subtipos, transacciones, procedimientos, webpanel, etc.

```
<Object Type="Transaction"Id="CategoryRelationTrn"Name="{Element.categoryItem}"Description="
{Element.categoryItem}"Element="instance/transactionsName" >
      <Part Type="Structure"Template="Templates\CategoryRelationStructure.dkt" />
      <Part Type="WebForm"Template="Templates\CategoryRelationWebForm.dkt" />
      <Part Type="Rules"Template="Templates\CategoryRelationRules.dkt" />
      <Part Type="Variables"Template="Templates\CategoryRelationVariables.dkt" />
</Object>
```

El elemento object tiene los siguientes atributos:

- Type : El tipo del elemento a generar
- Id : Identificador del elemento.
- Nombre : Nombre del objeto a generar.
- Descripción : Descripción del objeto a generar.
- Elemento : Opcional, si se deja en blanco se usa el elemento Root, si no se encuentra el archivo indicado, no se generar el objeto.

- Template : El archivo DKT usado para generar el objeto.
- Creando el archivo template .DKT (Ejemplo CategoryStructure.DKT)

Si el objeto tiene múltiples partes se necesita especificar la parte que se desea definir y asignar los otras partes con el valor por defecto.

```
<Object Type="Transaction"Id="CategoryRelationTrn"Name="{Element.categoryItem}"Description="
{Element.categoryItem}"Element="instance/transactionsName" >
      <Part Type="Structure"Template="Templates\CategoryRelationStructure.dkt" />
      <Part Type="WebForm"Template="Templates\CategoryRelationWebForm.dkt" />
      <Part Type="Rules"Template="Templates\CategoryRelationRules.dkt" />
      <Part Type="Variables"Template="Templates\CategoryRelationVariables.dkt" />
</Object>
```

Nuevamente la etiqueta “part” tiene los siguientes atributos:

- Tipo : El tipo de la parte que se generará (webform, eventos, reglas, variables).
- Template: El archivo DKT que genera esta parte.

Estructura del archivo DKT

Por defecto, todos los archivos DKT reciben el elemento que se selecciona en los atributos del elemento. Si no se especifica el elemento, se recibe el elemento raíz.

Ejemplo del archivo DKT:

```
<%@ Template Language="C#" TargetLanguage="GX" Description="Category Trn" %>
<%@ Assembly Name="Artech.Patterns.Category" %>
<%@ Import Namespace="Artech.Patterns.Category" %>
<%@ Property Name="Object" Type="Artech.Architecture.Common.Objects.KBObject" %>
<%@ Property Name="Part" Type="Artech.Architecture.Common.Objects.KBObjectPart" %>
<%@ Property Name="Instance" Type="Artech.Packages.Patterns.Objects.PatternInstance" %>
<%@ Property Name="Element" Type="Artech.Packages.Patterns.Objects.PatternInstanceElement" %>

<%    CategoryInstance catInstance = new CategoryInstance(Instance);%>
<Part type="<%= PartType.Structure %>">
      <Level Name="<%= catInstance.TransactionsName.Category %>" Type="<%= catInstance.TransactionsName.Category %>" Description="<%=
catInstance.TransactionsName.Category %>" >
            <Attribute key="True"><%= catInstance.AttributesName.CategoryId %></Attribute>
            <Attribute key="False"><%= catInstance.AttributesName.CategoryName %></Attribute>
            <Attribute key="False" isNullable="True"><%= catInstance.AttributesName.CategoryParentId %></Attribute>
            <Attribute key="False"><%= catInstance.AttributesName.CategoryParentName %></Attribute>
      </Level>
</Part>
```

En este ejemplo se define la estructura de una nueva transacción usando los valores seleccionados por el usuario en CategoryId, CategoryName, etc,.

Creando el archivo “instance”

En el elemento “intance” se definen los elementos y atributos que pueden ser cambiados por el usuario al aplicar el patrón. Por ejemplo el nombre del objeto a generar, que objetos son generados, filtros.

Ejemplo de un archivo de instance:

```
<?xmlversion="1.0"encoding="utf-8"?>
<PatternName="Category"Version="0.1.0"RootElement="instance"RootType="Instance">
 <ElementTypes>
    <ElementTypeName="Instance"Caption="Category Pattern Instance"ChildrenOrdered="Default">
      <Attributes>
        <AttributeName="showLeafItemsInTreeView"Type="bool"Description="Show Leaf Items In TreeView"DefaultValue="True" />
      </Attributes>
      <ChildrenElements>
        <ChildElementName="attributesName"ElementType="AttributesName"Multiple="false"Optional="false" />
        <ChildElementName="transactionsName"ElementType="TransactionsName"Multiple="false"Optional="false" />
      </ChildrenElements>
    </ElementType>
    <ElementTypeName="AttributesName"Caption="Attributes Names"ChildrenOrdered="Default">
      <Attributes>
        <AttributeName="categoryId"Type="string"Category="General" />
        <AttributeName="categoryName"Type="string"Category="General" />
      </Attributes>
```

Por ejemplo en este archivo existen atributos booleanos que determinan si las hojas son mostradas en un menú o no y su valor por defecto es true.

