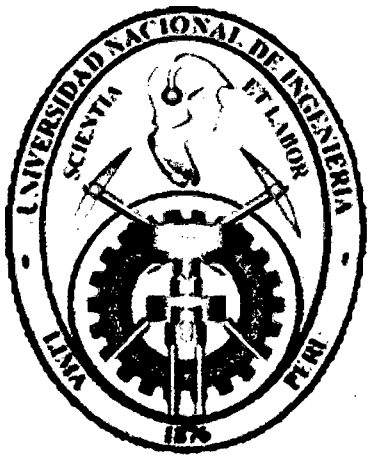
UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERIA INDUSTRIAL Y DE SISTEMAS

SECCION POSGRADO



Especificación de Metodología Agil para Desarrollo de Software, Apoyada en la Herramienta CASE Genexus

TESIS

PARA OPTAR EL GRADO ACADEMICO DE:

MAESTRO EN CIENCIAS CON MENCION EN INGENIERIA DE SISTEMAS

Lic. Daniel Luis Fernández Verástegui

Lima - Perú

2010

DEDICATORIA

A MIS PADRES ROSA Y DANIEL,

POR INCULCARME EL AMOR AL ESTUDIO Y A LA SUPERACION

A MI ESPOSA GLADYS Y A MIS HIJOS POR SU COMPRENSION Y APOYO PARA LOGRAR SUPERAME EN MI FORMACION PROFESIONAL

AGRADECIMIENTO

AGRADEZCO A MIS PROFESORES POR SUS VALIOSAS ENSEÑANZAS

A MIS COMPAÑEROS DE ESTUDIOS DE MAESTRIA DE LA UNI POR HABERME APOYADO A CULMINAR MIS ESTUDIOS

A MI ASESOR DE TESIS Y JURADOS ESPECIALISTAS POR SU VALIOSA COLABORACION PARA EL DESARROLLO DEL PRESENTE TRABAJO DE INVESTIGACION

INDICE

DESCRIPTORES TEMATICOS.

- Software.
- Ingeniería de Software.
- Metodología de Desarrollo de Software.
- Metodología Agil de Desarrollo de Software.
- Base de Conocimiento.
- Herramienta CASE Genexus.
- Ciclo de Vida del Desarrollo de Software.
- Calidad del Software.
- Calidad y Aseguramiento de la Calidad del Software.
- Microfinanzas.

RESUMEN

La Superintendencia de Banca Seguros y AFP (SBS) informó que entre el 8 y 10 de Octubre del 2008, en el Foro de la Microempresa, Perú fue elegido como el País que reúne las mejores condiciones para las microfinanzas en América Latina y el Caribe.

La mayoría de los pobres en el mundo y entre ellos Peruanos, no tienen acceso a los servicios proporcionados por la banca formal debido a que siempre se ha considerado muy costosos y de alto riesgo atender a los clientes, llamémosles pequeños, con las técnicas bancarias convencionales. Por lo que encuentran en el sector microfinanciero una alternativa para mejorar su condición económica.

Muchas de las instituciones microfinancieras del Perú que han ido creciendo hasta llegar a atender miles de clientes, no cuentan con el Software adecuado que les permita administrar su cartera de créditos, así como las operaciones financieras eficientemente. Este es el caso de la empresa PYME MICREDITPERU de la ciudad de Trujillo, que cuenta con 12 agencias a nivel nacional.

El objetivo de esta Tesis consistió en definir una Metodología Ágil para el Desarrollo de Software apoyada en la herramienta CASE Genexus, que pueda ser aplicada en el Desarrollo de Software de calidad, para la empresa MICREDITPERU y en otras empresas similares.

Esta Metodología se diseñó integrando herramientas de la Ingeniería de Sistemas tales como la Ingeniería de Software, las mejores prácticas de las principales Metodologías de Desarrollo de Software tradicionales y ágiles existentes y principalmente con el soporte de la herramienta CASE Genexus, en las diversas etapas del ciclo de vida del Desarrollo de Software.

Herramienta basada en una rama de la inteligencia artificial, la administración del conocimiento.

Los beneficios obtenidos de este trabajo de investigación fueron:

- Se definió una Metodología Ágil de Desarrollo de Software apoyada en la Herramienta CASE Genexus, que aporta valor a la comunidad informática y que permite producir Software de manera eficaz y eficiente, asegurando su Calidad.

- Se da a conocer a los de desarrolladores de software las bondades de las herramientas CASE, contribuyendo así, a romper el escepticismo que existe por parte de muchos de ellos en lo que se refiere a su utilidad.

- Se diseñó una Metodología Ágil de Desarrollo de Software que permite, eliminar o mitigar las limitaciones de las principales metodologías de desarrollo actuales y que puede aplicarse satisfactoriamente en el desarrollo de software de calidad, de la Empresa MICREDITPERU y de aquellas empresas que tengan características semejantes.

INTRODUCCION

Muchas de las instituciones microfinancieras son organizaciones sin fines de lucro que han decidido participar como intermediarios financieros por motivos sociales. Es entendible por lo tanto, que en sus inicios no cuenten con conocimientos administrativos y sistemas sofisticados y que además sus sistemas de software tengan la tendencia a ser rudimentarios. Pero a medida que han ido creciendo hasta llegar a atender miles de clientes y a muchos más, han empezado a sentir la necesidad de mejorarlos.

Los gerentes de instituciones crediticias en proceso de expansión pierden la capacidad de mantener contacto con lo que está sucediendo en el campo y llegan a una situación en la que se dan cuenta de que no pueden administrar su cartera de créditos, así como las operaciones financieras eficientemente sin contar con mejor información sobre ellas.

Por lo tanto, encontramos que muchas instituciones de microfinanzas están fuertemente motivadas a mejorar su sistema de software. Desafortunadamente, esto no es muy fácil de llevarse a cabo. Debido a que el Software de aplicaciones comerciales existentes usualmente, no proporciona la solución ideal. Muchas instituciones de microfinanzas han tratado de desarrollar ellas mismas gran parte de su Software. La mayoría de las que han optado por esta alternativa reportan que han tenido que sufragar costos mayores a los previstos inicialmente, tanto en dinero, tiempo requerido y dedicación de la gerencia, sin lograr soluciones del todo aceptables, principalmente por no haber usado una metodología de desarrollo de software adecuada,.

La Metodología de desarrollo de software en ingeniería de software es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo de Software.

Las Metodologías ágiles son temas de actualidad en ingeniería de software, han despertado gran interés en los ingenieros de software, profesores, e incluso alumnos, lo que hace prever una gran proyección industrial.

Actualmente, las áreas de Tecnologías de Información de las empresas u organizaciones que desarrollan Software en respuesta a estas exigencias se han visto en la necesidad de hacer uso de herramientas CASE (Computer Aided Software Engineering). Esto con el fin de aumentar la eficacia de los procesos de desarrollo de Software, al soportar la realización de sus actividades haciendo uso de modernas tecnologías de información.

Teniendo en cuenta la problemática antes expuesta, en el presente trabajo de tesis, se diseñó una Metodología Ágil para el Desarrollo de Software, que haciendo uso de las mejores prácticas de las Metodologías Tradicionales y ágiles de Desarrollo de Software y con el apoyo de tecnologías modernas en el Desarrollo de Software, como lo son las Herramientas CASE. (Computer-Aided Software Engineering, Ingeniería de Software Asistida por Computadora), para de ese modo capitalizar los beneficios que dichas herramientas proporcionan, tanto en el desarrollo como en el mantenimiento del Software.

Este estudio de investigación abarca 6 capítulos, las conclusiones y recomendaciones, la bibliografía y finalmente el glosario; que describimos brevemente a continuación:

Capítulo I, Comprende el planteamiento de la investigación, aquí definimos el planteamiento del problema, su definición, justificación y delimitación de la investigación, hipótesis y finalmente se definen los objetivos principales y específicos que se constituyen en las líneas directrices durante todo el proceso de la investigación.

Capítulo II, Comprende el marco teórico, conformado por los antecedentes de la investigación; el estudio de las principales metodologías tradicionales de desarrollo de software existentes, sus ventajas y

desventajas; el estudio de las principales metodologías ágiles de desarrollo de software existentes, sus ventajas, desventajas; Las características de los métodos formales de desarrollo de software, las herramientas CASE, sus características, la reseña organizacional de la Empresa Micrédito sac. Y finalmente las bases teóricas en que se basó esta investigación.

Capitulo III, comprende la metodología de la investigación, aquí se describe la metodología que sirvió de base a esta investigación, como son la investigación en ingeniería de software, el objeto de estudio de la ingeniería de software, el método de investigación para la ingeniería de software y finalmente el método de investigación en acción.

Capítulo IV, Comprende la metodología de desarrollo de software propuesta, conformado por, primero las bases de la propuesta como son: Los objetivos de la metodología propuesta, la descripción de las herramientas y técnicas usadas en la metodología propuesta, principalmente las características y funcionalidades de la herramienta CASE Genexus. Luego en una segunda sección se define la metodología propuesta propiamente dicha, principios y prácticas en que se basa la metodología, su ciclo de vida, las fases que comprende, se describe detalladamente las principales fases soportadas por la herramienta CASE Genexus, los roles representados, la naturaleza iterativa e incremental de la metodología propuesta y finalmente los factores críticos de éxito de la metodología.

Capítulo V, Comprende el estudio de casos, se aplica la metodología en el desarrollo de un prototipo de una empresa de venta de equipos de cómputo y del desarrollo de un prototipo de la empresa de microfinazas Micredito SAC. de la ciudad de Trujillo.

Capítulo VI, Comprende el análisis de resultados, En este capítulo, tomando como base la Metodología de Investigación descrita en el capítulo III, se analizan los resultados obtenidos de la aplicación de la Metodología propuesta, en los casos de estudio descritos en el capítulo V, de este trabajo de Tesis de Maestría. Se analiza el logro de los objetivos planteados al

inicio de este trabajo, y se indican los principales aportes de esta Tesis de Maestría.

Las conclusiones y recomendaciones, comprende las conclusiones a las que se arribó en este trabajo de investigación, para finalmente se anotan un conjunto de recomendaciones que surgen como consecuencia del trabajo a lo largo del proceso investigativo.

La bibliografía, comprende el soporte bibliográfico de este trabajo de investigación.

El glosario, comprende una breve descripción del significado de algunos de los términos empleados en este trabajo de investigación.

CAPITULO I

PROBLEMA DE INVESTIGACION

1.1. PLANTEAMIENTO DEL PROBLEMA

Según [Flores, R, 2009] "Las Microfinanzas han evolucionado de manera impresionante en los últimos veinte años. El Perú no sólo no ha sido ajeno a este desarrollo mundial, sino que se ha convertido, conjuntamente con Bolivia, en uno de los mercados líderes de Latinoamérica, tanto por el volumen de sus operaciones y calidad de sus instituciones como por la fortaleza de su regulación.

Este reconocimiento nos lleva a la necesidad de ser de los primeros en preguntarnos qué entendemos por Microfinanzas. Para ello, centraremos nuestro análisis en su componente más conocido: el microcrédito (1); y nos remitiremos a Muhammad Yunus, galardonado con el Premio Nobel de la Paz 2006, y quien es reconocido como el creador del microcrédito moderno.

De acuerdo con Yunus, el microcrédito lo constituyen aquellos programas que ofrecen crédito sin garantías para actividades generadoras de ingresos, y que son dirigidos encaminados a que los pobres superen la línea de la pobreza. De acuerdo con esta definición, el microcrédito debe ser entendido en un sentido más complejo, y no sólo pensar que se trata de prestar dinero a los pobres. De ello, surge la pregunta ¿qué instituciones vienen trabajando para promover este concepto de microcrédito en el país?

Asimismo, es necesario entender si las instituciones públicas y privadas que promueven y regulan el sector financiero están interesadas en promover este concepto de microcrédito. Finalmente, y quizás lo más importante, es saber si las instituciones que otorgan créditos a la población de escasos recursos tienen el objetivo de ayudarlos a superar su condición de pobres.

(1) El concepto de microfinanzas agrupa a diferentes servicios financieros y no financieros, dirigidos a atender las diferentes necesidades de la población marginada, tales como microseguros, capacitación, micropensiones, crédito para la construcción y mejoramiento de vivienda, etc.

Esto nos lleva a dos reflexiones: la primera, es que la definición de microcrédito en la regulación financiera resulta muy amplia (2), ya que aglutina diversas modalidades de crédito, destinos de fondos (3) y perfiles de clientes; por lo que sería necesario contar con una definición más precisa. Así, el concepto de microcrédito podría subdividirse en categorías que permitan una mayor precisión en el diseño de las políticas: microcrédito, créditos para pequeños negocios, inversión en mejora de la vivienda y consumo para personas de escasos recursos.

La segunda reflexión es que dado que las instituciones de microcrédito, bajo la definición de Yunus, deben tener como objetivo ayudar a los pobres a dejar de serlo, éstas deberían tener un fin social superior al fin de lucro de sus accionistas. En efecto, Yunus nos habla de manera permanente de la importancia de lograr instituciones microfinancieras sólidas y eficientes como requisito para asegurar el éxito de los programas de microcrédito; pero al mismo tiempo, resalta la importancia del cumplimiento de la misión social y su primacía sobre el lucro. Por ello, microcrédito y maximización del lucro son incompatibles.

El cumplimiento de la misión social y la importancia de lograr que los pobres puedan superar su condición de pobreza, debe llevar a las instituciones microfinancieras a identificar las necesidades insatisfechas de sus clientes.

(2) La Superintendencia de Banca, Seguros y AFP clasifica al microcrédito como aquel dirigido a una persona jurídica o natural con negocio (es decir que cuente con RUS o RUC) y cuya exposición en el sistema financiero nacional no supere los 30,000 USD.

(3) Bajo la calificación actual, podría incluirse como microcrédito a un crédito otorgado a un micro o pequeño empresario, sin diferenciar si los recursos se utilizan para invertir en su negocio, para comprar ropa o regalos para su familia o para construir un cuarto adicional a su vivienda. Más aún, se podría incluir en esta calificación a la subsidiaria de una transnacional si ésta tiene deudas menores a US$ 30,000 USD y no forma parte de ningún grupo financiero registrado localmente.

En este sentido, el microcrédito debe ser entendido como una herramienta y no como la solución mágica para aliviar la pobreza. Es por ello que, dado el gran número de necesidades insatisfechas de la población de bajos recursos, el acceso al crédito resulta tan importante como el acceso a la oportunidad de ahorrar, de acceder a seguros de vida y planes de salud, a un plan de pensiones que asegure su tranquilidad futura, a la oportunidad de adquirir o mejorar su vivienda, a dar una educación técnica o superior a sus hijos, etc. Estas son necesidades que en la actualidad no viene siendo suficientemente atendidas por las microfinanzas.

La Superintendencia de Banca Seguros y AFP (SBS) de Perú informó en el marco del Foro de la Microempresa, organizado por el Fondo Multilateral de Inversiones (FOMIN) y el gobierno de Paraguay, celebrado en su capital Asunción, entre el 8 y 10 de Octubre del 2008, los resultados de un último estudio realizado por el Economist Intelligence Unit (EIU) con apoyo del Banco Interamericano de Desarrollo (BID) y de la Corporación Andina de fomento (CAF), en donde el Banco Interamericano de Desarrollo (BID) en su informe anual "Entorno de negocios para las microfinanzas de América Latina y el Caribe", más conocido como el Microscopio Regional. Otorgó el primer lugar a Perú, como el País que reúne las mejores condiciones para las microfinanzas en America latina y el Caribe.

Según [Sánchez C., 2008], "Por el Tratado de Libre Comercio (TLC) que Perú suscribió con Estados Unidos, se estima que habrá nuevas oportunidades laborales para los peruanos, también se prevé que generará algunas limitaciones en algunos sectores denominados "sensibles", como en la agricultura, que es el sector donde generalmente donde laboran las personas más pobres del país.

No obstante, el acuerdo comercial impulsará a la mediana, pequeña y micro empresa, permitiendo que el sector microfinanciero genere también nuevas oportunidades para las PYMEs.

La pobreza que tiene sus raíces en los asentamientos humanos, comunidades urbano-marginales y rurales, así como en los grupos étnicos de la Amazonía, encuentra en el sector microfinanciero una gran resistencia para seguir avanzando.

El sector microfinanciero será en los siguientes años, uno de los sectores más dinámicos de la economía peruana. Urge entonces que desde ahora, este sector se prepare para asumir nuevos roles que van desde la lucha frontal contra la pobreza hasta la formalización de nuevas pequeñas empresas.

En el Perú hay nuevas generaciones de emprendedores que requieren microcréditos para salir adelante. La cultura emprendedora es ya una realidad en el país. Hay más de tres millones de unidades productivas y comerciales y esta cifra va en aumento".

Según [Waterfield C., Ramsing N., 1998] "La mayoría de los pobres en el mundo (entre ellos Peruanos), no tienen acceso a los servicios proporcionados por la banca formal debido a que siempre se ha considerado muy costoso y de alto riesgo atender a los clientes, llamémosles pequeños, con las técnicas bancarias convencionales.

Muchas de las instituciones microfinancieras son organizaciones sin fines de lucro que han decidido participar como intermediarios financieros por motivos sociales. Es entendible por lo tanto, que en sus inicios no cuenten con conocimientos administrativos y sistemas sofisticados y que además sus sistemas de información computarizados tengan la tendencia a ser rudimentarios. Pero a medida que han ido creciendo hasta llegar a atender miles de clientes y a muchos más, han empezado a sentir la necesidad de mejorarlos.

Los gerentes de instituciones crediticias en proceso de expansión pierden la capacidad de mantener contacto con lo que está sucediendo en el campo y llegan a una situación en la que se dan cuenta de que no pueden

administrar su cartera de créditos, así como las operaciones financieras eficientemente sin contar con mejor información sobre ellas.

Por lo tanto, encontramos que muchas instituciones de microfinanzas están fuertemente motivadas a mejorar su sistema de información computarizados. Desafortunadamente, esto no es muy fácil de llevarse a cabo. Debido a que el Software de aplicaciones comerciales existentes usualmente, no proporciona la solución ideal. Muchas instituciones de microfinanzas han tratado de elaborar ellas mismas gran parte de su Software. La mayoría de las que han optado por esta alternativa informan que han tenido que sufragar costos mayores a los previstos inicialmente, tanto en dinero, tiempo requerido y dedicación de la gerencia, sin lograr soluciones del todo aceptables".

[Ś Finanzgruppe, 2003] menciona que: *"7. Las microfinanzas significan buena organización y técnica."*

Las microfinanzas no se caracterizan por volúmenes altos, sino por grandes números de transacciones.

La concentración de un gran número de depósitos pequeños y de la otorgación de muchos créditos pequeños, no requieren particularmente de técnicas sofisticadas de análisis, sino más bien una organización eficiente. Esto es muy importante, ya que los clientes, no viven solamente en la capital o en los centros urbanos, sino que están repartidos por todo el país. Estos dos factores juntos – grandes cantidades de transacciones y la organización descentralizada – solo se pueden manejar con personal bien capacitado y con un equipo técnico apropiado.

Las microfinanzas por lo tanto no quieren decir papel y lápiz, sino hardware robusto y software eficiente."

La tarea de Desarrollar Software es similar a la de construir una casa. Los arquitectos preparan los planos que guiaran las labores de los trabajadores de la construcción, quienes usan diversas herramientas para llevar a cabo los trabajos de carpintería, electricidad, plomería, etc. El trabajo

de los Desarrolladores de Software se guían por planos que se denominan Modelos, tales como: Los Modelos de Casos de Uso, Modelo de Clases, de Actividades, Prototipos, etc.

La Metodología de desarrollo de software en ingeniería de software es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo de Software.

En los últimos 20 años las notaciones de modelado y luego las herramientas pretendieron ser la "solución" para lograr el éxito en el desarrollo de software, sin embargo, estas expectativas no fueron satisfechas, esto se debe en gran medida a que se dejó de lado un elemento importante, como son las Metodologías de desarrollo. Hasta hace poco el proceso de desarrollo le daba una gran importancia al control del proceso por medio de definiciones rigurosas de roles, actividades y artefactos, así como modelado y documentación exhaustiva. Esta forma "tradicional" de enfocar el desarrollo de software ha demostrado ser efectivo y útil en proyectos de gran envergadura en cuanto a tiempo y recursos empleados. Sin embargo, este enfoque no resulta ser el más indicado para la gran mayoría de los proyectos actuales donde el entorno del sistema es cambiante y la exigencia de entrega de los sistemas requiere tiempos de desarrollo mínimos pero conservando alta calidad.

Debido a la dificultad de usar las Metodologías tradicionales contemplando estas exigencias de tiempo y flexibilidad, muchos desarrolladores desisten del uso de estas Metodologías y optan por el uso de las denominadas Metodologías ágiles como una posible solución para llenar ese vacío metodológico.

Las Metodologías ágiles son temas de actualidad en ingeniería de software, han despertado gran interés en los ingenieros de software, profesores, e incluso alumnos, lo que hace prever una gran proyección industrial.