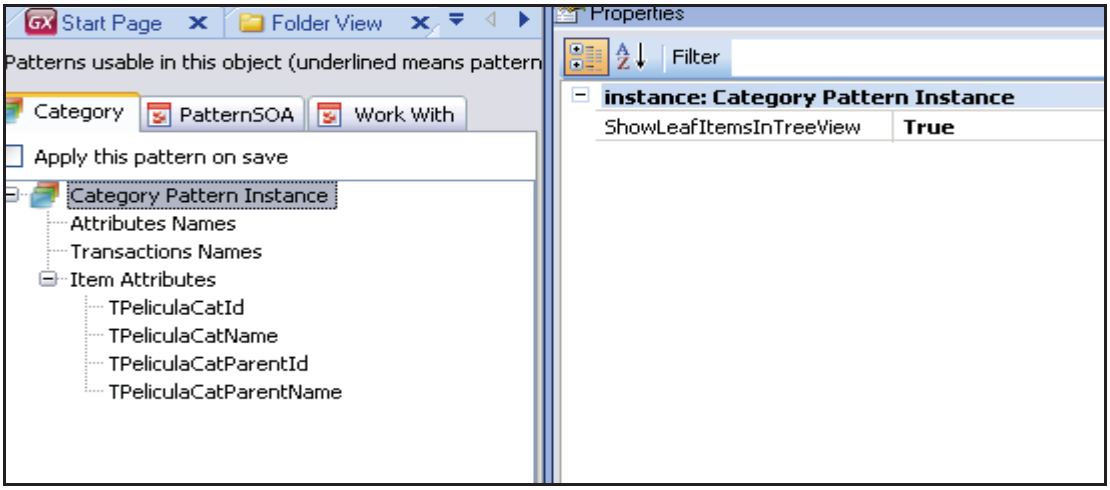


Fig. 35 – Opciones patrones [IDE Genexus]

Además muestra otros niveles (children elements) y los atributos de dichos niveles.

Anexo 3: Microsoft Visual Studio

Visual estudio

Visual Studio es la herramienta que se usa para desarrollar y compilar el patrón Genexus, es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones Web ASP.NET, Servicios Web XML, aplicaciones de escritorio y aplicaciones móviles. Visual Basic, Visual C++, Visual C# y Visual J# utilizan el mismo entorno de desarrollo integrado (IDE), que les permite compartir herramientas y facilita la creación de soluciones en varios lenguajes. Asimismo, dichos lenguajes aprovechan las funciones de .NET Framework, que ofrece acceso a tecnologías clave para simplificar el desarrollo de aplicaciones Web ASP y Servicios Web XML **[23]**

Algunos aspectos destacados de visual studio:

Característica	Descripción
Servicio web XML	Los Servicios Web XML son aplicaciones que pueden recibir solicitudes y datos mediante XML a través de HTTP. no están ligados a una tecnología de componentes particular o a una convención de llamada de objetos y, por tanto, se puede obtener acceso a ellos mediante cualquier lenguaje, modelo de componente o sistema operativo. En Visual Studio, se pueden crear e incluir con rapidez Servicios Web XML mediante Visual Basic, Visual C#, JScript o servidor ATL.
Formularios web	Los formularios Web Forms son una tecnología ASP.NET que se utiliza para crear páginas Web programables. Los formularios Web Forms se representan como código HTML y secuencias de comandos compatibles con exploradores, lo que permite ver las páginas en cualquier explorador y plataforma. Mediante el uso de formularios Web Forms se pueden crear páginas Web arrastrando y colocando controles en el diseñador y agregando código posteriormente, de forma parecida a la creación de formularios en Visual Basic.

Windows Forms	Los formularios Windows Forms sirven para crear aplicaciones de Microsoft Windows en .NET Framework. Este marco de trabajo proporciona un conjunto de clases claro, orientado a objetos y ampliable, que permite desarrollar complejas aplicaciones para Windows. Además, los formularios Windows Forms pueden actuar como interfaz de usuario local en una solución distribuida de varios niveles.
Visual studio tools para Office	Microsoft Visual Studio 2005 Tools para Microsoft Office System puede ayudarle a crear soluciones al extender documentos de Word 2003 y libros de Excel 2003 mediante Visual Basic y Visual C#. Visual Studio Tools para Office incluye nuevos proyectos de Visual Studio para crear el código subyacente en documentos de Word, plantillas de Word, libros de Excel y plantillas de Excel.
Visual web developer	Visual Studio incluye un nuevo diseñador de páginas Web denominado Visual Web Developer que incluye muchas mejoras para la creación y edición de páginas Web ASP.NET y páginas HTML. Proporciona una forma más fácil y rápida de crear páginas de formularios Web Forms que en Visual Studio .NET 2003. Visual Web Developer incluye mejoras en todas las áreas de desarrollo de sitios Web. Puede crear y mantener los sitios Web como carpetas locales, en Servicios de Internet Information Server (IIS), o en un servidor FTP o SharePoint. El diseñador Visual Web Developer admite todas las mejoras de ASP.NET, incluidas las casi dos docenas de nuevos controles que simplifican muchas tareas de desarrollo Web.