Existe un amplio rango de proyectos industriales de software en los

cuales sus características se ajustan a las de las Metodologías ágiles.

La manera de Desarrollar Software, ha ido experimentando continuamente cambios a través del tiempo, buscando asegurar el logro de resultados esperados cada vez que se inicia un nuevo proyecto de desarrollo. Los objetivos del Software a Desarrollar podrían definirse fácilmente, pero seguir el ciclo adecuado para poder alcanzarlos y que este ciclo conserve el equilibrio necesario entre la eficiencia y la efectividad es una tarea difícil. Esto significa buscar la manera de desarrollar Software de Calidad.

Actualmente, las áreas de Tecnologías de Información de las empresas u organizaciones que desarrollan Software en respuesta a estas exigencias se han visto en la necesidad de hacer uso de herramientas CASE (Computer Aided Software Engineering). Esto con el fin de aumentar la eficacia de los procesos de desarrollo de Software, al soportar la realización de sus actividades haciendo uso de modernas tecnologías de información.

La industria del software no evoluciona a la misma velocidad del hardware, ni al mismo ritmo al que nacen los requerimientos por nuevas aplicaciones en las organizaciones que les permitan adaptarse a un entorno dinámico y competitivo. Sin embargo, la industria del software ha respondido con algunas herramientas; y entre ellas se tienen los lenguajes de programación de cuarta generación, pero estas tuvieron poco impacto en la solución de los diversos problemas de software que confrontaban las organizaciones, sobre todo para reducir el trabajo acumulado de aplicaciones, tanto de nuevas como de aquellas que requieren mantenimiento. Otra herramienta de uso más reciente es la llamada Ingeniería de Software Asistida por Computador (CASE), recomendada por Kerr [Kerr, 1989] como la mejor dirección a tomar para enfrentar la crisis del software, pues opina que con ella existe un desarrollo sistemático uso de mejores métodos de prueba y se obtiene efectivamente un producto de calidad, aunque advierte: "incorporar un conjunto de herramientas CASE sin planificación y en manos de gente no entrenada puede producir efectos

negativos".

Una herramienta CASE es un software utilizado con la finalidad de aumentar la productividad del analista y la calidad de los sistemas desarrollados por éstos [Smith, 1989]; [Kerr, 1989]; [Kendall, 1999];[Chen y Norman, 1992]. Por tal motivo muchas organizaciones en los EE.UU. las adquirieron, pero su implementación en muchas de ellas fue un fracaso [Kemerer, 1992].

La herramientas CASE es una herramienta y no una Metodología, como se refieren frecuentemente a ella, algunos vendedores de CASE [Callaos, 1991]. Actualmente las empresas peruanas, tanto públicas como privadas, están interesadas en adquirir nuevas tecnologías, desarrollar sistemas de información y/o aplicar reingeniería a los viejos sistemas. En todos estos procesos las herramientas CASE representan una pieza clave, pues contribuyen con el trabajo del analista, haciéndolo más eficiente.

Teniendo en cuenta la problemática antes expuesta, en el presente trabajo de tesis, se diseñó una Metodología Ágil para el Desarrollo de Software, que haciendo uso de las mejores prácticas de las Metodologías Tradicionales y ágiles de Desarrollo de Software y con el apoyo de Tecnologías modernas en el Desarrollo de Software, como lo son las Herramientas CASE. (Computer-Aided Software Engineering, Ingeniería de Software Asistida por Computadora), para de ese modo capitalizar los beneficios que dichas herramientas proporcionan, tanto en el desarrollo como en el mantenimiento del Software.

DEFINICION DEL PROBLEMA

Tomaremos como referencia de pequeñas empresas de microfinanzas a la Empresa PYME MICREDITPERU cuya oficina principal está ubicada en la ciudad de Trujillo y viene operando desde hace aproximadamente un año.

Actualmente se ha expandido en 12 agencias: Trujillo, Otuzco, Huamachuco, Santiago de Chuco, Cajabamba, Paiján, Chepén, Pacasmayo, Cartavio, Virú, Cascas y Tayabamba.

Sus Procesos están Apoyados por un Software que no satisface sus necesidades de información manifestándose de la manera siguiente:

La Gerencia General y la Gerencia de Operaciones manifiestan que, a nivel de la oficina principal, no se cuenta con información oportuna y confiable tanto a nivel de cada una de las agencias como a nivel de toda la Empresa que les permitan obtener en forma oportuna y confiable: Estados financieros, Indicadores de gestión, Flujos de caja, Gestionar la cartera de préstamos, Presupuestos proyectados y ejecutados, Estadísticas numéricas y graficas.

Los Administradores de las Agencias manifiestan que tienen problemas de oportunidad y confiabilidad en la información que les permita gestionar adecuadamente la cartera de créditos, el desempeño de sus Analistas de Crédito, la calificación de solicitudes de créditos, los estados de cuenta de los Clientes y la obtención de sus estados financieros.

Manifiestan también que presentan la existencia de inconsistencias en la información contenida en los diferentes reportes que se emiten.

Este Software se ha desarrollado sin seguir una Metodología de Desarrollo adecuada. Debido a esto es que aún continúa "afinándose" y presenta problemas de inconsistencias, falta de integración y falta de confiabilidad en la información producida, así como dificultad para obtener información que apoye en la toma de decisiones, a nivel operativo, de control y de planeamiento.

La labores de mantenimiento del Software insume demasiados recursos horas/programador y no satisface los requerimientos de los usuarios.

El mantenimiento correctivo del sistema actual, generalmente, no se hace oportunamente, dando lugar a que se brinde un servicio inadecuado y el consiguiente reclamo de los clientes.

Como parte de las acciones identificadas en el **Plan Estratégico 2008-2020**, de la empresa PYME MICREDITPERU, se ha considerado el

Desarrollo de Software usando herramientas y metodologías de última generación que permita maximizar la productividad y calidad en todo el ciclo de vida del Desarrollo del Software.

1.2. HIPOTESIS Y OBJETIVOS DE LA INVESTIGACION

HIPOTESIS

Al inicio del trabajo de investigación llevado en esta Tesis de Maestría y considerando lo escrito, con anterioridad, en esta sección, se planteo la siguiente hipótesis de trabajo:

"Es factible la especificación de una Metodología Ágil para Desarrollo de Software, apoyada en la Herramienta CASE GENEXUS, que cumpla con los principales requisitos que debe tener una Metodología de Desarrollo de Software, que permita superar o mitigar las limitaciones de las metodologías de desarrollo de software actuales, que permita desarrollar software que satisfaga los requerimientos funcionales y no funcionales de la empresa PYME de microfinanzas MICREDITPERU, así como, los requerimientos generalmente aceptados de las PYMES de microfinanzas de la localidad de Trujillo – Perú."

OBJETIVO PRINCIPAL

El objetivo principal de este trabajo de investigación derivado de esta hipótesis es: Especificar una Metodología Ágil para Desarrollo de Software, apoyada en la Herramienta CASE GENEXUS, que cumpla con los principales requisitos que debe tener una Metodología de Desarrollo de Software, que permita superar o mitigar las limitaciones de las metodologías de desarrollo de software actuales, desarrollar software que satisfaga los requerimientos funcionales y no funcionales de la empresa PYME de microfinanzas MICREDITPERU, así como, los requerimientos generalmente aceptados de las PYMES de microfinanzas de la localidad de Trujillo – Perú.

Para el logro de este objetivo se han planteado los siguientes objetivos específicos:

OBJETIVOS ESPECIFICOS

1. Identificar los principales requisitos deseables que se deben considerar al construir una metodología de desarrollo de software y proponer una serie de criterios de evaluación de dichos requisitos.
2. Establecer las principales características que debe reunir una Metodología para desarrollar el Software de una PYME de Microfinanzas localizada en la ciudad de Trujillo -Perú, asegurando la Calidad del Software.
3. Determinar las limitaciones de las metodologías ágiles de desarrollo de software actuales.
4. Determinar las mejores prácticas usadas en las principales Metodologías de Desarrollo de Software existentes, tanto de las Tradicionales como de las Ágiles.
5. Determinar las principales características y funcionalidades de la Herramienta CASE GENEXUS.
6. Especificar la Metodología a proponer, teniendo en cuenta los objetivos 1., 2., 3. y 4.
7. Validar la Metodología a proponer mediante su aplicación a 2 casos de estudio: Aplicación de Facturación de una Empresa de venta de Equipos de cómputo y de La PYME de microfinanzas MICREDITPERU de la ciudad de Trujillo - Perú.

Con respecto al objetivo específico 1, consideramos lo propuesto por Rafael Menéndez – Barzallano- Ascencio del Departamento de Informática y Sistemas de la Universidad de Murcia – España:

http://www.um.es/docencia/barzana/IAGP/Iagp3.html

¿Qué hay que saber para construir o elegir una Metodología?

En el momento de construir una Metodología, se han de considerar unos requisitos deseables, por lo que seguidamente se proponen una serie de criterios de evaluación de dichos requisitos.

1. **La Metodología debe ajustarse a los objetivos:** Cada aproximación al desarrollo de software está basada en unos objetivos. Por ello la Metodología que se elija debe recoger el aspecto filosófico de la aproximación deseada, es decir que los objetivos generales del desarrollo deben estar implementados en la Metodología de desarrollo.
2. **La Metodología debe cubrir el ciclo entero de desarrollo de software:** Para

   Ello, la Metodología ha de realizar unas etapas:

   Investigación, Análisis de requisitos, Diseño, Implementación, Pruebas y Mantenimiento.

3. **La Metodología debe integrar las distintas fases del ciclo de desarrollo:**

   _ **Rastreabilidad.** Es importante poder referirse a otras fases de un proyecto y fusionarlo con las fases previas. Es importante poder moverse no sólo hacia adelante en el ciclo de vida, sino hacia atrás de forma que se pueda comprobar el trabajo realizado y se puedan efectuar correcciones.

   _ **Fácil interacción entre etapas del ciclo de desarrollo**. Es necesaria una validación formal de cada fase antes de pasar a la siguiente. La información que se pierde en una fase determinada queda perdida para siempre, con un impacto en el sistema resultante.

4. **La Metodología debe incluir la realización de validaciones:** La Metodología debe detectar y corregir los errores cuanto antes. Uno de los problemas más frecuentes y costosos es el aplazamiento de la detección y corrección de problemas en las etapas finales del proyecto. Cuanto más tarde sea detectado el error más caro será corregirlo.

   Por lo tanto cada fase del proceso de desarrollo de software deberá incluir una actividad de validación explícita.

5. **La Metodología debe soportar la determinación de la exactitud del sistema a través del ciclo de desarrollo**: La exactitud del sistema implica muchos asuntos, incluyendo la correspondencia entre el sistema y sus especificaciones, así como que el sistema cumple con las necesidades del usuario. Por ejemplo, los métodos usados para análisis y especificación del sistema deberían colaborar a terminar con el problema del entendimiento entre los informáticos, los usuarios, y otras partes implicadas. Esto implica una comunicación entre usuario y técnico, amigable y sencilla, exenta de consideraciones técnicas.
6. **La Metodología debe ser la base de una comunicación efectiva**: Debe ser posible gestionar a los informáticos, y éstos deben ser capaces de trabajar conjuntamente. Ha de haber una comunicación efectiva entre analistas, programadores, usuarios y gestores, con pasos bien definidos para realizar progresos visibles durante la actividad del desarrollo.
7. **La Metodología debe funcionar en un entorno dinámico orientado al usuario a lo largo de todo el ciclo de vida del desarrollo se debe producir una transferencia de conocimientos hacia el usuario.** La clave del éxito es que todas las partes implicadas han de intercambiar información libremente. La participación del usuario es de importancia vital debido a que sus necesidades evolucionan constantemente. Por otra parte la adquisición de conocimientos del usuario le permitirá la toma de decisiones correctas.

   Para involucrar al usuario en el análisis, diseño y administración de datos, es aconsejable el empleo de técnicas lo más sencillas posible. Para esto, es esencial contar una buena técnica de diagramación.

8. **La Metodología debe especificar claramente los responsables de resultados:** Debe especificar claramente quienes son los participantes de cada tarea a desarrollar, debe detallar de una manera clara los resultados de los que serán responsables.

9. **La Metodología debe poder emplearse en un entorno amplio de proyectos de desarrollo de software:**

   _ **Variedad**. Una empresa deberá adoptar una Metodología que sea útil para un gran número de sistemas que vaya a construir. Por esta razón no es práctico adoptar varias Metodologías en una misma empresa.

   _ **Tamaño, ciclo de vida**. Las Metodologías deberán ser capaces de abordar sistemas de distintos tamaños y rangos de ciclos de vida.

   _ **Complejidad.** La Metodología debe servir para sistemas de distinta complejidad, es decir puede abarcar un departamento, varios departamentos o diversas empresas.

   _ **Entorno**. La Metodología debe servir con independencia de la tecnología disponible en la empresa.

10. **La Metodología se debe de poder enseñar:** En una organización de dimensiones reducidas, serán muchas las personas que la van a utilizar, incluso los que se incorporen posteriormente a la empresa. Cada persona debe entender las técnicas específicas de la Metodología, los procedimientos organizativos y de gestión que la hacen efectiva, las herramientas automatizadas que soportan la Metodología y las motivaciones que subyacen en ella.
11. **La Metodología debe estar soportada por herramientas CASE:** La Metodología debe estar soportada por herramientas automatizadas que mejoren la productividad, tanto del ingeniero de software en particular, como la del desarrollo en general.

    El uso de estas herramientas reduce el número de personas requeridas y la sobrecarga de comunicación, además de ayudar a producir especificaciones y diseños con menos errores, más fáciles de probar, modificar y usar.

12. **La Metodología debe soportar la eventual evolución del sistema:** Normalmente durante su tiempo de vida los sistemas tienen muchas versiones, pudiendo durar incluso más de 10 años. Existen herramientas CASE para la gestión de la configuración y otras denominadas "Ingeniería inversa" para ayudar en el mantenimiento de los sistemas no estructurados, permitiendo estructurar los componentes de éstos facilitando así su mantenimiento.
13. **La Metodología debe contener actividades conducentes a mejorar el proceso de desarrollo de software:** Para mejorar el proceso es básico disponer de datos numéricos que evidencian la efectividad de la aplicación del proceso con respecto a cualquier producto software resultante del proceso.

    Para disponer de estos datos, la Metodología debe contener un conjunto de mediciones de proceso para identificar la calidad y coste asociado a cada etapa del proceso. Sería ideal el uso de herramientas CASE.

14. **La Metodología debe** permitir desarrollar software que satisfaga los requerimientos funcionales y no funcionales de la empresa PYME de microfinanzas MICREDITPERU, así como, los requerimientos generalmente aceptados de las PYMES de microfinanzas de la localidad de Trujillo – Perú.
15. **La Metodología debe** superar o mitigar las limitaciones de las metodologías de desarrollo de software actuales.

1.3. JUSTIFICACION Y DELIMITACION DE LA INVESTIGACION.

1.3.1. IMPORTANCIA Y JUSTIFICACION

1.3.1.1. Relevancia Organizacional

Debido a la imperiosa necesidad que tienen las Empresas de Microfinanzas del Perú, entre ellas la Empresas MICREDITPERU, de contar con un Software, confiable, que posibilite la comunicación, Integración de la Información, apoyando a los procesos de Créditos, Tesorería, Contabilidad y

principalmente que apoye en los procesos de Toma de Decisiones de la Gerencia General. Agilizándose significativamente la atención a los clientes contribuyendo en esa forma a acelerar la ejecución de sus proyectos.

1.3.1.2. UTILIDAD METODOLOGICA.

Valiéndonos de los avances en Tecnología de Información existentes, tal como la Herramienta CASE GENEXUS y de las mejores prácticas usadas en las Metodologías de Desarrollo de Software existentes, tanto tradicionales como las denominadas Agiles, se diseñará una Metodología Ágil para Desarrollo de Software apoyada en dicha herramienta CASE.

Es conveniente que la comunidad informática cuente con una Metodología Ágil de desarrollo de Software que se apoye en herramientas de última generación para llevar a cabo con eficacia y eficiencia las diversas etapas que conllevan el Desarrollo de Software.

1.3.1.3. VALOR TEORICO

Se dará a conocer como la Herramienta CASE GENEXUS apoya en las diferentes etapas de la Metodología Ágil para Desarrollo de Software propuesta en el presente trabajo de investigación, asegurando la Calidad del Software e incrementando su productividad.

1.3.1.4. IMPLICACIONES PRACTICAS.

Analistas de Créditos, los Clientes, Cajeros, Administradores y todas las personas involucradas en el uso de la Información en las empresas de Microfinanzas, serán los primeras beneficiarios al desarrollar Software usando la Metodología propuesta, ya que contarán con información oportuna, confiable y pertinente. Así también, disminuirá la dedicación de las personas encargadas actualmente de su elaboración, disponiendo así de mayor tiempo para dedicarse al análisis, a mejorar los procesos de créditos, a analizar a los clientes y a desarrollar nuevos productos. Se tendrá mejor información que apoyen a la mejor toma de decisiones en las diferentes áreas de la empresa.

1.3.2. DELIMITACION

En cuanto a la temática, en esta tesis, se responderá las preguntas siguientes:

¿Cómo definir una Metodología Ágil para Desarrollo de Software, teniendo como base las Metodologías y notaciones actualmente empleadas (Rational Unified Process, Extreme Programming, etc), que incorpore las mejores y más avanzadas prácticas existentes en cada etapa del desarrollo de Software?

¿Cómo la Herramientas CASE GENEXUS, apoya en las diferentes etapas de la Metodología Ágil de Desarrollo de Software a proponer?

¿Cómo la Metodología de Desarrollo de Software a definir, permitirá superar o mitigar las limitaciones de las metodologías de desarrollo de software actuales?

CAPITULO II

MARCO TEORICO

2.1. ANTECEDENTES

La comunidad de desarrolladores de software, motivados y podría decirse presionados, ante exigencias de los usuarios, tales como las de lograr el desarrollo de software comercial de calidad, en plazos acordes con las exigencias del mercado y de la competencia, siempre está en busca de métodos más eficientes de desarrollo de los productos software, esta búsqueda ha dado lugar al descubrimiento de nuevos paradigmas para el desarrollo de software, búsqueda que aún continua, en la medida de que el proceso de desarrollo de software no se asemeje al de las demás ingenierías.

A continuación haremos una breve descripción de las principales Metodologías actuales, así también, mencionaremos sus ventajas y desventajas.

2.1.1. PRINCIPALES METODOLOGÍAS TRADICIONALES:

2.1.1.1. PROCESO RATIONAL UNIFICADO – RUP:

Es un proceso para el desarrollo de un proyecto de un software que define claramente **quien**, **cómo**, **cuándo** y qué debe hacerse en el proyecto. Tiene 3 características esenciales; está dirigido por Casos de Uso: que orientan el proyecto a la importancia para el usuario y lo que este quiere, está centrado en la arquitectura: que Relaciona la toma de decisiones que indican cómo tiene que ser construido el sistema y en qué orden, y es iterativo e incremental: donde divide el proyecto en miniproyectos donde los casos de uso y la arquitectura cumplen sus objetivos de manera más depurada

PRINCIPIOS CLAVE DEL RUP:

Adaptación del proceso: El proceso deberá adaptarse a las características propias de la organización. El tamaño del mismo, así como las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

Balancear prioridades: Los requerimientos de los diversos inversores pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos.

Colaboración entre equipos: El desarrollo de software no lo hace una única persona sino múltiples equipos. Debe haber una comunicación fluida para coordinar requerimientos, desarrollo, evaluaciones, planes, resultados, etc.

Demostrar valor iterativamente: Los proyectos se entregan, aunque sea de un modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados

Elevar el nivel de abstracción: Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes 4GL o esquemas (frameworks) por nombrar algunos. Éstos se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con UML.

Enfocarse en la calidad: El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción

EL CICLO DE VIDA DE RUP

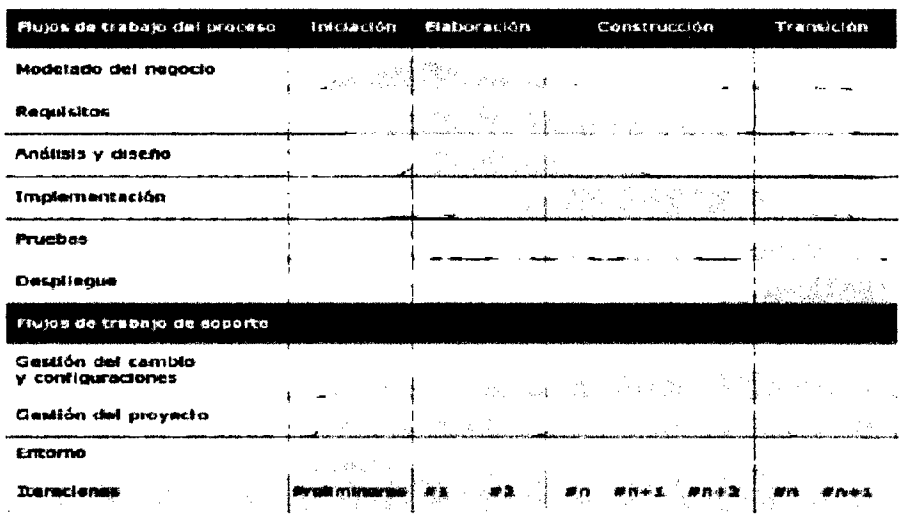
RUP divide el proceso en 4 fases (ver Fig. Nº 01), dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en los distintas actividades. [Booch, 1999]

En las iteraciones de cada fase se hacen diferentes esfuerzos en diferentes actividades

· **Iniciación**: Se hace un plan de fases, se identifican los principales casos de uso y se identifican los riesgos. Se define el alcance del proyecto.

· **Elaboración:** Se hace un plan de proyecto, se completan los casos de uso y se eliminan los riesgos.

Figura Nº 01. Fases e Iteraciones del RUP



Fuente:
http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

· **Construcción:** Se concentra en la elaboración de un producto totalmente operativo y eficiente y el manual de usuario.

· **Transición:** Se Instala el producto en el cliente y se entrena a los usuarios. Como consecuencia de esto suelen surgir nuevos requisitos a ser analizados.

2.1.1.2. METRICA – VERSION 3

METODOLOGIA DE PLANIFICACION, DESARROLLO Y MANTENIMIENTO DE SISTEMAS DE INFORMACION

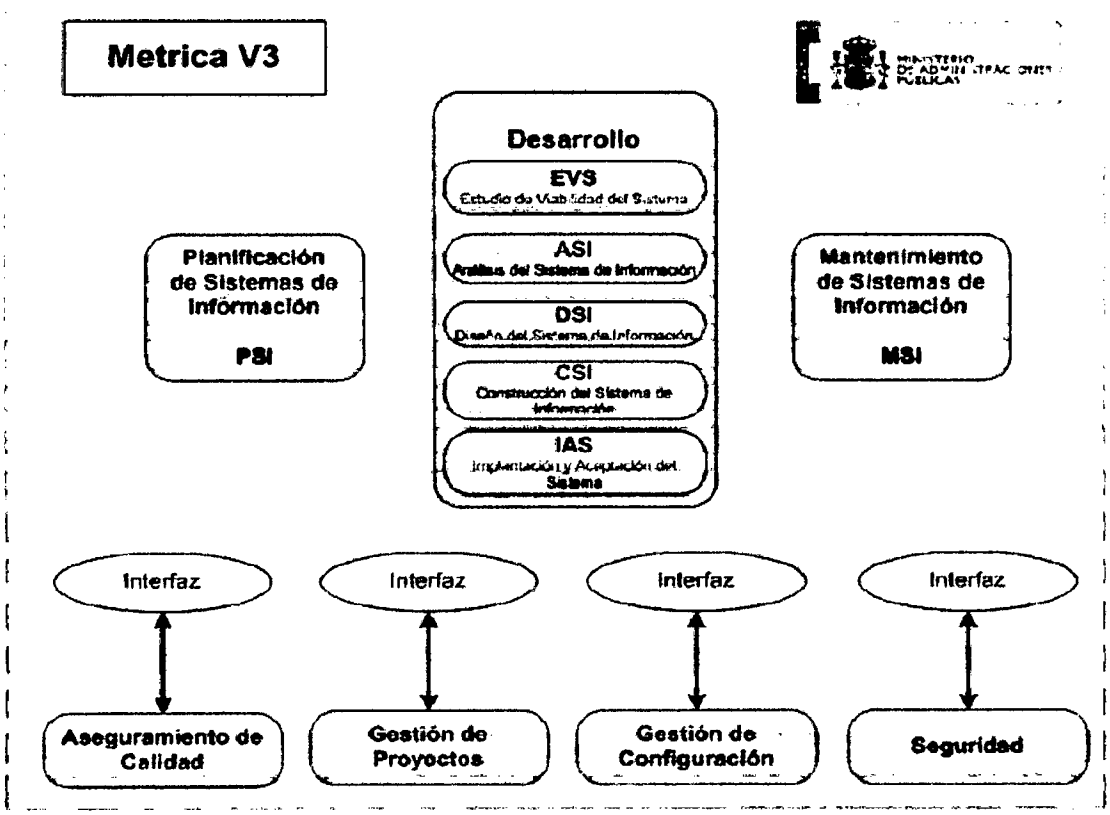
Métrica V3 es una Metodología de desarrollo elaborada por el Consejo Superior de Informática del Ministerio de Administraciones Públicas de España. Métrica tiene ya varios años de vida y su actual versión, la 3, se crea con la finalidad de incorporar las nuevas técnicas derivadas de la programación y el análisis orientado a objetos, al proceso de desarrollo de

software que a través de esta Metodología las administraciones públicas españolas pretenden llevar a cabo.

Todo proyecto Métrica 3 consta de un conjunto de fases que se desglosan en múltiples puntos cuya cronología hay que seguir con claridad para ir avanzando en el desarrollo del proyecto. La Figura 02. muestra el esquema general de estas fases:

En general, con Métrica 3 es posible gestionar proyectos orientados a objetos y proyectos estructurados. En tanto que la tendencia de la ingeniería de software actual lleva con claridad hacia los sistemas y herramientas cuya utilización está basada exclusivamente en el modelo de la orientación a objetos.

Figura Nº 02. Estructura del Métrica V3.



Fuente: Ministerio de Administración Electrónica – Gobierno de España

2.1.1.3. CMMI: CAPABILITY MATURITY MODEL INTEGRATION

CMMI, es un modelo para la mejora o evaluación de los procesos de desarrollo y **mantenimiento de sistemas** y **productos de software**. Fue desarrollado por el Instituto de Ingeniería del Software de la Universidad Carnegie Mellon (SEI), y publicado en su primera versión en enero de 2002.

Dos representaciones: Continua y escalonada (Ver Figuras 03 y 04 respectivamente)

Niveles de madurez

El "escalonado" CMM define 5 niveles posibles de madurez para las organizaciones que desarrollan y mantienen software.

Nivel 1: Inicial

Los resultados de calidad obtenidos son consecuencia de las personas y de las herramientas que emplean. No de los procesos, porque o no los hay o no se emplean.

Nivel 2: Repetible

Se considera un nivel 2 de madurez cuando se llevan a cabo prácticas básicas de gestión de proyectos, de gestión de requisitos, control de versiones y de los trabajos realizados por subcontratistas. Los equipos de los proyectos pueden aprovechar las prácticas realizadas para aplicarlas en nuevos proyectos.

Nivel 3: Definido

Los procesos comunes para desarrollo y mantenimiento del software están documentados de manera suficiente en una biblioteca accesible a los equipos de desarrollo. Las personas han recibido la formación necesaria para comprender los procesos.

Nivel 4: Gestionado

La organización mide la calidad del producto y del proceso de forma cuantitativa en base a métricas establecidas.

La capacidad de los procesos empleados es previsible, y el sistema de medición permite detectar si las variaciones de capacidad exceden los rangos aceptables para adoptar medidas correctivas.

Nivel 5: Optimizado

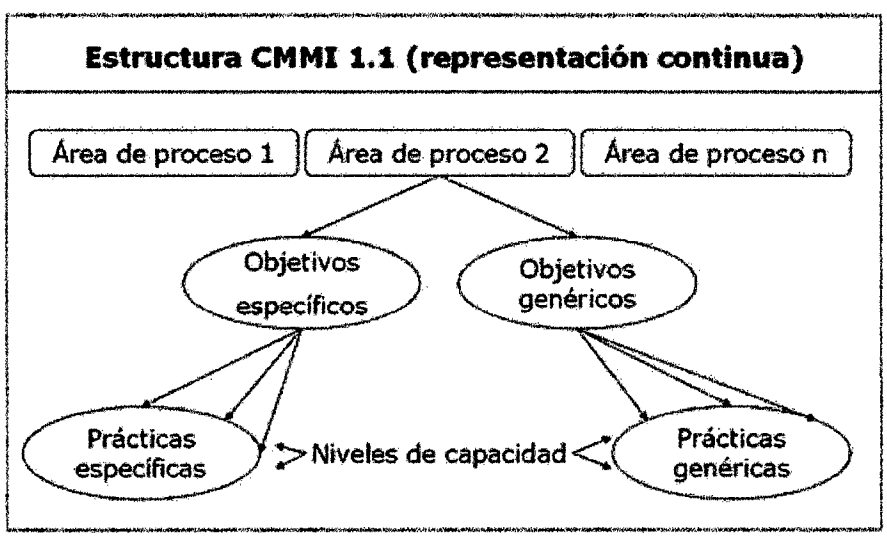
La mejora continua de los procesos afecta a toda la organización, que cuenta con medios para identificar las debilidades y reforzar la prevención de defectos. Se analizan de forma sistemática datos relativos a la eficacia de los procesos de software para analizar el coste y el beneficio de las adaptaciones y las mejoras.

Se analizan los defectos de los proyectos para determinar las causas, y su mapeado sobre los procesos.

COMPONENTES ESPERADOS

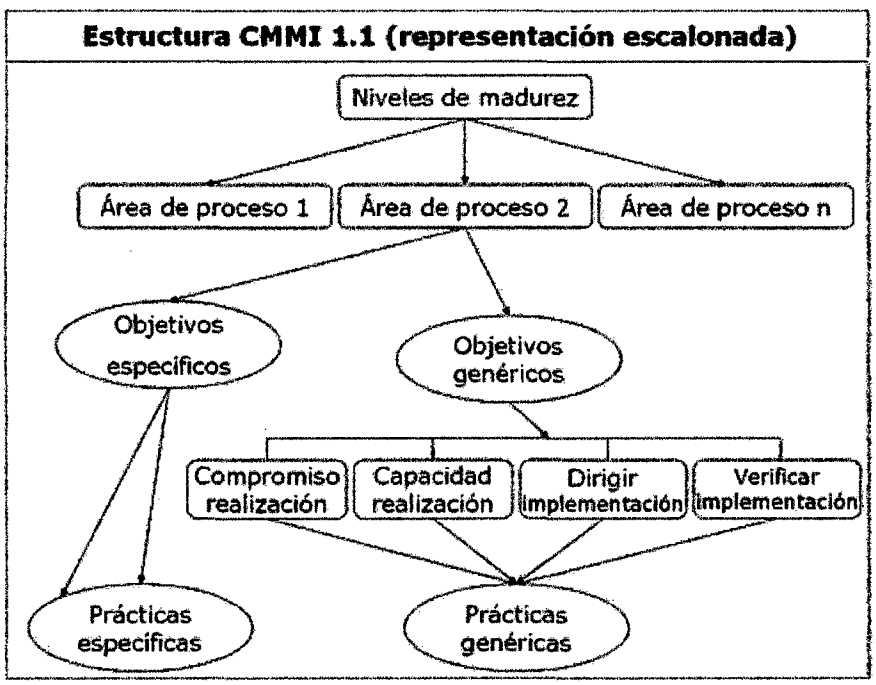
- **Práctica genérica**: Una práctica genérica se aplica a cualquier área de proceso porque puede mejorar el funcionamiento y el control de cualquier proceso.
- **Práctica específica**: Una práctica específica es una actividad que se considera importante en la realización del objetivo específico al cual está asociado. Las prácticas específicas describen las actividades esperadas para lograr la meta específica de un área de proceso.

Figura N°03. Representación Continua



Fuente: CMMI® for Development, Version 1.2 CMU/SEI-2006-TR-008 ESC-TR-2006-008, copyright 2006 by Carnegie Mellon University.

Figura Nº 04. Representación Escalonada



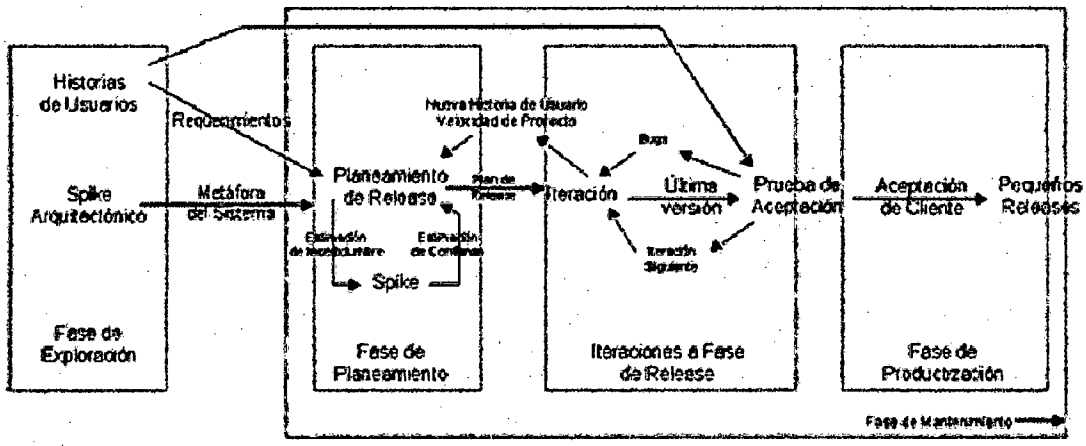
Fuente: CMMI® for Development, Version 1.2 CMU/SEI-2006-TR-008 ESC-TR-2006-008, copyright 2006 by Carnegie Mellon University.

2.1.2. PRINCIPALES METODOLOGÍA ÁGILES (AMs):

2.1.2.1. PROGRAMACION EXTREMA (Extreme Programming, XP): [Beck, K, 2000].

Es una Metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. El creador de la Metodología XP es Kent Beck , en la Figura Nº 05 se muestra el Ciclo de vida de XP.

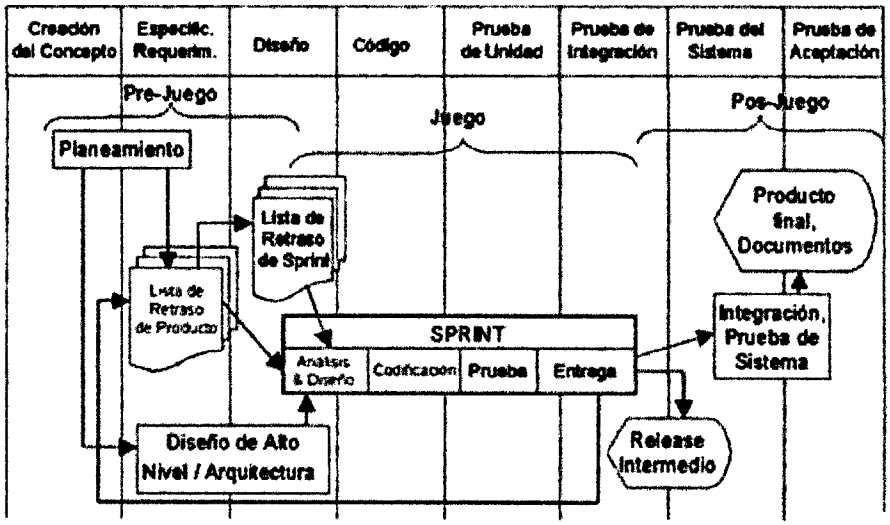
Figura Nº 05. Ciclo de Vida de XP.



Fuente: Kent Beck, "RE: (OTUG) XP and Documentation". Rational's Object Technology User Group Mailing List, 23 de Marzo de 1999.

2.1.2.2. SCRUM [Ken Schwaber, 2004] Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Esta especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se desarrolla mediante iteraciones, denominadas sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo del proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración. En la Figura Nº 06 se muestra el ciclo de vida del SCRUM.

Figura Nº 06. Ciclo de Vida de Metodología SCRUM



Fuente: Pekka Abrahamsson. "Agile Software development methods: A minitutorial". VTT Technical Research Centre of Finland, http://www.vtt.fi/virtual/agile/seminar2002/Abrahamsson_agile_methods_minitutorial.pdf, 2002.

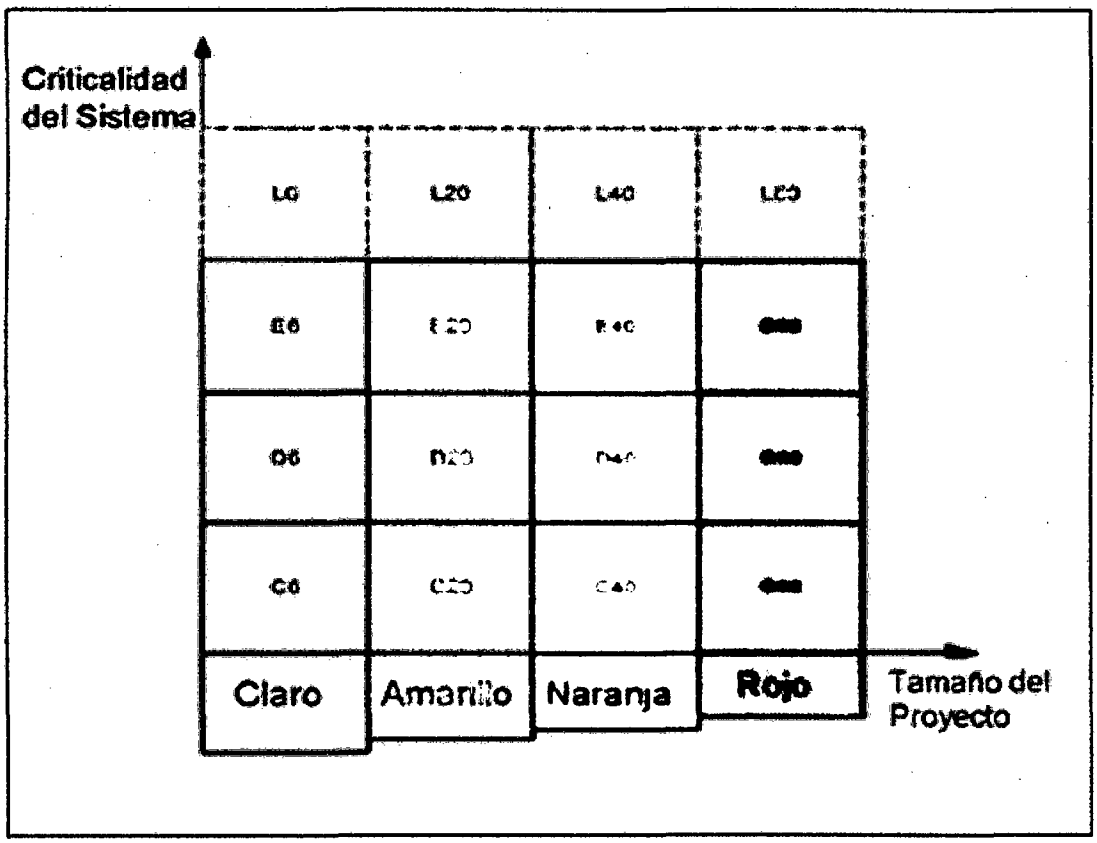
2.1.2.3. CRYSTAL METODOLOGIES: [Alistar Cockburn and Jim Highsmith, 2004]

Son un conjunto de Metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. Han sido desarrolladas por Alistair Cockburn. El desarrollo de software es considerado como un juego cooperativo de invención y cooperación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Esta políticas dependerán del tamaño del equipo, estableciéndose un clasificación por colores, por ejemplo Crystal Clear (3 a 8 miembros) y Crystal Orange (25 a 50 miembros). La Figura Nº 07, muestra esta clasificación.

El proceso de Cristal Clear (CC) se basa en una exploración refinada de los inconvenientes de los modelos clásicos. Dice Cockburn que la mayoría de los modelos de proceso propuestos entre 1970 y 2000 se

describían como secuencias de pasos. Aún cuando se recomendaran iteraciones e incrementos (que no hacían más que agregar confusión a la interpretación) los modelos parecían dictar un proceso en cascada, por más que los autores aseguraran que no era así. El problema con estos procesos es que realmente están describiendo un workflow requerido, un grafo de dependencia: el equipo no puede entregar un sistema hasta que está integrado y corre. No puede integrar y verificar hasta que el código no está escrito y corriendo. Y no puede diseñar y escribir el código hasta que se le dice cuáles son los requerimientos. Un grafo de dependencia se interpreta necesariamente en ese sentido, aunque no haya sido la intención original.