Aplicaciones para dispositivos inteligentes	El entorno integrado de Visual Studio incluye herramientas destinadas a dispositivos como los PDA y Smartphone. Entre las mejoras se encuentran tiempos de ejecución de dispositivos nativos y herramientas de Visual C++, diseñadores administrados que proporcionan un modo WYSIWYG mejorado específico para cada plataforma y compatibilidad con varios factores de forma, un nuevo emulador, herramientas de control de datos similares al escritorio, y proyectos de implementación para el usuario final que eliminan la edición manual de los archivos .inf.
Compatibilidad con XML	El Lenguaje de marcado extensible (XML) proporciona un método para describir datos estructurados. XML es un subconjunto de SGML optimizado para la entrega a través de Web. El Consorcio World Wide Web (W3C) define los estándares de XML para que los datos estructurados sean uniformes e independientes de las aplicaciones. Visual Studio es totalmente compatible con código XML e incluye el Diseñador XML para facilitar la edición de XML y la creación de esquemas XML.

Tabla 6 - Características .Net

El entorno .NET Framework

.NET Framework es un entorno multilenguaje que permite generar, implantar y ejecutar aplicaciones y Servicios Web XML. Consta de tres partes principales:

- **Common Language Runtime:** A pesar de su nombre, el motor en tiempo de ejecución desempeña una función tanto durante la ejecución como durante el desarrollo de los componentes. Cuando el componente se está ejecutando, el motor en tiempo de ejecución es responsable de administrar la asignación de memoria, iniciar y detener subprocesos y procesos, y hacer cumplir la directiva de seguridad, así como satisfacer las

posibles dependencias del componente sobre otros componentes. Durante el desarrollo, el papel del motor en tiempo de ejecución cambia ligeramente; a causa de la gran automatización que permite (por ejemplo, en la administración de memoria), el motor simplifica el trabajo del desarrollador, especialmente al compararlo con la situación actual de la tecnología COM.

- **Clases de programación unificadas:** El entorno de trabajo ofrece a los desarrolladores un conjunto unificado, orientado a objetos, jerárquico y extensible de bibliotecas de clases. Actualmente, los desarrolladores de C++ utilizan las Microsoft Foundation Classes y los desarrolladores de Java utilizan las Windows Foundation Classes. El entorno de trabajo unifica estos modelos dispares y ofrece a los programadores de Visual Basic y JScript la posibilidad de tener también acceso a las bibliotecas de clases. Con la creación de un conjunto de Interfaces de Programación de Aplicaciones (API) comunes para todos los lenguajes de programación, Common Language Runtime permite la herencia, el control de errores y la depuración entre lenguajes. Todos los lenguajes de programación, desde JScript a C++, pueden tener acceso al entorno de trabajo de forma parecida y los desarrolladores pueden elegir libremente el lenguaje que desean utilizar.

- ASP.NET construye las clases de programación de .NET Framework, lo que proporciona un modelo de aplicación Web con un conjunto de controles e infraestructura que facilitan la generación de aplicaciones Web. ASP.NET incluye un conjunto de controles que encapsulan elementos comunes de interfaz de usuario de HTML, como cuadros de texto, botones y cuadros de lista. Sin embargo, dichos controles se ejecutan en el servidor Web, y representan la interfaz de usuario en el explorador como HTML. En el servidor, los controles exponen un modelo de programación orientado a objetos que proporciona la riqueza de la programación orientada a objetos al desarrollador Web. ASP.NET también proporciona servicios de infraestructura, como la administración de estado y el reciclaje de procesos, que reduce aún más la cantidad de código que debe escribir el desarrollador y aumenta la confiabilidad de la aplicación. Asimismo, ASP.NET utiliza estos mismos conceptos para permitir a los desarrolladores la entrega de software como un servicio. Al utilizar características de Servicios Web XML, los desarrolladores de ASP.NET pueden escribir su lógica empresarial y utilizar la infraestructura de ASP.NET para entregar ese servicio a través de SOAP. Para obtener más información, vea Introducción a la programación de servicios Web XML en código administrado.