Figura Nº 07. Clasificación de Metodogias Crystal por colores

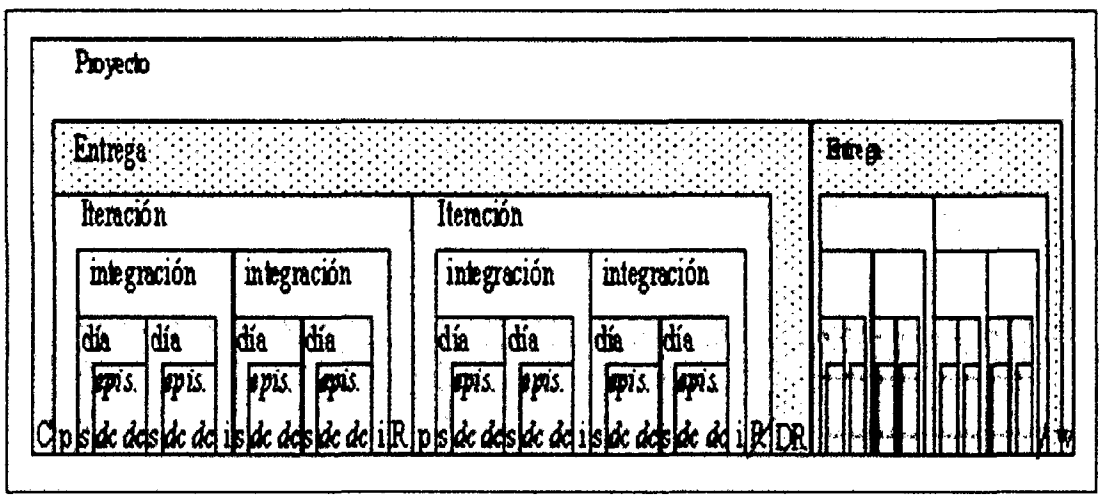


Fuente: Pekka Abrahamsson. "Agile Software development methods: A minitutorial". VTT Technical Research Centre of Finland. http://www.vtt.fi/virtual/agile/seminar2002/Abrahamsson_agile_methods_minitutorial.pdf, 2002.

En lugar de esta interpretación lineal, CC enfatiza el proceso como un conjunto de ciclos anidados. En la mayoría de los proyectos se perciben

siete ciclos: (1) el proyecto, (2) el ciclo de entrega de una unidad, (3) la iteración (nótese que CC requiere múltiples entregas por proyecto pero no muchas iteraciones por entrega), (4) la semana laboral, (5) el período de integración, de 30 minutos a tres días, (6) el día de trabajo, (7) el episodio de desarrollo de una sección de código, de pocos minutos a pocas horas. Ver Figura Nº 08.

Figura Nº 08. Ciclos de un proyecto Crystal Metodologies



Fuente: Pekka Abrahamsson. "Agile Software development methods: A minitutorial". VTT Technical Research Centre of Finland. http://www.vtt.fi/virtual/agile/seminar2002/Abrahamsson_agile_methods_minitutorial.pdf, 2002.

Cockburn subraya que interpretar no linealmente un modelo de ciclos es difícil; la mente se resiste a hacerlo. Él mismo asegura que estuvo diez años tratando de explicar el uso de un proceso cíclico y sólo recientemente ha logrado intuir cómo hacerlo. La figura muestra los ciclos y las actividades conectadas a ellos. Las letras denotan Chartering, planeamiento de iteración, reunión diaria de pie (standup), desarrollo, check-in, integración, taller de Reflexión, Entrega (Delivery), y empaquetado del proyecto (Wrapup).

Los métodos Crystal no prescriben las prácticas de desarrollo, las herramientas o los productos que pueden usarse, pudiendo combinarse con otros métodos como Scrum, XP y Microsoft Solutions Framework.

2.1.2.3.1. DYNAMIC SYSTEMS DEVELOPMENT METHOD (DSDM):

[Stapleton J., 1997] Define el marco para desarrollar un proceso de producción de software. Nace en 1994 con el objetivo de crear una Metodología RAD (Rapid application development) unificada. Sus principales características son: Iterativo e incremental y el equipo de desarrollo y usuarios trabajan juntos. Propone cinco fases: Estudio de viabilidad, estudio del negocio, modelado funcional, diseño y construcción y finalmente implementación. Las tres últimas son iterativas, además de existir realimentación en todas sus fases.

En Febrero de 1995 DSDM Consortium liderado por Tony Mobbs, Jennifer Stapleton, gary Hosdon, Paul Herizlich y Peter Constable, publicó la 1era. Versión de DSDM.

La versión actual es la 4.1 y es el método más usado en el Reino Unido y va extendiéndose por Europa y Estados Unidos.

La idea dominante detrás de DSDM es explícitamente inversa a la que se encuentra en otras partes, y al principio resulta contraria a la intuición; en lugar de ajustar tiempo y recursos para lograr cada funcionalidad, en esta Metodología tiempo y recursos se mantienen como constantes y se ajusta la funcionalidad de acuerdo con ello. Esto se expresa a través de reglas que se conocen como "reglas MoSCoW" por las iniciales de su estipulación en inglés. Las reglas se refieren a rasgos del requerimiento:

1. Must have: Debe tener. Son los requerimientos fundamentales del sistema, de estos, el subconjunto mínimo ha de ser satisfecho por completo.
2. Should have: Debería tener. Son requerimientos importantes para los que habrá una resolución en el corto plazo.
3. Could have: Podría tener. Podrían quedar fuera del sistema si no hay más remedio.

4. Want to have but won't have this time around: Se desea que tenga, pero no lo tendrá esta vuelta. Son requerimientos valorados, pero pueden esperar.

MSDM consiste de cinco fases:

1. Estudio de viabilidad.
2. Estudio del negocio.
3. Iteración del modelo funcional.
4. Iteración de diseño y versión.
5. Implementación.

Las últimas tres fases son iterativas e incrementales. De acuerdo con la iniciativa de mantener el tiempo constante, las iteraciones de DSDM son cajas de tiempo. La iteración acaba cuando el tiempo se consume. Se supone que al cabo de la iteración los resultados están garantizados. Una caja de tiempo puede durar de unos pocos días a unas pocas semanas.

A diferencia de otros AMs, DSDM ha desarrollado sistemáticamente el problema de su propia implantación en una empresa. El proceso de Examen de Salud (Health Check) de DSDM se divide en dos partes que se interrogan, sucesivamente, sobre la capacidad de una organización para adoptar el método y sobre la forma en que éste responde a las necesidades una vez que el proyecto está encaminado. Un Examen de Salud puede insumir entre tres días y un mes de trabajo de consultoría.

1. *Estudio de factibilidad.* Se evalúa el uso de DSDM o de otra Metodología conforme al tipo de proyecto, variables organizacionales y de personal. Si se opta por DSDM, se analizan las posibilidades técnicas y los riesgos. Se preparan como productos un reporte de viabilidad y un plan sumario para el desarrollo. Si la tecnología no se conoce bien, se hace un pequeño prototipo para ver qué pasa. No se espera que el estudio completo insuma más de unas pocas semanas. Es mucho para un método ágil, pero menos de lo que demandan algunos métodos clásicos.

2. *Estudio del negocio.* Se analizan las características del negocio y la tecnología. La estrategia recomendada consiste en el desarrollo de talleres, donde se espera que los expertos del cliente consideren las facetas del sistema y acuerden sus prioridades de desarrollo. Se describen los procesos de negocio y las clases de usuario en una Definición del Área de Negocios. Se espera así reconocer e involucrar a gente clave de la organización en una etapa temprana. La Definición utiliza descripciones de alto nivel, como diagramas de entidad-relación o modelos de objetos de negocios. Otros productos son la Definición de Arquitectura del Sistema y el Plan de Bosquejo de Prototipado. La definición arquitectónica es un primer bosquejo y se admite que cambie en el curso del proyecto DSDM. El plan debe establecer la estrategia de prototipado de las siguientes etapas y un plan para la gestión de configuración.

3. *Iteración del modelo funcional.* En cada iteración se planea el contenido y la estrategia, se realiza la iteración y se analizan los resultados pensando en las siguientes. Se lleva a cabo tanto el análisis como el código; se construyen los prototipos y en base a la experiencia se mejoran los modelos de análisis. Los prototipos no han de ser descartados por completo, sino gradualmente mejorados hacia la calidad que debe tener el producto final. Se produce como resultado un Modelo Funcional, conteniendo el código del prototipo y los modelos de análisis. También se realizan pruebas constantemente. Hay otros cuatro productos emergentes: (1) Funciones Priorizadas es una lista de funciones entregadas al fin de cada iteración; (2) los Documentos de Revisión del Prototipado Funcional reúnen los comentarios de los usuarios sobre el incremento actual para ser considerados en iteraciones posteriores; (3) los Requerimientos Funcionales son listas que se construyen para ser tratadas en fases siguientes; (4) el Análisis de Riesgo de Desarrollo Ulterior es un documento importante en la fase de iteración del modelo, porque desde la fase siguiente en adelante los problemas que se encuentren serán más difíciles de tratar.

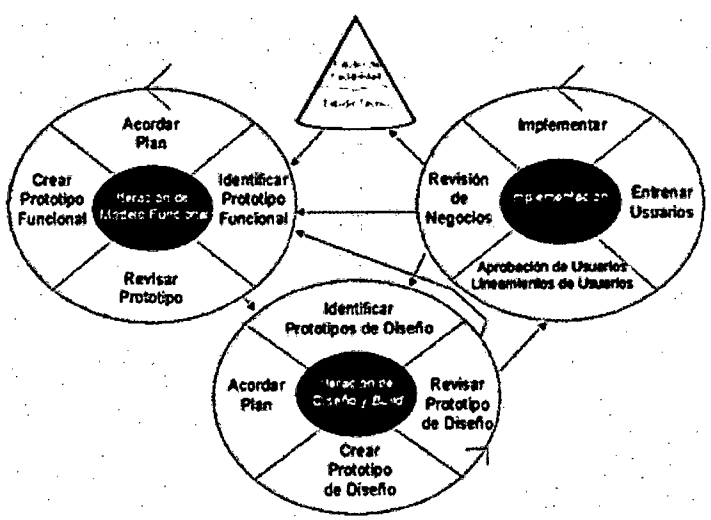
4. *Iteración de diseño y construcción.* Aquí es donde se construye la mayor parte del sistema. El producto es un Sistema Probado que

cumplimenta por lo menos el conjunto mínimo de requerimientos acordados conforme a las reglas MoSCoW. El diseño y la construcción son iterativos y el diseño y los prototipos funcionales son revisados por usuarios. El desarrollo ulterior se atiene a sus comentarios.

5. *Despliegue*. El sistema se transfiere del ambiente de desarrollo al de producción. Se entrena a los usuarios, que ponen las manos en el sistema. Eventualmente la fase puede llegar a iterarse. Otros productos son el Manual de Usuario y el Reporte de Revisión del Sistema. A partir de aquí hay cuatro cursos de acción posibles: (1) Si el sistema satisface todos los requerimientos, el desarrollo ha terminado. (2) Si quedan muchos requerimientos sin resolver, se puede correr el proceso nuevamente desde el comienzo. (3) Si se ha dejado de lado alguna prestación no crítica, el proceso se puede correr desde la iteración funcional del modelo en adelante. (4) si algunas cuestiones técnicas no pudieron resolverse por falta de tiempo se puede iterar desde la fase de diseño y construcción.

La configuración del ciclo de vida de DSDM se representa en el diagrama que se muestra en la Figura Nº 09.

Figura Nº 09: Configuración del Ciclo de vida del DSDM.

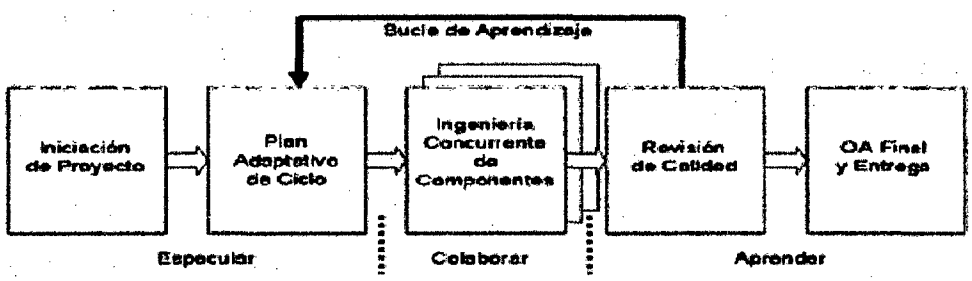


Fuente: *http://www.dsdm.org*

2.1.2.4. ADAPTATIVE SOFTWARE DEVELOPMENT (ASD):

[Highsmith Jr., Orr K., 2000], Su impulsor es Jim Highsmith. Sus principales características son: iterativo, orientado a los componentes de software más que a las tareas y tolerante a los cambios. El ciclo de vida que propone tiene tres fases esenciales: Especulación, colaboración y aprendizaje (Ver Fig. Nº 10). En la primera de ellas se inicia el proyecto y se planifican las características del software; en la segunda desarrollan las características y finalmente en la tercera se revisa su calidad y se entrega al cliente. La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.

Figura Nº 10. Fases del ciclo del ASD



Fuente: Jim Highsmith. Adaptive software development: A collaborative approach to managing complex systems. Nueva York, Dorset House, 2000.

Aspectos claves de ASD:

1. Un conjunto no estándar de "artefactos de misión" (documentos para tí y para mí), incluyendo una visión del proyecto, una hoja de datos, un perfil de misión del producto y un esquema de su especificación

2. Un ciclo de vida, inherentemente iterativo.

3. Cajas de tiempo, con ciclos cortos de entrega orientados por riesgo.

Un ciclo de vida es una iteración; este ciclo se basa en componentes y no en tareas, es limitado en el tiempo, orientado por riesgos y tolerante al cambio. Que se base en componentes implica concentrarse en el desarrollo de software que trabaje, construyendo el sistema pieza por pieza. En este paradigma, el cambio es bienvenido y necesario, pues se concibe como la oportunidad de aprender y ganar así una ventaja

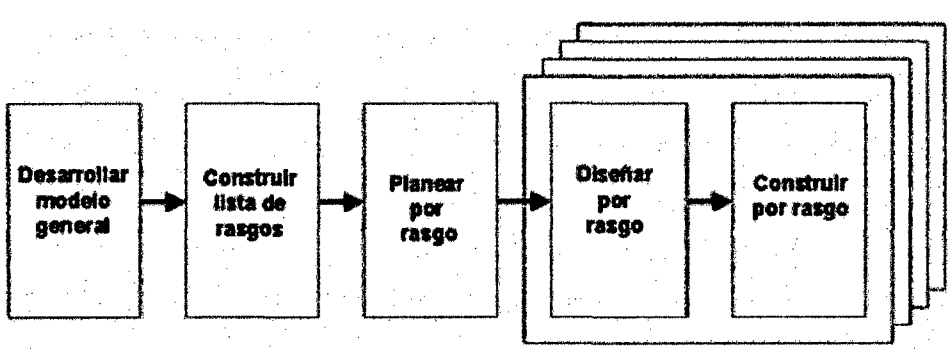
competitiva; de ningún modo es algo que pueda ir en detrimento del proceso y sus resultados.

2.1.2.5. FEATURE-DRIVEN DEVELOPMENT (FDD):

[Stepen Palmer, 2002] Define un proceso iterativo que consta de cinco pasos. Las iteraciones son cortas (hasta 2 semanas). Se centra en las fases del diseño e implementación del sistema partiendo de una lista de características que debe reunir el software Sus impulsores son Jeff de Luca y Peter Coad.

Los cinco roles de soporte comprenden (1) administrador de entrega, que controla el progreso del proceso revisando los reportes del programador jefe y manteniendo reuniones breves con él; reporta al manager del proyecto; (2) abogado/guru de lenguaje, que conoce a la perfección el lenguaje y la tecnología; (3) ingeniero de construcción, que se encarga del control de versiones de los builds y publica la documentación; (4) herramientista (toolsmith), que construye herramientas ad hoc o mantiene bases de datos y sitios Web y (5) administrador del sistema, que controla el ambiente de trabajo o "productiviza" el sistema cuando se lo entrega. (Ver Fig. Nº 11)

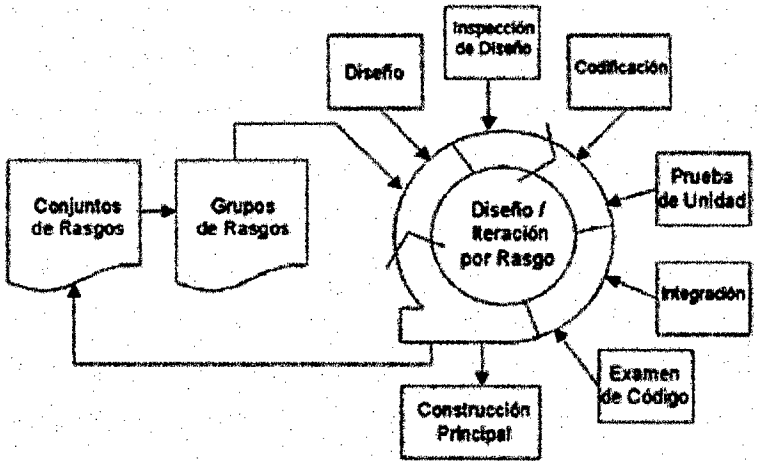
Figura Nº 11. Proceso FDD



Fuente: *http://togethercommunities.com*

Los tres roles adicionales son los de verificadores, encargados del despliegue y escritores técnicos. Un miembro de un equipo puede tener otros roles a cargo, y un solo rol puede ser compartido por varias personas. (ver Fig. Nº 12).

Figura Nº 12. Ciclo FDD



Fuente: Pekka Abrahamsson, Outi Salo, Jussi Ronkainen y Juhani Warsta. "Agile Software Development Methods". VTT Publications 478, Universidad de Oulu, Suecia, 2002.

2.1.2.6. LEAN DEVELOPMENT (LD):

[Poppendieck M., Poppendieck T, 2003] Definida por Bob Charette's a partir de su experiencia en proyectos con la industria japonesa del automóvil en los años 80 y utilizada en numerosos proyectos de telecomunicaciones en Europa. En LD, los cambios se consideran riesgos, pero si se manejan adecuadamente se pueden convertir en oportunidades que mejoren la productividad del cliente. Su principal característica es introducir un mecanismo para implementar dichos cambios.

Dado que LD es más una filosofía de management que un proceso de desarrollo no hay mucho que decir del tamaño del equipo, la duración de las iteraciones, los roles o la naturaleza de sus etapas. Últimamente LD ha evolucionado como Lean Software Development (LSD); su figura de referencia es Mary Poppendieck.

Uno de los sitios primordiales del modelo son las páginas consagradas a LSD que mantiene Darrell Norton [Nor04], donde se promueve el desarrollo del método aplicando el framework .NET de Microsoft. Norton ha reformulado los valores de Charette reduciéndolos a

siete y suministrando más de veinte herramientas análogas a patrones organizacionales para su implementación en ingeniería de software.

Otra preceptiva algo más amplia es la de Mary Poppendieck, cuidadosamente decantadas del Lean Manufacturing y de Total Quality Management (TQM), que sólo coincide con la de Norton en algunos puntos:

Aunque la formulación del método es relativamente reciente, la familiaridad de muchas empresas con los principios de Lean Production & Lean Manufacturing ha facilitado la penetración en el mercado de su análogo en ingeniería de software. LD se encuentra hoy en América del Norte en una situación similar a la de DSDM en Gran Bretaña, llegando al 7% entre los MAs a nivel mundial (algo menos que Crystal pero el doble que Scrum). Existen abundantes casos de éxito documentados empleando LD y LSD, la mayoría en Canadá. Algunos de ellos son los de Canadian Pacific Railway, Autodesk y PowerEx Corporation. Se ha aplicado prácticamente a todo el espectro de la industria.

2.1.2.7. VENTAJAS DE LAS METODOLOGIAS AGILES:

a. Rápida respuesta a cambios de requisitos a lo largo del desarrollo.
b. Entrega continua y en plazos cortos de software funcional.
c. Trabajo conjunto entre el cliente y el equipo de desarrollo.
d. Minimiza los costos frente a cambios.
e. Importancia de la simplicidad, al eliminar el trabajo innecesario.
f. Atención continua a la excelencia técnica y al buen diseño.
g. Mejora continua de los procesos y el equipo de desarrollo.
h. Evita malentendidos de requerimientos entre el cliente y el equipo.
i. El equipo de desarrollo no malgasta el tiempo y dinero del cliente desarrollando soluciones innecesariamente generales y complejas que en realidad no son un requisito del cliente.
j. Cada componente del producto final ha sido probado y satisface los requerimientos.

2.1.2.8. DESVENTAJAS DE LAS METODOLOGIAS AGILES:

Las Metodologías Agiles, como en cualquiera otra metodología, también presentan desventajas y problemas que se evidencian durante su implementación:

a. Falta de documentación del diseño. El código no puede tomarse como una documentación. En sistemas de tamaño grande se necesitar leer los cientos o miles de páginas del listado de código fuente.

b. Problemas derivados de la comunicación oral. Este tipo de comunicación resulta difícil de preservar cuando pasa el tiempo y está sujeta a muchas ambigüedades.

c. Falta de calidad. Probar el código de forma constante no genera productos de calidad, sólo revela falta de análisis y diseño.

d. Fuerte dependencia de las personas. Como se evita en lo posible la documentación y los diseños convencionales, los proyectos ágiles dependen críticamente de las personas.

e. Falta de procesos de revisión del código. Con métodos como el Personal Software Process (PSP) o Team Software Process (TSP) se han conseguido reducciones de errores que oscilan entre el 60 y el 80%. La programación en parejas tiene resultados del 20 al 40%, que no es mucho frente al 10 y el 25% de un programador.

f. Falta de reusabilidad. La falta de documentación hacen difícil que el código ágil pueda reutilizarse.

g. Sobre costos y retrasos derivados de la refactorización continua. Para un sistema de ciertas proporciones, los costos y retrasos derivados de la refactorización no pueden despreciarse.

h. Restricciones en cuanto a tamaño de los proyectos abordables.

i. Rigidez. Algunos métodos ágiles son muy rígidos: deben cumplirse muchas reglas de una forma estricta para garantizar el éxito del proyecto. Por ejemplo XP exige en realidad mucho esfuerzo, concentración y orden.

j. Cambios. Los modelos de datos son "pesados" y no pueden cambiarse así como así solo porque el cliente que va a incorporar más funciones al sistema.

k. Problemas derivados del fracaso de los proyectos ágiles. Si un proyecto ágil fracasa no hay documentación o hay muy poca; lo mismo ocurre con el diseño. La comprensión del sistema se queda en las mentes de los desarrolladores.

2.1.3. LOS METODOS FORMALES DE DESARROLLO DE SOFTWARE.

Según [Pressman, Roger S., 2005], Los Métodos Formales, permiten que un Ingeniero de Software cree una especificación más completa, consistente y precisa que los que se producen empleando los métodos convencionales. Se utiliza la notación de teoría de conjuntos y lógica para crear un claro planteamiento de los hechos (requisitos). Esta especificación matemática luego se analiza para mejorar (e incluso probar) su corrección y consistencia. Puesto que la especificación se crea mediante notación matemática, es inherentemente menos ambigua que los modos informales de representación.

Las especificaciones formales las crean los Ingenieros de Software especialmente entrenados.

En los Sistemas de seguridad o de Misión Crítica, las fallas tienen un precio elevado, cuando el software de computadora falla es posible que se pierdan vidas o surjan graves consecuencias económicas. En tales circunstancias es esencial que los errores sean descubiertos antes que el software sea puesto en operación. Los métodos formales reducen sustancialmente los errores de especificación y como consecuencia, sirven como base para que el software tenga tan pocos errores una vez que el cliente empiece a usarlo.

Los pasos que se siguen para crear una especificación formal son: La heurística de conjuntos y la especificación constructiva – Operadores de conjuntos, operadores lógicos y de secuencias – forman la base de los métodos formales. Estos definen los datos invariantes, estados y

operaciones para la función de un sistema al traducir los requisitos formales del problema en una representación más formal.

La aplicación eficaz de los métodos formales requiere que se tenga un conocimiento operativo de la notación matemática asociada con los conjuntos y las sucesiones y de la notación lógica utilizada en el cálculo de predicados.

Aunque las técnicas formales de especificación no se emplean con amplitud en la industria, si ofrecen ventajas sustanciales sobre las técnicas menos formales. [Liskov, B.H. y V. Berzins , 1986] "An appraisal of program especifications", en software specificatrions techniques, N. Gehani y A.T. Mackkittrik (eds). Addison – Wesley, 1986, p3.

Resumiendo podemos afirma que los métodos formales ofrecen un cimiento para los entornos de especificación que conducen a modelos de análisis más completos, consistentes y sin ambigüedades que aquellos producidos por métodos convencionales u orientados a objetos. Las facilidades descriptivas de la teoría de conjuntos y la notación lógica permiten que un Ingeniero de software cree un planteamiento claro de hechos (requisitos).

Los conceptos subyacentes que rigen a los métodos formales son: 1) *El Invariante de Datos*, una condición verdadera a través de la condición del sistema que contiene una colección de datos. 2) *El Estado*, una representación del modo de comportamiento observable externamente de un sistema, o (en lenguaje $Z^{(2)}$ o en lenguajes relacionados) los datos almacenado a los que un sistema tiene acceso y altera; y 3) *La Operación* una acción, que tiene lugar en un sistema y lee o escribe datos a un *Estado.* Una Operación está asociada con dos condiciones: Una *Precondición* y una *Postcondición.* La matemáticas discretas – las heurísticas asociadas con conjuntos y la especificación constructiva, operadores de conjuntos, Operados lógicos y sucesiones – forman la base de los métodos formales. Las matemáticas discretas se implementan en el contexto de los lenguajes formales de especificación, tales como $OCL^{(1)}$ y $Z^{(2)}$. Estos lenguajes tienen dominio tanto sintáctico como semántico. El dominio sintáctico utiliza una

simbología alineada de manera cercana con la notación de conjuntos y el cálculo de predicados. El dominio semántico permite que el lenguaje exprese los requisitos de una forma concisa.

La decisión de usar métodos formales debe considerar los costos de arranque, así como los cambios culturales asociados con una tecnología radicalmente diferente.

En la mayoría de las instancias, los métodos formales tienen mayores rendimientos respecto de los sistemas cruciales para la seguridad y los negocios. [Liskov, B.H. y V. Berzins , 1986], resumen esto así: *Las especificaciones formales se pueden estudiar matemáticamente, pero no las informales. Por ejemplo, un programa correcto se prueba para satisfacer sus especificaciones, o se puede probar que dos conjuntos alternativos de especificaciones son equivalentes ... Ciertas formas de incompletud o inconsistencias se pueden probar automáticamente.*

Además la especificación formal elimina la ambigüedad y alienta el mayor rigor en las primeras etapas del proceso de ingeniería de software.

No obstante, los problemas persisten, la especificación formal se enfoca principalmente en la función y los datos. La temporalidad, el control y los aspectos de comportamiento de un problema son más difíciles de representar. Además ciertos elementos de un problema (por ejemplo humano/máquina) se especifican mejor empleando técnicas gráficas o prototipos. Finalmente, la especificación que emplea métodos formales es más difícil de aprender que los métodos que incorporan notación UML y representan un significativo "choque cultural" para algunos profesionales del software.

[1]OCL (siglas en ingles de Lenguaje restringido a Objetos), es una notación formal desarrollada de modo que los usuarios de UML puedan conferirle mayor precisión a sus especificaciones. El lenguaje dispone de toda la lógica y las matemáticas discretas. Sin embargo los diseñadores de OCL decidieron que en este lenguaje solo deberían utilizarse caracteres ASCII (en lugar de notación matemática convencional). Esto permite que el lenguaje sea más asequible a las personas menos inclinadas a las

matemáticas y que el procesador los procese con mayor facilidad. Pero esto también favorece que OCL se apoco farragoso (o ininteligible) en algunos lugares.

[2]Z es un lenguaje de especificación que ha evolucionado durante dos décadas pasadas y hoy se utiliza ampliamente entre la comunidad de de los métodos formales. El lenguaje Z aplica conjuntos tipificados relaciones y funciones dentro del contexto de predicados lógicos de primer orden para construir *esquemas,* un medio para estructurar una especificación formal.

2.1.4. LAS HERRAMIENTAS CASE:

La utilización de una herramienta CASE es uno de los aportes más importantes en el campo de la Ingeniería de software. Aunque el concepto fue abordado desde hace tres décadas, son los productos que se encuentran hoy en el mercado los que han provocado una revolución por el aumento que propician en la eficiencia, productividad y calidad del producto software final.

Las herramientas CASE. son una tecnología para automatizar el desarrollo y mantenimiento del software, combinando herramientas de software y Metodologías. Estas herramientas deben constituir un conjunto integrado que automatice todas las partes del ciclo de vida y por tanto ahorren trabajo.

Las facilidades que brindan para revisar especificaciones de un sistema, diagnosticar errores cometidos, desarrollar prototipos al generar parte o completamente una aplicación a partir de especificaciones, y el soporte para el mantenimiento como resultado de haber guardado las especificaciones del sistema en un depósito central de datos; son razones suficientes que justifican la importancia del desarrollo de estas herramientas acompañadas de las Metodologías y métodos.

A partir de especificaciones de diseño, las herramientas generadoras de código pueden generar un esquema o un programa completo.

Automatizan aspectos claves del diseño y desarrollo de sistemas. Su propósito primordial es proporcionar a los analistas y programadores de sistemas, una alternativa de desarrollo, apoyando sus actividades con una herramienta accesible y completamente funcional de acuerdo a sus necesidades.

Con el paso de los años el CASE, parte del tejido de la Ingeniería de Software. Al igual que los Ingenieros Mecánicos Eléctricos se basan en le CAD/CAE/SIM para el diseño de productos de alta tecnología, los Ingenieros de Software se basarán en el CASE. para el análisis el diseño y las pruebas de los sistemas basados en computadoras del siglo veintiuno [Pressman, 2005].

De acuerdo con Kendall y Kendall la ingeniería de sistemas asistida por ordenador es la aplicación de tecnología informática a las actividades, las técnicas y las Metodologías propias de desarrollo, su objetivo es acelerar el proceso para el que han sido diseñadas, en el caso de CASE para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas. [Kendall y Kendall, 1999].

La tecnología CASE está experimentando un gran avance en distintas áreas, solventando bastante de los defectos que presentaba la primera generación de estos productos. Sin embargo, el principal desafío de esta tecnología sigue siendo su adopción por parte de los profesionales y las empresas. [Piattini, Daryanani, 1995].

Las principales herramientas CASE son:

2.1.4.1. RATIONAL ROSE ENTERPRISE

IBM Rational Rose Enterprise es uno de los productos más completos de la familia Rational Rose. Todos los productos de Rational Rose dan soporte a Unified Modeling Language (UML), pero no son compatibles con las mismas tecnologías de implementación.

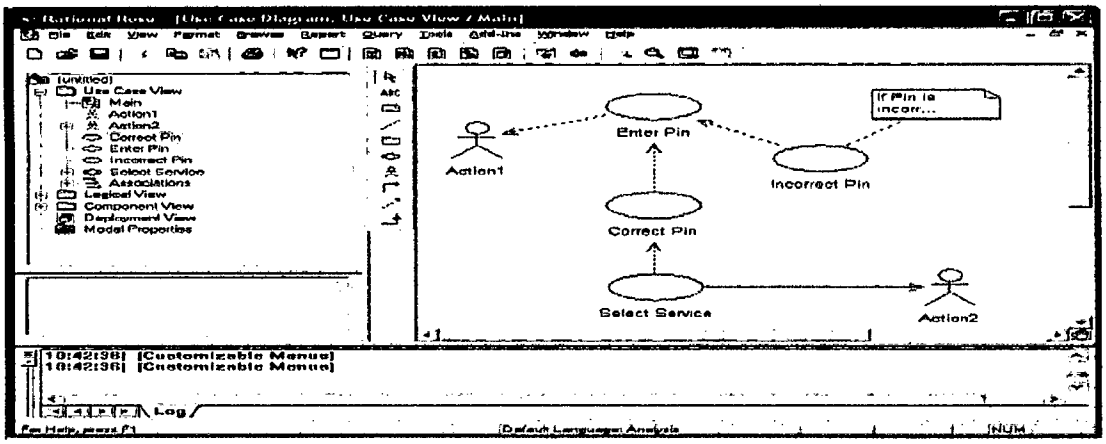
Rational Rose Enterprise es un entorno de modelado que permite generar código a partir de modelos Ada, ANSI C++, C++, CORBA,

Java/J2EE, Visual C++ y Visual Basic. Al igual que todos los productos de Rational Rose, ofrece un lenguaje de modelado común que agiliza la creación del software.

Incluye también estas funciones:

- Soporte a modelos de análisis, ANSI C++, Rose J y Visual C++ según el documento "Design Patterns: Elements of Reusable Object-Oriented Software".
- Los componentes del modelo se pueden controlar independientemente, lo que permite una gestión y un uso de modelos más granular.
- NUEVO: Soporte para compilación y descompilación de las construcciones más habituales de Java 1.5.
- Generación de código en lenguaje Ada, ANSI C++, C++, CORBA, Java y Visual Basic, con funciones configurables de sincronización entre los modelos y el código.
- Soporte para Enterprise Java Beans 2.0.
- Funciones de análisis de calidad de código.
- Complemento de modelado Web que incluye funciones de visualización, modelado y herramientas para desarrollar aplicaciones Web.
- Modelado en UML para diseñar bases de datos, que integra los requisitos de datos y aplicaciones mediante diseños lógicos y analíticos.
- Creación de definiciones de tipo de documento DTD en XML.
- Integración con otras herramientas de desarrollo de IBM Rational.
- Integración con cualquier sistema de control de versiones compatible con SCC, como IBM Rational ClearCase.
- Posibilidad de publicar en la Web modelos e informes para mejorar la comunicación entre los miembros del equipo. Ver Paneles en Figura Nº 13.

Figura Nº 13 Paneles del Rational Rose



Fuente: IBM Rational Rose

2.1.4.2. ENTERPRISE ARCHITECT (EA)

Si necesita modelar y gestionar información compleja, diseñar y visualizar software, o construir y desplegar diversos sistemas, Enteprise Architect es la solución perfecta. Proporciona un entorno de modelización de carácter colaborativo y potenciado mediante UML 2.1, abarca por completo el ciclo de vida de desarrollo de software, con herramientas que le proporcionan una infraestructura enormemente competitiva en torno a la modelización de negocio, diseño de software, ingeniería de sistemas, arquitectura corporativa, gestión de requerimientos, pruebas y mucho más.

PERSPECTIVA GENERAL

Eficaz - Características sobresalientes.

Enterprise Architect es una herramienta de uso muy sencillo, que aborda el diseño y análisis UML y cubre el desarrollo de software desde la captura de requerimientos a lo largo de las etapas de análisis, diseño, pruebas y mantenimiento. EA es una herramienta multi-usuario, Windows, diseñada para ayudar a construir software robusto y fácil de mantener. Además, permite generar documentación e informes flexibles y de alta calidad.

Velocidad, estabilidad y rendimiento.

El Lenguaje Unificado de Modelado ofrece beneficios significativos para ayudar a construir modelos software rigurosos, donde es posible mantener la trazabilidad de manera consistente. Enterprise Architect lo realiza de un modo fácil, rápido y flexible.

Trazabilidad de extremo a extremo.

Enterprise Architect proporciona trazabilidad completa desde el análisis de requerimientos y los artefactos de diseño, hasta la implementación y el despliegue. En combinación con la asignación de recursos y tareas que incorpora, los equipos de Gestión de Proyectos y Calidad están dotados con toda la información necesaria para ayudarles a controlar los proyectos y sus entregas.

Construido sobre la base de UML 2.1

Las bases de Enterprise Architect están sustentadas en la especificación de UML 2 - pero no se detiene ahí! Usa Perfiles UML para extender el dominio de modelado, mientras que la validación del modelo asegura la integridad del proyecto. Combina los procesos de negocio, información y flujos de trabajo en un modelo usando las extensiones gratuitas para BPMN y Eriksson-Penker.

EA ayuda a administrar la complejidad permitiendo analizar las dependencias, gestionar modelos muy grandes, control de versiones con proveedores CVS o SCC, Líneas Base por cada instante de tiempo, capacidad de comparar (diff) para seguir los cambios del modelo, interfaz intuitiva y de alto rendimiento con una vista del proyecto en forma de "explorador".

EA permite la generación de documentos y herramientas de informes con un editor de plantilla WYSIWYG, que puede alcanzar niveles de calidad extraordinarios. Genera informes detallados y complejos de EA

con la información que necesita en el formato que su compañía o cliente demanda.

EA soporta la generación e ingeniería inversa de código fuente para muchos lenguajes, incluyendo C++, C#, Java, Delphi, VB.Net, Visual Basic, ActionScript y PHP. También hay disponibles Add-ins gratuitos para CORBA y Python. Con un editor de código fuente con "marcador de sintaxis" incorporado, EA le permite navegar y explorar su modelo de código fuente en un mismo entorno completamente integrado. Para aquellos que trabajan con Eclipse o Visual Studio.Net, Sparx Systems también vende interfaces ligeros para estos IDE's, permitiéndo modelar en EA y saltar directamente al código fuente en su editor preferido. Las plantillas de generación de código le permiten personalizar el código fuente generado de acuerdo a las especificaciones de su compañía.

EA le ayuda a visualizar sus aplicaciones permitiendo la ingeniería inversa de un amplio rango de lenguajes de desarrollo de software y esquemas de repositorios de Base de Datos. Puede modelar y representar frameworks completos (como Struts, por ejemplo) desde el código fuente o archivos Java .jar - o incluso ensambladores binarios .Net! Importando los frameworks y librerías de código, puede maximizar la re-utilización y compresión de sus activos tecnológicos.

EA soporta transformaciones de Arquitectura dirigida por modelos (MDA) usando plantillas con transformaciones fáciles de editar y desarrollar. Con las transformaciones incorporadas para DDL, C#, Java, EJB y XSD, puede desarrollar rápidamente soluciones complejas desde los "modelos independientes de plataforma" (PIM) simples que son el objetivo en los "modelos específicos de plataforma" (PSM). Un PIM se puede usar para generar y sincronizar múltiples PSM's - proporcionando un aumento de productividad significativo.

2.2. RESEÑA ORGANIZACIONAL

La investigación se llevó a cabo en la Empresa de Microfinanzas "MICREDITPERU" ubicada en la ciudad de Trujillo – Perú.

Nombre Comercial:	MicreditPerú
Actividad Principal:	Servicio de Crédito a Microempresarios y personas naturales.
Ámbito de Influencia:	A nivel nacional Actualmente en Los departamentos de La Libertad y Cajamarca.
Inicio de Operaciones:	29 de Agosto del 2007.
Estructura Organizacional	Ver Figura Nº 14

VALORES:

La Responsabilidad Social debe implicar las Expectativas de todos los grupos de interés (Accionistas/Inversionistas, Colaboradores y sus familias, Comunidad, Clientes, Proveedores, Medio Ambiente y Gobierno) alrededor de la empresa, para lograr el desarrollo sostenible esto es nuestra principal búsqueda y preocupación.

Creatividad: Característica del emprendedor y necesidad del empresario. La velocidad con que evoluciona la sociedad, la tecnología y el mundo de los negocios es vertiginosa. Lo cual nos empuja a innovar diariamente y crear nuevas formas de progreso y desarrollo y trasmitirlos a nuestros clientes.

Excelencia:

Es el conjunto de prácticas sobresalientes en la gestión de una organización y el logro de resultados basados en conceptos fundamentales que incluyen: la orientación hacia los resultados, orientación al cliente, liderazgo y perseverancia, procesos y hechos, implicación de las personas,

mejora continua e innovación, alianzas mutuamente beneficiosas y responsabilidad social.

Servicio a los Clientes:

El Servicio al cliente es un método eficaz para distinguirse de la competencia. De hecho, el servicio al cliente es una de las fortalezas de MicreditPerú, la amabilidad y el buen trato nos caracteriza.

VISION.

Brindar servicios financieros de Excelencia para el logro de los sueños de los pequeños emprendedores y personas naturales de las zonas urbano marginales y rurales de la zona norte del país.

MISION.

Financiamos los sueños de las personas naturales y microempresarios de las zonas urbano marginales y rurales de la zona norte del país, que desean desarrollar negocios sostenibles y adquirir bienes muebles e inmuebles para mejorar las condiciones de vida de la familia, brindándoles servicios financieros de calidad que satisfagan sus expectativas presentes y futuras.

OBJETIVOS.

OBJETIVO1: Brindar las herramientas informáticas necesarias para lograr el crecimiento sano de la institución

OBJETIVO 2: Fortalecer el conocimiento sobre las herramientas informáticas en todo el personal

OBJETIVO 3: Desarrollar un nuevo sistema de Software usando metodologías y herramientas de última generación que permitan maximizar la productividad y calidad en todo el ciclo de vida del Desarrollo del Software.

En Figuras Nº 15, y 16 se muestra la distribución de agencias en el departamento de La Libertad y Cajamarca respectivamente.