Anexo 4: Archivos DKT

Los archivos DKT son los template que definen las partes de los objetos generados por el pattern: ejemplo eventos de los webform, formulario de los webform, variables, conditions, et.

Algunos de los tags y objetos más comúnmente usados son los siguientes:

Directivas

Directivas del template

Todo archivo template requiere iniciar con una directiva, esta especifica las propiedades generales del template, tal como el lenguaje del template (El lenguaje en el cual las instrucciones de proceso son escritas, las cuales pueden ser C# o VB.net), se ingresa el lenguaje y una pequeña descripción del template para propósitos de la documentación.

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

Directivas de importación

La directiva de importación agrega una sentencia C# (o visual studio) al código del template, Esto, junto con la directiva de ensamblado es lo que se necesita para invocar clases de C#. El parámetro namespace indica el espacio que se desea importar

```
<% @ Import Namespace = "System.XML"%>
```

Directiva de ensamblado

La directiva del ensamblado le dice al motor de plantillas a que ensamblados externos debe hacer referencia a la plantilla. Por ejemplo, si desea utilizar las clases XML de. NET en la plantilla, es necesario hacer referencia al ensamblado System.Xml. El parámetro "Name" es el nombre completo del ensamblado. El ensamblado debe existir en la caché global de ensamblados, en el directorio donde se encuentra la plantilla, o en un directorio superior al que la plantilla se encuentra.

```
<% @ Assembly Name = "System.XML"%>
```

Las referencias al System, System.Drawing, System.Design y System.Windows.Forms se incluyen automáticamente.

Directiva de propiedades

La directiva propiedad declara propiedades que se van a utilizar como parámetros por la plantilla. Los parámetros necesarios son el nombre y tipo de la propiedad, el cual debe corresponder a un tipo de datos .Net válido (clase, enumeración, etc, y en particular para los tipos de datos "primitivas" deben utilizar sus nombres .net completos, como' por ejemplo “System.Int32" para los números enteros en lugar de sólo int.

```
<% @ Nombre de la propiedad Tipo = "Nombre" = "System.String"%>
```

Directiva subplantilla

Para aplicar el mismo "fragmento de plantilla" en varios lugares, se puede crear una subplantilla e invocarlo. El uso de una sub-plantilla es un proceso de dos pasos: en primer lugar tiene que declarar en la parte superior de la plantilla principal. Entonces, es necesario invocarlo en la plantilla con la directiva CallSubTemplate. Por ejemplo:

```
<% @ Subplantilla Name = "GridAttributes" Name = MergeProperties "GridAttributes.dkt"% = "False">
```

El parámetro Name determina el "nombre lógico" de la sub-plantilla, mientras que el parámetro del archivo indica nombre real de la plantilla. Debe estar ubicado en la misma carpeta que la plantilla que realiza el llamado. Si el parámetro booleano MergeProperties está establecido en" verdadero", todas las propiedades de las plantillas padre se asignan automáticamente a la sub-plantilla.

Etiquetas

Las etiquetas se utilizan para incluir código .NET en el cuerpo de la plantilla:

Etiquetas **<%** y **%>**

Estas etiquetas se utilizan para cualquier cantidad de código que no proporciona salida directamente a la plantilla.

<% Foreach (atributo GXAttribute en transaction.AllAttributes ())%>

<% {%>

1

<%}%>

Este ejemplo se repetirá a través de cada elemento de la colección, y entregará como salida un 1 a la plantilla. Esto no es muy útil, ya que es probable que queremos dar salida a algo así como el nombre del atributo, que es donde la siguiente etiqueta entra en escena:

Etiquetas <% = y %>

Estas etiquetas se utilizan para retornar una cadena a la plantilla, cualquier código que se utiliza entre estas etiquetas debe convertirse en una cadena simple. He aquí un ejemplo de estas etiquetas:

```
<% Foreach (atributo GXAttribute en transaction.AllAttributes ()%>
<% {%>
Nombre de atributo es <% = attribute.Name%>
<%}%>
```

Etiquetas Script

Etiquetas de secuencias de comandos se utilizan para incluir un trozo de código en la plantilla que no se utiliza para afectar directamente a la salida de la plantilla. Este es el mejor lugar para poner los métodos de ayuda que usted puede llamar a través del resto de la plantilla. El uso de etiquetas de script que puede reducir al mínimo la cantidad de código que se incluye entre las etiquetas <% %>, por lo que las plantillas quedaran más legibles y manejables. Aunque aún son soportadas las etiquetas de script están consideradas obsoletas por estar incluidas en la directiva de "incluye".

Etiquetas de comentario

Etiquetas de comentario se puede utilizar para incluir comentarios en su plantilla sin afectar la salida de la plantilla. El carácter de comentario es "-" y se puede utilizar en el interior del <%%> etiquetas de código.

```
<% - Esto es un comentario -%>
```