Figura Nº 14. ESTRUCTURA ORGANIZACIONAL

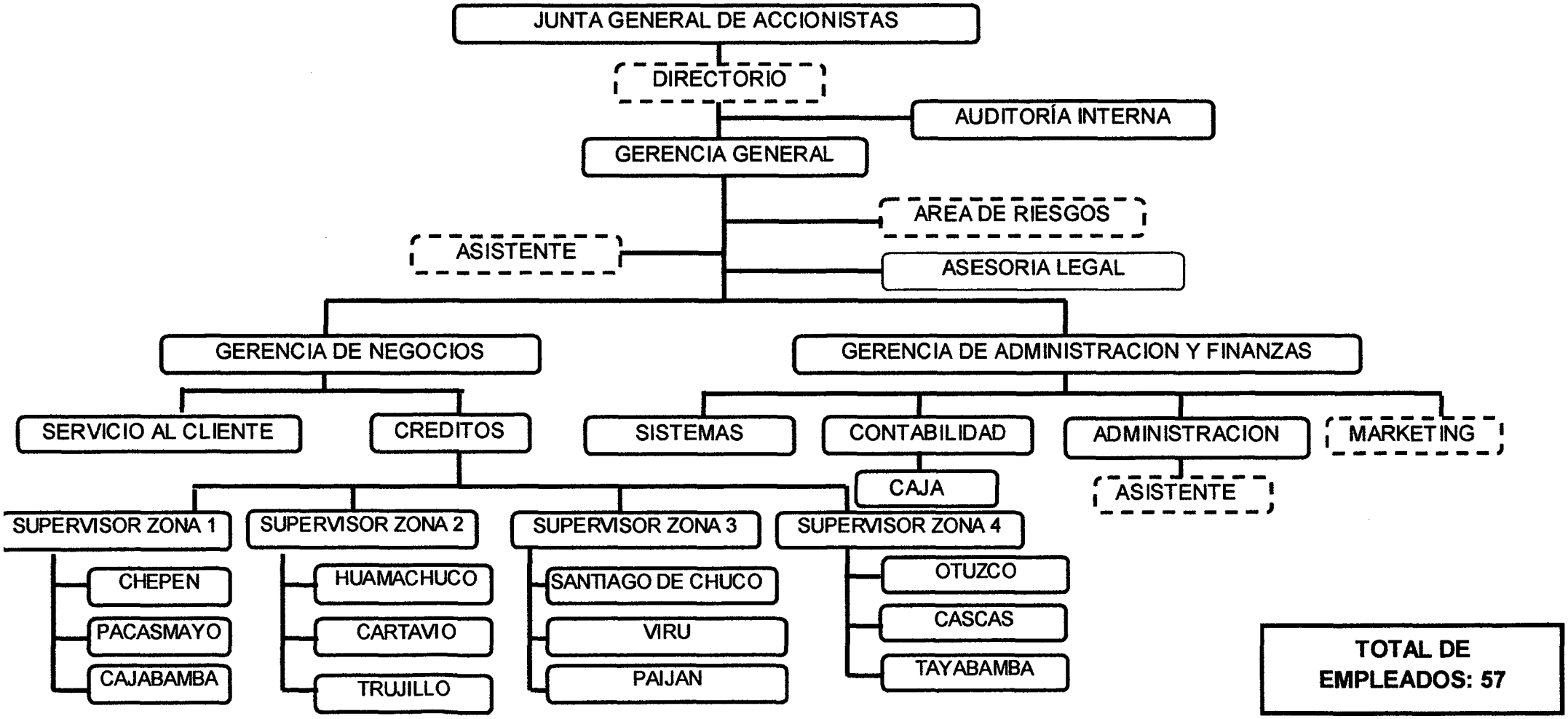


Figura Nº 15. Distribución de Agencias en departamento de La Libertad

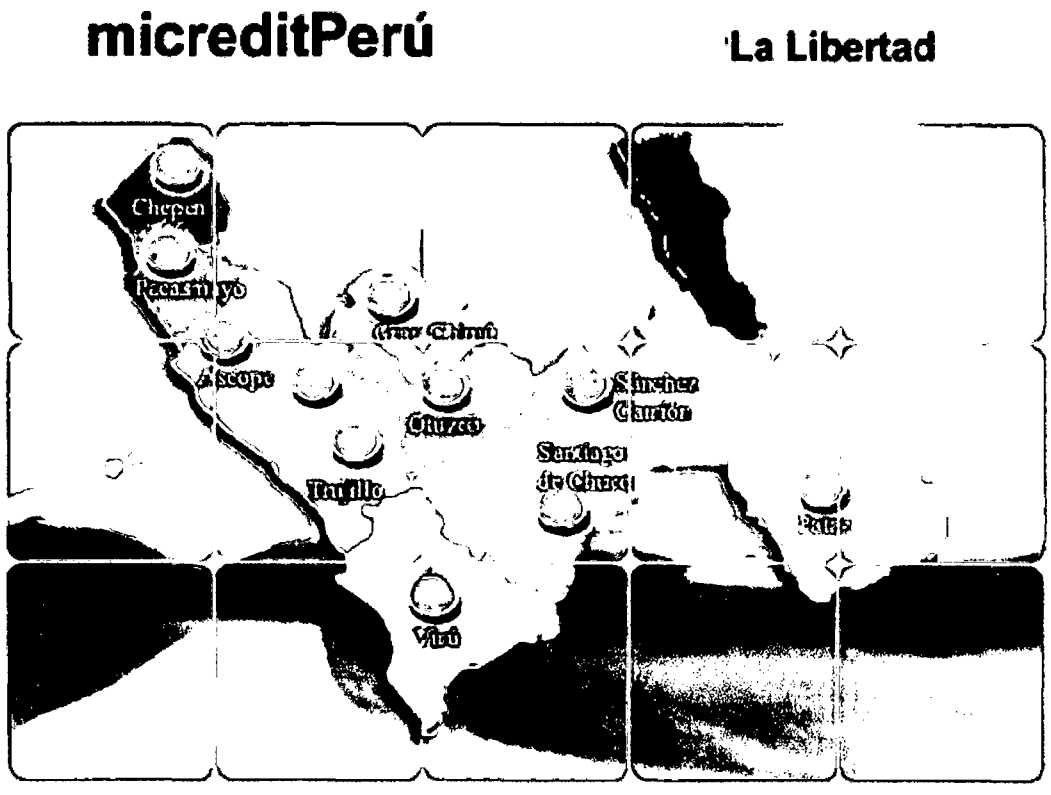
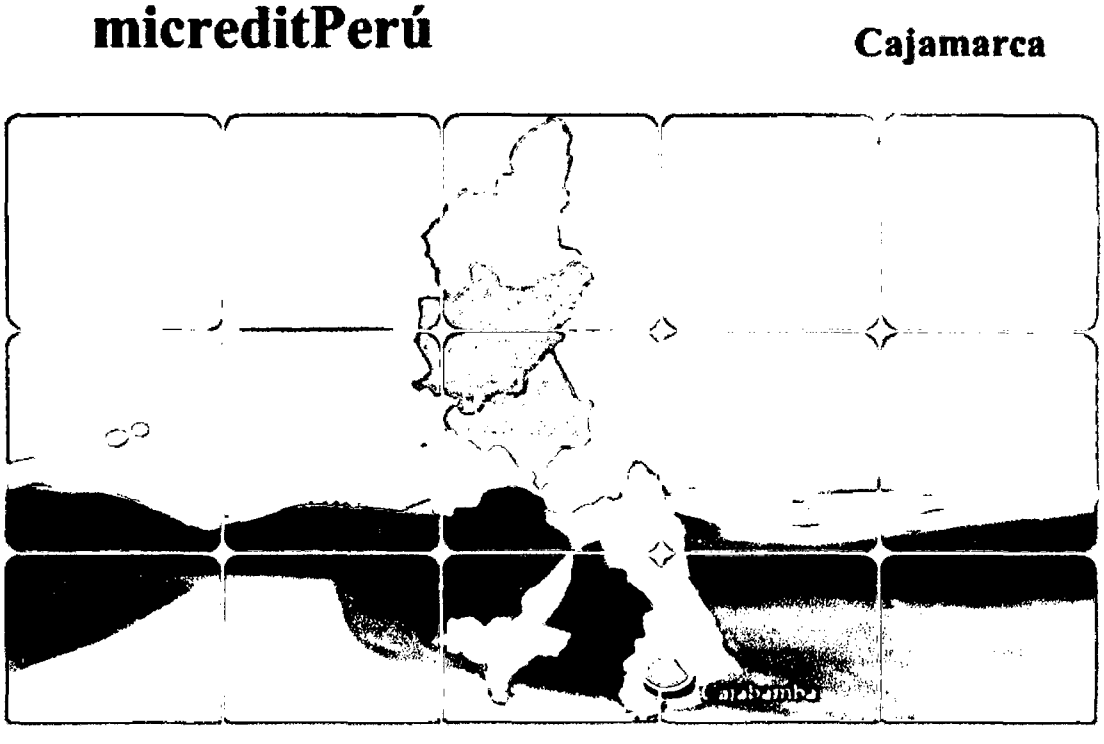


Figura Nº 16. Distribución de Agencias en departamento de Cajamarca



2.3. BASES TEORICAS

Con la finalidad de ubicar el enfoque que se adoptó en el análisis del objeto de estudio de esta tesis, en la perspectiva de lineamientos de carácter teórico; identificaremos un marco de referencia sustentado en el conocimiento existente; para ello, tomaremos en cuenta el conocimiento previamente construido, esto es, partiremos de la estructura teórica ya existente; para esto iniciaremos con algunas definiciones importantes:

a) **SOFTWARE:**

El Software es un conjunto de programas con la documentación necesaria para instalar, usar y mantener dichos programas. Es más lógico que físico. Es Desarrollado pero aun no fabricado y las fallas de diseño son corregidas más fácilmente que la de otros productos. Así mismo, el software no se deteriora ni se rompe y no tiene repuestos, y cuando falla solo hay que ajustarlo. [Carranza, 2008].

El software no son sólo programas para el computador, sino todos los documentos asociados y la configuración de datos que se necesitan para hacer que estos programas operen de manera correcta. Por lo general, un sistema de software consiste en diversos programas independientes, archivos de configuración que se utilizan para ejecutar estos programas, un sistema de documentación que describe la estructura del sistema, la documentación para el usuario que explica cómo utilizar el sistema y sitios Web que permitan a los usuarios descargar la información de productos recientes. [Sommerville, 2005].

b) **INGENIERIA DE SOFTWARE**:

La Ingeniería de Software es un enfoque sistemático del desarrollo, operación, mantenimiento y retiro del software. [Carranza, 2008].

Conjunto de métodos o procesos con el fin de organizar, controlar y estandarizar el desarrollo de software funcional y de calidad, surge a partir de los años 50's donde los sistemas que tienen un fin primordial de facilitar y agilizar la vida de los seres humanos estaba siendo creado de manera artesanal y sin ningún manejo controlado. Por ende los indefinidos problemas de la época y que se pueden seguir observando en algunos casos como:

- Sistemas con demasiadas funcionalidades que no cumplen las necesidades de las empresas.
- Software "intermitente" (a veces funciona a veces no)
- Programas inseguros, con problemas en diseño y manejabilidad nula para los usuarios.
- Sistemas incompletos costosos y de baja calidad.
- Retrasos y desviaciones en la planificación.
- Costos de mantenimiento elevados.
- Alta tasa de defectos.
- Requisitos mal comprendidos.
- Falsa riqueza de características.
- Cambios de personal.

La "Ingeniería del Software" define métodos que satisfacen las definiciones formales en el desarrollo de un producto e integra paradigmas de programación que dan el soporte a las metodologías ágiles para el desarrollo de software.

El objetivo principal de la ingeniería del software es el desarrollo eficaz de sistemas de computación [Pressman, 2005].

La ingeniería del software se encarga del "*Establecimiento y uso de principios de ingeniería robustos, orientados a obtener software económico que se afiable y funcione de manera eficiente sobre máquinas reales.*" [Pressman, 2005].

Además, en el contexto de la ingeniería de software se observa la referencia a modelos usados para el desarrollo de software. Según Whitten [Whitten y Bentley, 1998] un modelo es una representación de la realidad y de acuerdo a esto, los modelos son usados para definir elementos del minimundo en estudio, de los cuales dependerá el desarrollo correcto del software.

De acuerdo a Chikofsky [Chikofsky y Rubenstein, 1998] con la introducción de las microcomputadoras a mediados de los años 1980, se introdujeron al mercado de software herramientas y entornos automatizados, que permitieron hacer prácticos y económicos el uso de métodos formales en el desarrollo de Software.

Según Chikofsky, estas tecnologías son conocidas con el nombre de Ingeniería de Software Asistida por Computadora (del ingles, Computer Aided Software Engineering – CASE).

"Ingeniería de Software Asistida por Computadora – CASE, es un tipo de Ingeniería de Software en la que se intenta aumentar la eficacia de sus procesos, al soportar la realización de las tareas con el uso de tecnologías". [Perez, 1999].

De acuerdo a Pressman [Pressman, 2005], en el contexto de desarrollo de software se ha realizado mucho trabajo atendiendo las peticiones de automatización de diferentes tipos de contextos de desarrollo del software. Esto generó que no se pensara durante mucho tiempo en la posibilidad de atenderse a si mismos, por ejemplo creando soportes que automatizaran los trabajos de desarrollo de software.

Se concluye entonces que es necesario, cada vez más, con la finalidad de mejorar la productividad y calidad del software, proveer a los analistas y desarrolladores de herramientas que le permita automatizar su trabajo.

c) CALIDAD Y ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE [Pressman, 2005]:

La Calidad del Software es la concordancia con los requisitos especificados explicita e implícitamente. Pero cuando se considera de manera más general, la Calidad del Software abarca muchos productos diferentes, y factores de proceso y métricas relacionadas.

El Aseguramiento de la calidad del Software define y conduce las actividades requeridas para asegurar la Calidad del Software.

d) METODOLOGÍA DE DESARROLLO DE SOFTWARE:

Se entiende por Metodología de desarrollo de Software a una colección de documentación formal referente a los procesos, las políticas y los procedimientos que intervienen en el desarrollo del software.

La finalidad de una Metodología de desarrollo de software es garantizar la eficacia (p.ej. cumplir los requisitos iniciales) y la eficiencia (p.ej. minimizar las pérdidas de tiempo) en el proceso de generación de software.

Los riesgos a afrontar y los controles a establecer varían en función de las diferentes etapas del ciclo de vida de desarrollo. De forma general podríamos encontrar las siguientes fases:

- Definición del proceso de negocio y los requerimientos
- Documentación funcional
- Arquitectura y diseño técnico
- Codificación y ejecución de pruebas unitarias
- Pruebas globales del sistema
- Pruebas de integración
- Implantación
- Formación de usuarios
- Mantenimiento del sistema

Adicionalmente, durante todo el ciclo de vida del proyecto se deberán realizar tareas tales como:

- Gestión de la configuración: identificación de versiones, control de cambios, etc.
- Gestión de la calidad: seguimiento de errores, revisiones del nivel de calidad.
- Revisión de las premisas iniciales: revisión de los requerimientos y de los diseños.
- Gestión del entorno de desarrollo: herramientas de desarrollo, librerías, ficheros, gestión de datos (p.ej. para pruebas)

El núcleo de cualquier Metodología de desarrollo de Software se encuentra constituido por documentos escritos que detallan cada uno de los puntos expuestos.

La relación entre los diferentes procesos variará en función de la Metodología que la organización desee utilizar. En términos generales, podríamos hacer 3 agrupaciones:

- **Procesos en cascada**: El desarrollo se compone por fases secuenciales. Una vez cerrada una, no se considera la posibilidad de volver atrás.
- **Procesos en espiral**: Se desarrollan prototipos de software que son validados y mejorados a partir de la validación por parte del usuario. Una vez se tiene el prototipo definitivo, se realiza el desarrollo, implantación y mantenimiento de la misma forma que el modelo en cascada.
- **Procesos Iterativos**: Desarrollo incremental e iterativo, cada elemento desarrollado es validado con objeto de redefinir las necesidades funcionales. En este caso no se utilizan prototipos. Se dispone de marcos de trabajo de desarrollo ágil que son bastante populares como Extreme Programming (XP).

Si bien es importante formalizar una Metodología de desarrollo para minimizar riesgos e incrementar las posibilidades de éxito en los desarrollos, también es necesario establecer mecanismos de mejora continua sobre los procesos que componen la Metodología. En ese sentido disponemos del estándar **ISO 15504** que proporciona un marco de trabajo para la verificación de los procesos de software (planificación, adquisición, desarrollo, soporte, etc.). Es decir, nos ofrece unas pautas para verificar si estos procesos son efectivos en la consecución de sus objetivos.

Las Metodologías para Desarrollo de Software son métodos que indican cómo hacer más eficiente el desarrollo de Software. Para ello suelen estructurar en fases o ciclos la vida de dicho Software, con el fin de facilitar su planificación, desarrollo y mantenimiento.

Las Metodologías de desarrollo de software deben definir: objetivos, fases, tareas, productos y responsables, necesarios para la correcta realización del proceso y su seguimiento.

Los principales objetivos de una Metodología de desarrollo de Software son:

- Asegurar la uniformidad y calidad tanto del desarrollo como del sistema en sí.
- Satisfacer las necesidades de los usuarios del sistema.
- Conseguir un mayor nivel de rendimiento y eficiencia del personal asignado al desarrollo.
- Ajustarse a los plazos y costes previstos en la planificación.
- Generar de forma adecuada la documentación asociada a los sistemas.
- Facilitar el mantenimiento posterior de los sistemas.

e) METODOLOGIAS AGILES PARA EL DESARROLLO DE SOFTWARE

Las metodologías ágiles surgen dentro de la Ingeniería del Software fuera de la academia como un contexto del desarrollo creado y usado por pensadores y filósofos quienes establecen prácticas que toman elementos tradicionales y nuevos, los aplican en los procesos y las personas, de nada serviría una excelente documentación si el sistema no ofrece las características deseadas.

Un modelo de desarrollo ágil, generalmente es un proceso incremental, (pequeños y frecuentes entregables de software con ciclos rápidos), también cooperativo (Clientes y Desarrolladores trabajan constantemente con una comunicación muy fina y constante), sencillo (El método es fácil de aprender o modificar para el equipo, es bien documentado mediante libros o la Web) y principalmente adaptativo (capaz de permitir cambios de último momento).

Las características de las metodologías ágiles pueden explicarse a través de los siguientes cuatro principios fundamentales:

- **Los Individuos y las Interacciones son más importantes que los procesos y las herramientas:** Dado que el proceso de desarrollo es creativo, no es posible pensar que las personas funcionen respondiendo a órdenes, a procesos rígidos.
- **Que el software funcione es más importante que la documentación exhaustiva:** Puesto que si el software no funciona la documentación no vale de nada. A nivel interno puede haber documentación, pero solo la necesaria y a nivel externo lo que el cliente requiera.

- **La colaboración con el Cliente es más importante que la negociación de contratos:** Supone que la satisfacción del cliente con el producto será mayor, mientras exista una conversación y realimentación continúa entre este y la empresa.
- **La respuesta ante el cambio es más importante que el seguimiento de un plan:** Puesto que si un proyecto de software no es capaz de adaptarse a los cambios fracasará, especialmente en productos de gran envergadura, La estrategia de planificación se basa en: Planes detallados para las próximas semanas, planes aproximados para los próximos meses y muy generales para plazos mayores.

Las metodologías ágiles son otra opción para el desarrollo de software, muy aplicadas y además presenta adeptos y gurús en contra, algunos expertos mencionan que las metodologías ágiles son una moda y quedarán ahí, sin embargo existen empresas que desde hace tiempo utilizan y han evolucionado gracias a dichos métodos.

Existen gran variedad de Metodologías o también para algunos autores enfoques que tienen gran utilidad en la elaboración de un sistema de información, entre las más comunes se encuentran:

f) HERRAMIENTA CASE GENEXUS

GENEXUS es un software, basado en conocimiento, que le permite enfocarse en su negocio para desarrollar en forma incremental aplicaciones de misión crítica basadas en su propio y exclusivo know how [Márquez, 2006].

Con GENEXUS es posible embarcarse en todo tipo de proyecto, gracias a su flexibilidad e integración permite desarrollar aplicaciones en las tecnologías de vanguardia sin necesidad de conocerlas. Esto garantiza, a su vez, un seguro ante el cambio tecnológico, porque con GENEXUS la fortaleza del desarrollador es el conocimiento que tenga del negocio frente a los lenguajes o plataformas que use el mercado.

Características:

Genera el 100% de la aplicación y de la Base de Datos así como también de

mantener en forma automática el Modelo de Datos, la información y las aplicaciones.

Es una herramienta con un entorno de desarrollo orientado a intenciones y necesidades del desarrollador, que hacen intuitivo su uso y facilitan su aprendizaje.

Automatiza el desarrollo y mantenimiento de las aplicaciones, liberando a los desarrolladores de tareas rutinarias y tediosas.

Permite integrar diferentes tecnologías a fin de lograr herramientas más potentes y reciclar trabajos anteriores o de terceras partes.

Tecnologías soportadas [www.Genexus.com].

Plataformas

Plataformas de Ejecución

JAVA, Microsoft.NET, Pocket PC.

Sistemas Operativos

IBM AS/400, LINUX, Windows NT/2000/2003 Servers, Windows NT/2000/XP, y Windows Mobile.

Internet

JAVA, ASP.NET, Visual Basic (ASP), HTML, WebServices

Bases de Datos (DBMS)

IBM DB2, Informix, Microsoft SQL Server, MySQL, Oracle, PostgreSQL

Lenguajes de Programación.

JAVA, C#, COBOL, RPG, Visual Basic, Visual FoxPro

Servidores WEB

Microsoft IIS, Apache, WebSphere, etc.

Múltiples Arquitecturas

Arquitecturas de múltiples capas, basadas en Web, Cliente/Servidor, y centralizadas (iSeries)

Herramientas de Business Intelligence y Workflow

Soluciones para Reporting, Data Warehousing, y Workflow para todos los servidores soportados.

GENEXUS es una herramienta que genera y mantiene el 100% de sus aplicaciones de negocios, permitiéndole gestionar eficientemente todo el ciclo de vida de sus sistemas.

Diseño basado en el conocimiento

• Diseño de aplicaciones a partir de las visiones de los usuarios

• Diseño del Modelo de Datos totalmente automático

• Diseño y prototipo independiente de la plataforma

Desarrollo automático

• Generación 100% automática de bases de datos y aplicaciones

• Generación y mantenimiento automático de la documentación de aplicaciones

Mantenimiento automático

• Mantenimiento 100% automático de la Base de Datos y las aplicaciones

• 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

En esta tesis, tomando como base los conceptos antes mencionados, se definió una Metodología Ágil para el Desarrollo de Software, con el apoyo de la herramienta CASE GENEXUS.

CAPITULO III

METODOLOGIA DE LA INVESTIGACION

3.1. INVESTIGACION EN INGENIERIA DE SOFTWARE.

La metodología de investigación empleada en este trabajo de tesis de maestría es resultante de la combinación de un método propuesto por [Marcos y Marcos, 1998], [Marcos, 2005], y el método denominado investigación en acción [Advison et,al., 1999].

Según [Marcos y Marcos, 1998], [Marcos, 2005], la investigación en ingeniería difiere sustancialmente, tanto en *objeto de estudio* como en *método*, de la investigación en las tradicionalmente llamadas ***Ciencias***. Mientras las ***Ciencias*** se ocupan del estudio de objetos y fenómenos existentes (física, metafísica o conceptualmente), las Ingenierías basan sus estudios en *cómo* hacer; *cómo* crear nuevos objetos. Es por esto que los métodos de investigación Científicos no son siempre directamente aplicables a problemas de investigación de carácter ingenieril, incluyendo a la Investigación en Ingeniería de Software.

3.1.1. OBJETO DE ESTUDIO DE LA INGENIERIA DE SOFTWARE [Marcos, 2005], [Marcos y Marcos, 1998].

¿Cuál es el objeto de estudio de la Ingeniería de Software?

Al dividir los problemas de la Ingeniería de Software según la naturaleza del conocimiento, una primera aproximación es:

A) La investigación enfocada a la *construcción de nuevos objetos* (procesos, modelos, metodologías, técnicas, etc.). Su objeto de estudio es la construcción de nuevas herramientas (métodos, modelos, etc.).

B) Investigación enfocada al *estudio de dichos objetos* (métricas, optimización, etc.). Su objeto de estudio no difiere de las ciencias tradicionales sino en que los objetos estudiados son artificiales en lugar de naturales.

C) Investigación enfocada a la *implantación y uso* de estos nuevos objetos.

3.1.2. METODO DE INVESTIGACION PARA LA INGENIERIA DE SOFTWARE [Marcos, 2005], [Marcos y Marcos, 1998].

Método de investigación para casos cuyo objeto de estudio tipo A): No es posible aplicar métodos empíricos, ya que el objeto de estudio aún no existe. Tienen un importante componente de **creatividad,** así como un fuerte componente social y cultural en cuanto a la adopción de nuevos paradigmas, la parte de trabajo en equipo de los procesos de software, el trabajo colaborativo, etc. Por todo ello, y cada vez más se están empleando métodos de carácter **cualitativo** en la Investigación en Ingeniería de Software.

Un método para, por ejemplo, la definición de un modelo nuevo, consistirá fundamentalmente en estudiar los modelos existentes, reflexionar acerca de ellos determinando sus ventajas y limitaciones, y plantear un nuevo modelo que, manteniendo las ventajas de los modelos estudiados, supere, en la medida de lo posible, las limitaciones de los mismos. El llegar a una mejor propuesta final dependerá, en gran medida, de la creatividad y sentido común aplicados a la construcción del nuevo modelo.

El método de investigación hipotético – deductivo

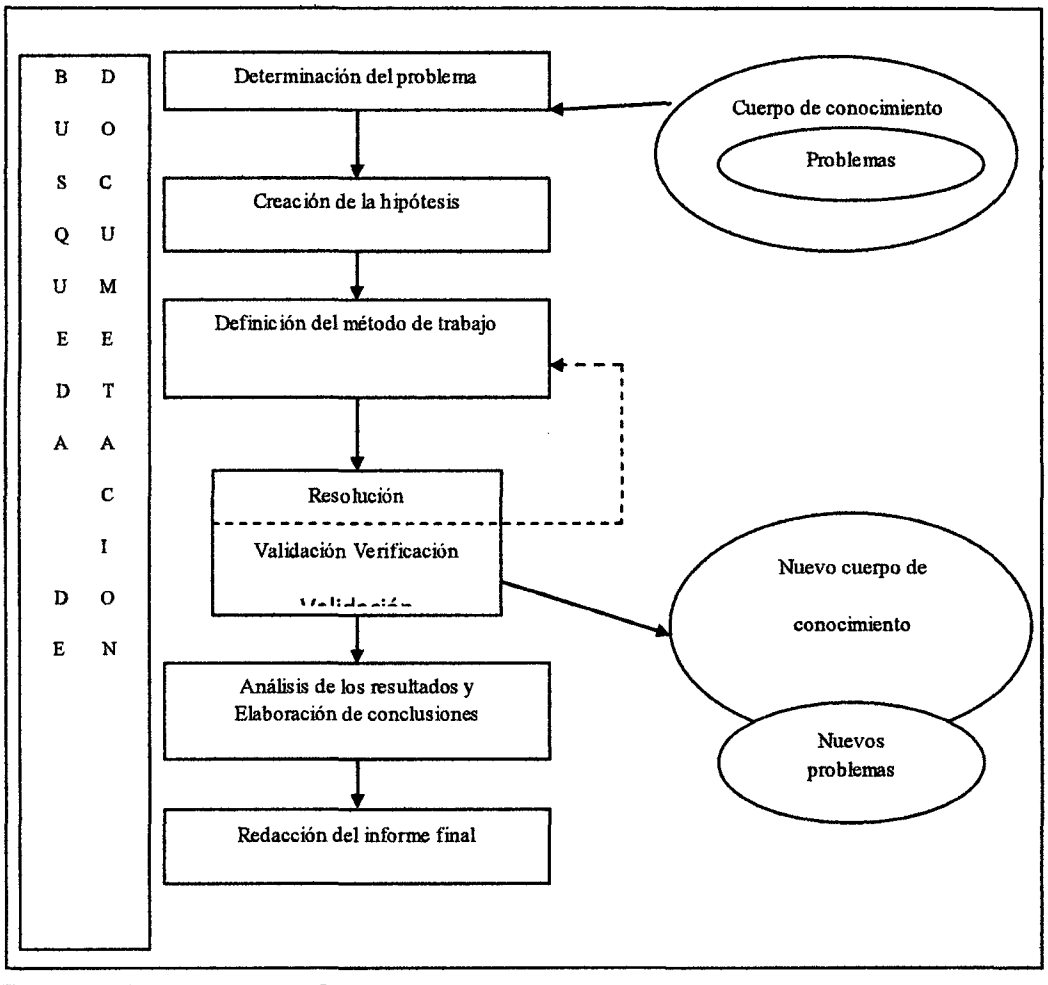
En [Marcos, 2005] se propone un método para la investigación en Ingeniería de Software, basado en el método hipotético - deductivo [Bunge, 1976], consta de varias etapas que, por su generalidad, son aplicables, con ciertas modificaciones, a cualquier tipo de investigación (ver figura 17). Esta etapas son: *Búsqueda de documentación, determinación del problemas, la creación de la hipótesis, la definición del método de trabajo, la resolución, validación y verificación, el análisis de los resultados y elaboración de conclusiones y la redacción de un informe final.*

Etapa: Creación de la hipótesis:

En los métodos tradicionales de investigación científica la hipótesis se formula en términos causales (si ocurre A entonces ocurre B). Estas hipótesis son conjeturas de hechos que el método científico deberá contrastar y verificar. Sin embargo, es fácil comprobar que la hipótesis en una investigación en Ingeniería de Software, en la especificación de una nueva metodología de desarrollo de software, donde el objeto de estudio es la construcción de nuevos

objetos (métodos, modelos, técnicas), no responde a un planteamiento de causa efecto. Que, por no existir, el objeto de estudio, no es susceptible de experimentación. No es este el caso de la investigación realizada para objetos de estudio existentes. Una hipótesis válida para este caso sería: *Si aumenta el número de claves foráneas en un esquema relacional disminuye la mantenibilidad del mismo.* Hipótesis en términos causales que podrá ser verificada por medio de métodos cuantitativos (normalmente experimental).

Figura Nº 17: Método de Investigación basado en método hipotético – deductivo de [Bunge, 1976]



Fuente: [Marcos, 2005]

Para el caso de Investigación de Ingeniería de Software, donde el objeto de estudio es la construcción de nuevos objetos (métodos, modelos, técnicas), la hipótesis se formulará como la descripción del nuevo objeto que se desea

construir, que en el caso que se desee construir una nueva metodología de desarrollo de software, será la descripción de la nueva metodología de desarrollo, a que sistemas se quiere aplicar, que etapas del ciclo de vida se abordarán, en que tecnología se basará.

Etapa: Definición del método de trabajo:

No existe un método universal aplicable a cualquier investigación.

Al igual que ocurre con el "método" de investigación, no podemos hablar del "proceso" o de la "metodología" de desarrollo de software. Una metodología de desarrollo de software debe tomarse como una guía, pero no como algo rígido; debe adaptarse para cada utilización de la misma, del mismo modo que el método de investigación. Al iniciar una investigación es preciso elegir el paradigma metodológico que se va a seguir (cualitativo, cuantitativo, etc.), así como el método concreto (investigación en acción, experimentación, etc).

La realimentación entre la fase de *resolución* y *verificación*, y la de *definición del método de trabajo* (señalados con trazo discontinuo en la figura Nº 17), muestra como el método se va haciendo, refinando, a medida que se avanza en la resolución del problema. Así, podemos decir que la especificación del método de trabajo no concluye hasta que finaliza la etapa de resolución y verificación. La etapa de resolución mostrará como resolver el problema y, al igual como ocurre con la etapa de definición el método de desarrollo de software en ella definido, se irá haciendo y refinando en aproximaciones sucesivas.

Etapa: Resolución, validación y verificación.

Para el caso de la definición de un nuevo método de desarrollo de software, es necesario realizar, de un modo simplificado, las siguientes actividades: especificación del proceso de desarrollo de software, especificación de actividades a realizar en el mismo, especificación de las técnicas a utilizar. Aunque para ello se podrían emplear diferentes métodos de investigación, proponemos para cada una de estas actividades las siguientes:

a) **Resolución:** Mediante el análisis de casos de estudio y un proceso de imaginación y creatividad.

b) **Verificación:** Mediante la implementación de un prototipo que permita eliminar ambigüedades y verificar su corrección (podría también aquí utilizarse alguna otra técnica de especificación formal no necesariamente compilable).

c) **Validación:** Mediante su aplicación en casos de prueba.

Otro método recomendable para la *resolución*, *validación* y *verificación* es la *Investigación en Acción.* Este método nos permitirá definir la metodología al tiempo que se va usando y refinando en casos reales. Es especialmente útil para la especificación, validación y verificación de procesos. El problema de este método estriba en la dificultad de trabajar con empresas y de que éstas nos permitan efectivamente utilizar sus desarrollos como casos de prueba. El método basado en casos de prueba se asemejaría a una Investigación en Acción de laboratorio, cuando el caso de estudio y el de prueba coinciden.

Un método de desarrollo de software nos da las pautas para la construcción de nuevos objetos (software), al igual que el método de investigación nos da las pautas para la construcción de nuevos objetos (metodologías, modelos, etc.).

Los métodos tradicionales de investigación científica y, en particular el propuesto por [Bunge, 1976], solo señalan la necesidad de verificar la hipótesis planteada (o la teoría propuesta, si se trata de ciencias formales). Sin embargo, en el caso de la creación de una metodología de desarrollo nueva, la etapa de verificación tiene dos tareas: La *validación*, comprobar que el modelo (la metodología) se ha construido según la hipótesis planteada, y la *verificación*, comprobar que se ha construido correctamente, esto es, que es consistente.

Etapa: Análisis de resultados elaboración de conclusiones:

Se trata de contrastar la hipótesis planteada al comienzo de la investigación con los resultados obtenidos de esta. Se debe comprobar hasta que punto se han cumplido los objetivos y en qué medida se ha resuelto el problema. En esta etapa es muy importante delimitar los aspectos que no se han podido resolver y otros nuevos problemas que

hayan surgido como consecuencia de la investigación y que pasarán a ser puntos de partida de nuevas investigaciones.

En métodos de naturaleza cuantitativa, parece que este aspecto es claro. Cuando la investigación es de naturaleza cualitativa y la hipótesis formulada no responde a una expresión de causa efecto, la contrastación de la hipótesis consiste básicamente en comprobar hasta que punto se han cumplido los requisitos impuestos al principio de la investigación: ¿Cubre la metodología desarrollada todas las fases del proceso requeridas?, ¿Es posible su utilización en aquellos entornos para los que fue pensada?, etc.

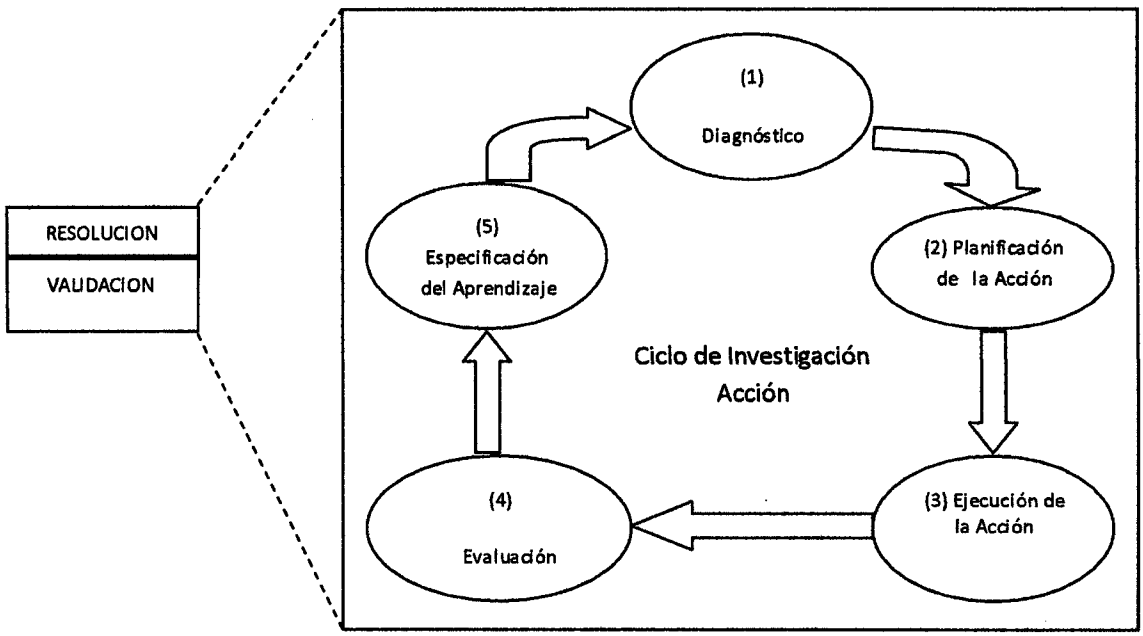
3.2. INVESTIGACION EN ACCION

La Investigación en Acción [Avison et al., 1999] es un método de investigación cualitativo utilizado para la validación de los trabajos de investigación mediante su aplicación en proyectos reales, reforzando la interacción entre los investigadores y los participantes de dichos proyectos reales. Este método, por su carácter de validación práctica, es especialmente apropiado para la investigación en ingeniería y, especialmente, en Ingeniería de Software.

Una de las características de este método es que la investigación se realiza a la vez que se aplican sus resultados, de modo que esta aplicación permite validar y refinar los resultados de la investigación en un proceso iterativo. El proceso de investigación definido en el método de Investigación en Acción no es por tanto un proceso lineal, sino que va avanzando mediante la realización de ciclos, llamados Ciclos de Investigación en Acción, en cada uno de los cuales se ponen en marcha nuevas ideas, que son puestas en práctica y comprobadas hasta el siguiente ciclo [Wadswroth, 1998].

La Figura Nº 18 representa las etapas definidas en [Susman y Evered, 1978] para cada ciclo de Investigación en Acción. Como puede verse en la figura, en este caso, los ciclos de investigación se realizan durante la etapa de resolución y validación de la Figura Nº 17.

Figura Nº 18. Etapa de Resolución y Validación



Fuente. [Susman y Evered, 1978]

A continuación se describen las etapas de cada ciclo de Investigación en Acción.

Diagnóstico: Es la etapa inicial de cada iteración, involucra la identificación de los aspectos a mejorar y los problemas a resolver propios de cada iteración. Los problemas que se identifican y describen en esta etapa guardan relación con los problemas identificados durante la etapa de *determinación del problema* ilustrada en la Figura Nº 17. Sin embargo, es importante tener en cuenta que según van sucediéndose las iteraciones sobre los diferentes casos de estudio, pueden surgir nuevos problemas y oportunidades de mejora de los resultados de las iteraciones anteriores.

Planificación de la acción: En esta etapa se consideran los diferentes cursos de acción a tomar para resolver los problemas detectados en la etapa anterior, a partir de los casos de estudios identificados preliminarmente durante la etapa de definición del método de trabajo (ver Figura Nº 17). El resultado de esta actividad es la identificación de una serie de actividades a ejecutar sobre un determinado caso de estudio.

Ejecución de la Acción: Implica la implementación del curso de acción elegido en la etapa anterior sobre el caso de estudio seleccionado.

Evaluación: Después de completar la acción, se examinan los resultados. La evaluación incluye determinar si se ha alcanzado los efectos esperados de la acción, y si los efectos mitigaron los problemas identificados previamente. Si los resultados obtenidos fueran no satisfactorios, los problemas identificados se trasladan a las siguientes iteraciones del ciclo de Investigación en Acción.

Especificación del aprendizaje: Si bien esta actividad figura al final del ciclo, se trata de una actividad que se lleva a cabo durante todo el ciclo. En ella, los conocimientos adquiridos durante el ciclo de investigación en acción, por más que hayan sido no satisfactorios, se comparten con las personas involucradas en el caso de estudio, para que dicho conocimiento pueda ser asimilado en las tareas que realizan y para que se pueda planificar nuevos ciclos de investigación en acción.

CAPITULO IV

METODOLOGIA DE DESARROLLO PROPUESTA

INTRODUCCION

Este capítulo está conformado por dos secciones, en la sección 4.1 se describirá por qué y cómo ha sido derivada o creada la Metodología propuesta, objetivos, la herramienta CASE Genexus, sus características, funcionalidades avanzadas. En la sección 4.2., se presenta de manera formal la Metodología propiamente dicha, indicando sus componentes: Ciclos de vida, fases, roles, principios y buenas prácticas, etc.

4.1. BASES DE LA PROPUESTA.

Una vez estudiadas las principales metodologías ágiles y tradicionales, se han obtenido una serie de ventajas, limitaciones o desventajas y las buenas prácticas de cada una de ellas. La Metodología propuesta, eliminará o minimizará las limitaciones antes mencionadas y seleccionará y capitalizará, las buenas prácticas de dichas metodologías.

El método que se ha seguido para derivar, crear o definir la Metodología propuesta, consistió fundamentalmente, en estudiar las principales metodologías ágiles y tradicionales, reflexionar acerca de ellas determinando sus ventajas, limitaciones, buenas prácticas y apoyándonos además en las características y ventajas de la herramienta CASE Genexus, se plantea una nueva metodología que manteniendo las ventajas de las metodologías estudiadas, supere, en la medida de lo posible, las limitaciones de las mismas.

El llegar a una mejor propuesta final dependió, en gran medida de la creatividad y sentido común aplicados a la construcción o definición de la nueva Metodología. El método empleado para la resolución, validación y verificación de la Metodología propuesta se basó en casos de estudio y casos de prueba coincidentes, el cual se asemeja a una investigación en acción de laboratorio, este método nos permitió definir la metodología al tiempo que se va usando y refinando en casos reales. Para esto fue necesario realizar de un modo simplificado las actividades siguientes: especificación del procesos de desarrollo de software, especificación de actividades a realizar en el mismo, especificación de herramientas a utilizar, en este caso la herramienta CASE Genexus.

4.1.1. OBJETIVOS DE LA METODOLOGIA DE DESARROLLO PROPUESTA

Como principales objetivos consideramos lo propuesto por el profesor Sachidanandam Sakthivel.

http://www.um.es/docencia/barzana/IAGP/Iagp3.html

1. La Metodología debe permitir desarrollar sistemas de software con la calidad requerida y a satisfacción del usuario.
2. La Metodología debe permitir desarrollar gran variedad de sistemas de software.
3. La Metodología debe soportar todas las etapas del desarrollo de software.

Así también, adicionalmente tiene los objetivos siguientes:

4. La Metodología debe permitir superar o mitigar las limitaciones o desventajas de las metodologías de desarrollo de software tradicionales y ágiles, en lo referente a facilitar el desarrollo y mantenimiento de los sistemas de software.
5. Ofrecer a la comunidad informática una Metodología Ágil de desarrollo de Software que se apoye en herramientas de última generación, para llevar a cabo con eficacia y eficiencia las diversas etapas que conllevan el Desarrollo de Software de calidad.

En la siguiente sección se presentan los principios en los que se apoya la Metodología propuesta, describiendo los principios en los que se basa y las relaciones entre tales conceptos.

4.1.2. HERRAMIENTAS Y TECNICAS USADAS EN LA METODOLOGIA PROPUESTA.

4.1.2.1. LA HERRAMIENTA CASE GENEXUS.

INTRODUCCION

Según [Artech Consultores S.R.L., 2008]. Genexus es una herramienta desarrollada por la Empresa Artech, usando Inteligencia Artificial y empleando el Lenguaje de Programación Prolog y C++, cuyo objetivo es asistir al analista y a los usuarios en todo el ciclo de vida de las aplicaciones.

Según [Marquez L., 2006] es una herramienta de especificación de sistemas de software basada en la aplicación de un modelo matemático (basado en los trabajos de Codd [Cood,E.F., 1970] y Warnier-Orr [Orr, K.T., 1982]) que permite capturar e integrar las visiones de los usuarios en bases de conocimiento (KB – Knowledge Base) mediante objetos Genexus y a partir de los cuales genera, mediante ingeniería inversa y procesos de inferencia, bases de datos normalizadas y aplicaciones completas [Arias, F.J., Moreno, J., AND Ovalle, 2007]. Todo esto es generado para diferentes plataformas destino a partir de una misma especificación básica, pudiendo mantenerlas en forma automatizada ante cambios en los requerimientos.

El desarrollador Genexus cuenta con distintos artefactos para crear una aplicación, los cuales se denominan objetos Genexus, siendo los principales los siguientes:

Los objetos Transacciones son utilizados para definir la estructura de los datos, su forma básica de ingreso (interfaz gráfica) y las reglas de negocio que se desean aplicar.

Los objetos Theme, WorkPanel, WebPanel, WebComponent y MasterPage se utilizan para definir la interfaz gráfica.

Los objetos Procedure y DataProvider contienen lógica pura, el primero de forma procedural y el segundo en forma declarativa.

Los objetos estructurados (SDT - Structured Data Type) representan, en la plataforma Genexus, estructuras complejas.

Los objetos externos 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.

4.1.2.1.1. MODELANDO LA REALIDAD

El modelado de la realidad en el contexto Genexus se logra creando diferentes objetos que se almacenan en una Base de Conocimiento.

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; 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.

Está basado en la especificación del modelado de datos conceptual y utiliza el supuesto de nombres únicos (URA - Universal Relation Assumption), es decir, un mismo concepto se llama igual en todos los lugares donde se utiliza.

El diseño y prototipo que genera son realizados y probados en un ambiente Windows, Windows NT/2000/XP. Cuando el prototipo está totalmente aprobado por sus usuarios, la Base de Datos y los programas de aplicación son generados y/o mantenidos en forma totalmente automática, para el ambiente de producción.

La idea básica de Genexus es automatizar todo aquello que es automatizable: normalización de los datos y diseño, generación y mantenimiento de la Base de Datos y de los programas de aplicación. De esta manera se evita que el analista deba dedicarse a tareas rutinarias y tediosas, permitiéndole poner toda su atención en aquello que nunca un programa de cómputo podrá hacer: entender los problemas del usuario.

Como un subproducto, Genexus ofrece una documentación permanentemente actualizada.

4.1.2.1.2. EL PROBLEMA TEORICO: Metodologías tradicionales de Desarrollo de Software y problemas Asociados.

La forma tradicional de desarrollar aplicaciones parte de una premisa básica: *es posible construir un Modelo de Datos estable de la empresa.* Basándose en esa premisa, la primera tarea que se encara es el análisis de datos, donde se estudia la realidad en forma abstracta y se obtiene como producto el *Modelo de Datos de la empresa*. La segunda tarea es *diseñar la Base de Datos.*

Una vez que se ha estudiado la realidad desde el punto de vista de los datos, se hace lo propio desde el punto de vista de las funciones (análisis funcional). Sería deseable que el estudio de la realidad tuviera como producto una especificación funcional que dependiera sólo de dicha realidad. Lo que se hace en las metodologías más usadas, sin embargo, es obtener una especificación funcional que se refiere a los archivos de la Base de Datos (o bien a las entidades del Modelo de Datos, lo que es esencialmente equivalente).

Una vez que se tiene la Base de Datos y la especificación funcional, se pasa a la *implementación de las funciones*, existiendo tradicionalmente para ello varias opciones (lenguajes de 3ª. o 4ª generación, generadores, interpretadores).

Sin embargo, todas las formas de implementación vistas tienen un problema común: parten de la enunciada premisa: *es posible construir un Modelo de Datos estable de la empresa*, y esta premisa es falsa.

Realmente es imposible hacer, de una forma abstracta, un Modelo de Datos detallado de la empresa y con el suficiente nivel de detalle y objetividad, porque nadie la conoce como un todo.

Por ello es necesario recurrir a múltiples interlocutores, y cada uno de ellos proyecta sobre el modelo, su propia subjetividad. Una consecuencia de esto es que, durante todo el ciclo de vida de la aplicación, se producen cambios en el modelo.

Pero aún si se considerara la situación ideal, donde se conocen exactamente las necesidades y, entonces, es posible definir la Base de Datos

óptima, el modelo no podrá permanecer estático porque deberá acompañar la evolución de la empresa.

Todo esto sería poco importante, si la especificación funcional y la Base de Datos fueran independientes. Sin embargo, dado que la especificación funcional se refiere a la Base de Datos, las inevitables modificaciones en ésta implican la necesidad de modificaciones (manuales) en aquella.

La mayor consecuencia de lo anterior está constituida por los altos costos de mantenimiento: en la mayoría de las empresas que trabajan de una manera convencional se admite que el 80% de los recursos que teóricamente están destinados al desarrollo de software, realmente se utilizan para hacer mantenimiento de las aplicaciones ya implementadas.

Cuando se trata de aplicaciones grandes la situación es aún peor: este mantenimiento comienza mucho antes de la implementación, lo que hace que los costos de desarrollo crezcan en forma hiperlineal con respecto al tamaño del proyecto.

Dado que es muy difícil, en este contexto, determinar y propagar las consecuencias de los cambios de la Base de Datos sobre los procesos, es habitual que, en vez de efectuarse los cambios necesarios, se opte por introducir nuevos archivos redundantes, con la consiguiente degradación de la calidad de los sistemas y el incremento de los costos de mantenimiento.

4.1.2.1.3. DESARROLLO INCREMENTAL.

En los últimos años se ha hablado mucho en la industria de la Base Conocimiento y, dentro de este rótulo se han colocado muchas cosas que están bien distantes del Desarrollo Basado en Conocimiento a que nos queremos referir aquí.

Generalmente la industria se ha referido a maneras de organizar y/o acceder el conocimiento para ser utilizado de una forma tradicional por los seres humanos. Se trata de una versión actualizada, utilizando la tecnología actualmente disponible, de los libros (y que es de enorme utilidad para toda la humanidad): accedemos a un cierto conocimiento leyendo un libro y, en nuestra mente, hacemos razonamientos sobre ese conocimiento lo que, eventualmente, determina acciones. Los buscadores de texto inteligentes que

están disponibles desde hace pocos años hacen que este conocimiento sea cada vez más accesible a los seres humanos.

Como característica general, este conocimiento no es "entendible" por una máquina y, en consecuencia, no es operable. Adicionalmente, como el razonamiento de los seres humanos puede lidiar razonablemente (dentro de ciertos límites) con la ambigüedad y, aún, con la inconsistencia, este conocimiento muchas veces no es riguroso.

Entonces, es bueno restringir el concepto de conocimiento que utilizaremos en nuestro Desarrollo Basado en Conocimiento. Se trata de Conocimiento que cumple las siguientes condiciones:

- Riguroso
- Representable en forma objetiva
- Operable

Una nueva manera de resolver el problema del desarrollo de sistemas de software pasa por la sustitución de la premisa básica enunciada: asumir que no es posible construir un Modelo de Datos estable de la empresa y, en cambio, utilizar una filosofía incremental y hacer un Desarrollo Basado en Conocimiento. Un esquema incremental parece muy natural: no se encaran grandes problemas, sino que se van resolviendo los pequeños problemas a medida que se presentan.

¿Cuál será la repercusión de este tipo de esquema sobre los costos de mantenimiento?

Si se utilizaran, con este enfoque, las metodologías anteriormente reseñadas, esa repercusión sería muy grande: el Modelo de Datos se modificaría constantemente y los costos de mantenimiento serían aún mucho mayores que los enunciados.

Puede verse, sin embargo, lo siguiente: no se conoce la Base de Datos pero, cada usuario conoce muy bien las visiones de los datos que él utiliza cotidianamente.

Esas visiones de los datos pueden ser de varios tipos: pantallas, diálogos, flujos de procesos, listados, etc. que componen el aspecto exterior de la aplicación: Aquello que es tangible para el usuario.

¿Cómo puede ayudar el conocimiento de estas visiones a obtener el Modelo de Datos?

¿Puede transformarse el asunto en un problema lógico/matemático? Si ello fuera posible, la lógica y la matemática podrían brindar una amplia gama de recursos para ayudar a resolverlo automáticamente y, como consecuencia, se simplificaría mucho la tarea del analista. Una reflexión interesante es la siguiente: si se conociera la Base de Datos, las visiones de los datos que tienen los diferentes usuarios deberían poder derivarse de ella. O, dicho de otra manera, la Base de Datos debe satisfacer a todas las visiones conocidas. Puede demostrarse que, dado un conjunto de visiones de datos de usuarios, existe siempre una Base de Datos mínima que las satisface, la cual, además, es única. En este estado, el problema se ha transformado en un problema lógico/matemático y, entonces es preciso resolverlo, para hallar esa Base de Datos.

¿Cómo se implementa esta teoría?

Se trata de capturar el conocimiento que existe en las visiones de los usuarios, y sistematizarlo en una Base de Conocimiento (todo ello en forma automática). La característica fundamental de esta Base de Conocimiento, que la diferencia de los tradicionales diccionarios de datos, es su capacidad de inferencia: se pretende que, en cualquier momento, se puedan obtener de esta Base de Conocimiento, tanto elementos que se han colocado en ella, como cualquier otro que se pueda inferir a partir de ellos.

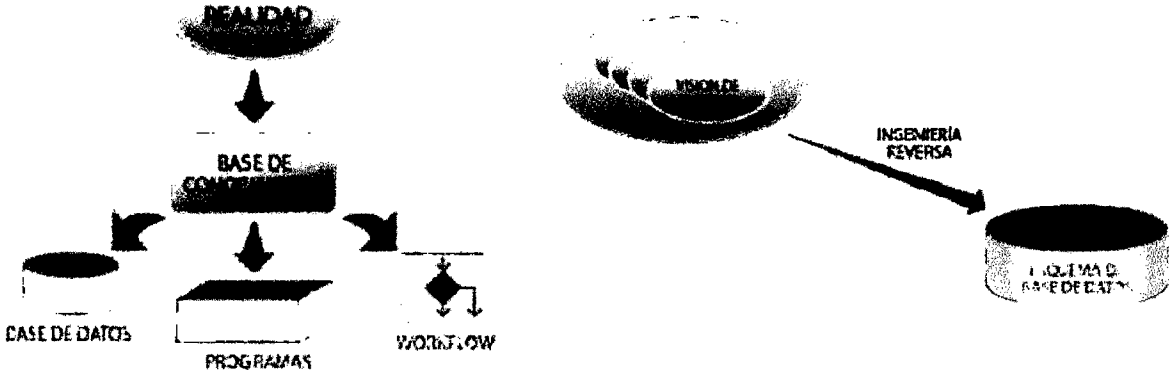
Si este objetivo se logra, la Base de Datos y los programas de aplicación pasan a ser transformaciones determinísticas de dicha Base de Conocimiento y ello permite:

- Generarlos automáticamente
- Ante cambios en las visiones de los usuarios determinar el impacto de dichos cambios sobre datos y procesos y propagar esos cambios generando:

- Los programas necesarios para convertir los datos;
- Los programas de la aplicación afectados por los cambios;
- Aquellos programas de aplicación que no han sido afectados por los cambios pero que, ahora, podrían ser sustituidos por otros más eficientes.

Partiendo de las visiones de los usuarios y haciendo uso de un procedimiento de Ingeniería Inversa Genexus obtiene como resultado el esquema de una Base de Datos relacional mínima. Ver figura Nº 19

Figura Nº 19. Esquema de trabajo de Herramienta Genexus



Fuente: www.genexus.com

4.1.2.1.4. IMPLEMENTACION DEL DESARROLLO INCREMENTAL.

Genexus es una herramienta que parte de las "visiones de los usuarios"; captura su conocimiento y lo sistematiza en una Base de Conocimiento. A partir de su Base de Conocimiento, Genexus es capaz de diseñar, generar y mantener de manera totalmente automática la estructura de la Base de Datos y los programas de la aplicación (los programas necesarios para que los usuarios puedan operar con sus visiones).

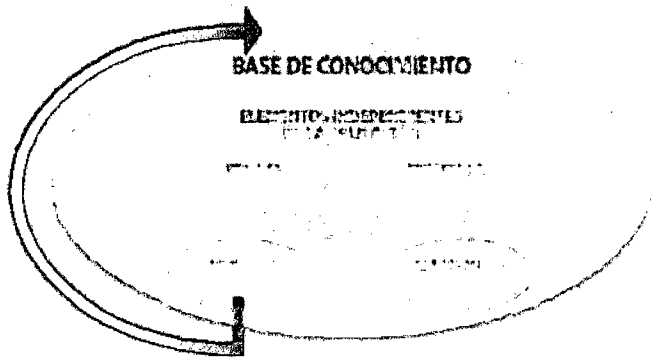
La principal fortaleza de Genexus, es una adecuada administración del conocimiento de los sistemas de negocios.

Genexus trabaja con conocimiento, lo que le permite realizar varias cosas: generar programas (software tradicional), entender ese conocimiento de los seres humanos (no necesita documentación adicional –que nunca estaría actualizada), y operar automáticamente con ese conocimiento (integrándolo

con otro proveniente de otras fuentes, difundiéndolo, otorgando licencias a terceros para que lo integren a sus aplicaciones).

Otra ventaja del trabajo con conocimiento es la posibilidad de generar aplicaciones para múltiples plataformas y múltiples arquitecturas y, muy especialmente, el poder contar con cierto tipo de "seguro" ante los cambios tecnológicos: por ejemplo, los usuarios Genexus que desarrollaron aplicaciones hace 8 o 10 años para AS/400 con pantallas de texto y tecnologías bastante primitivas, pueden ahora aprovechar el conocimiento sobre el desarrollo de esas aplicaciones que Genexus salvó para desarrollar aplicaciones Java y/o .NET con facilidad. Ver figura Nº 20

Figura Nº 20. Composición de una Base de Conocimiento Genexus



Fuente. http://www.Genexus.com

Cuando una aplicación se desarrolla con Genexus **la primera etapa consiste en hacer el Diseño de la misma registrando las visiones de usuarios (a partir de las cuales el sistema captura y sistematiza el conocimiento).**

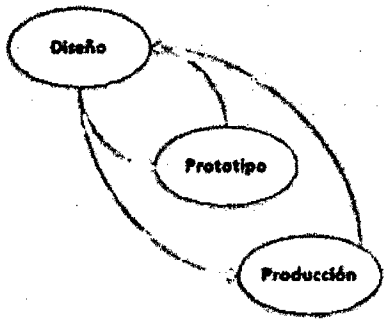
Posteriormente se pasa a la **etapa de Prototipación en donde Genexus genera la Base de Datos (estructura y datos) y programas para el ambiente de prototipo**. Una vez generado el Prototipo debe **ser puesto a prueba** por el analista y los usuarios.

Si durante la prueba del Prototipo se detectan mejoras o errores se retorna a la fase de Diseño, se realizan las modificaciones correspondientes y se vuelve al Prototipo. Llamaremos a este ciclo de Diseño/Prototipo.

Una vez que el Prototipo está aprobado, se pasa a la etapa de Implementación, en donde Genexus genera, también automáticamente, la Base de Datos y programas para el ambiente de producción.

En resumen, una aplicación comienza con un Diseño, luego se Prototipa, luego se Implementa o pone en producción y en cualquiera de los pasos anteriores se puede regresar al Diseño para realizar modificaciones. Ver figura Nº 21

Figura 21 - Ciclos Diseño-Prototipación y Diseño-Producción



Fuente. http://www.Genexus.com

A continuación se describe brevemente cada una de estos ciclos.

4.1.2.1.4.1. DISEÑO.

Esta tarea es realizada conjuntamente por el analista y el usuario, y consiste en identificar y describir las visiones de datos de los usuarios.

El trabajo se realiza en el ambiente del usuario. Este esquema permite trabajar con un bajo nivel de abstracción, utilizando términos y conceptos que son bien conocidos por el usuario final.

Una consecuencia muy importante, es que la actitud del usuario se transforma en una franca participación. El sistema pasa a ser una obra conjunta y, como el usuario sigue permanentemente su evolución, su calidad es mejor que la habitual.

De acuerdo a lo visto, Genexus captura el conocimiento por medio de visiones de objetos de la realidad del usuario. Los tipos de objetos soportados por Genexus son, entre otros: Transacciones, Reportes, Procedimientos, Work Panels, Web Panels, Data Views, Transacciones de BI.

La tarea de diseño consiste, fundamentalmente, en identificar y describir estos objetos. A partir de estas descripciones, y automáticamente, Genexus sistematiza el conocimiento capturado y va construyendo, en forma incremental, la Base de Conocimiento.

Esta Base de Conocimiento es un repositorio único de toda la información del diseño, a partir de la cual Genexus crea el Modelo de Datos físico (tablas, atributos, índices, redundancias, reglas de integridad referencial, etc.), y los programas de aplicación.

Así, la tarea fundamental en el análisis y diseño de la aplicación se centra en la descripción de los objetos Genexus.

Veamos en detalle las clases de objetos Genexus más importantes:

· **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.

Ejemplos:

- Pantalla para crear, modificar o eliminar los Clientes de la Empresa.
- Pantalla de facturación: proceso que permite a un usuario crear facturas e incluso imprimirlas.

Una pantalla permite al usuario tomar diferentes acciones como insertar, actualizar, eliminar, imprimir sin tener que volver al menú para hacerlo.

La transacción tiene elementos esenciales como la estructura de datos de la pantalla, reglas del negocio y fórmulas y elementos cosméticos como la forma de las pantallas (en este caso el desarrollador puede darle con los editores disponibles la forma que quiera u optar por utilizar la automáticamente inferida por el sistema).

- **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).

Con este objeto se pueden definir desde listados simples (por ejemplo, listar los clientes) hasta muy sofisticados, en donde existan varios cortes de control, múltiples lecturas a la Base de Datos y parametrización.

Un reporte no puede actualizar la Base de Datos.

Se dispone además de una herramienta GXquery para realizar reportes dinámicos sobre la Base de Datos.

- **Procedimientos**

Este objeto tiene todas las características de los Reportes, y además permite actualizar la Base de Datos. Los Procedimientos son comúnmente usados para tres tipos de procesos:

- **Procesos batch de actualización**. Por ejemplo: eliminar todas las facturas de fecha anterior a una fecha dada y que ya fueron pagadas
- **Subrutinas de uso general.** Por ejemplo: rutina de monto escrito en donde, dado un importe se devuelve un literal con el importe en letras (1010 => 'Mil diez')
- **Procesos a ejecutar en un servidor de aplicaciones o servidor de Base de Datos**: procesos (generalmente escritos en Java o .NET) para una Multi Tier Architecture, para ser ejecutados en un servidor de aplicaciones o de Bases de Datos.
- **Work Panels**

  Un Work Panel es una pantalla que permite al usuario realizar consultas interactivas a la Base de Datos. Cuanto más los usuarios utilizan el computador para su trabajo, se torna más necesaria la utilización de diálogos sofisticados, que le permitan sentarse a pensar frente al mismo. Los Work Panels permiten diseñar este tipo de diálogos del usuario.

  Por ejemplo: Un Work Panel que muestra la lista de clientes y que permite (a elección del usuario) ver cuales son sus facturas o su deuda.

- **Web Panels**

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

- **Data Views**

Permiten considerar correspondencias entre tablas de bases de datos preexistentes y tablas Genexus y tratar aquellas con la misma inteligencia como si fueran objetos Genexus.

4.1.2.1.4.1.1. DESARROLLO BASADO EN CONOCIMIENTO.

Partiendo de los objetos descritos, el Modelo de Datos físico es diseñado con base en la Teoría de Bases de Datos Relacionales, y asegura una Base de Datos en tercera forma normal (sin redundancia). Esta normalización es efectuada automáticamente por Genexus.

El analista puede, sin embargo, definir redundancias que, a partir de ello, pasan a ser administradas (controladas o propagadas, según corresponda), automáticamente por Genexus.

El repositorio de Genexus mantiene las especificaciones de diseño en forma abstracta, o sea que no depende del ambiente objeto, lo que permite que, a partir del mismo repositorio, se puedan generar aplicaciones funcionalmente equivalentes, para ser ejecutadas en diferentes plataformas.

4.1.2.1.4.1.2. MULTIPLES PLATAFORMAS / ARQUITECTURA DE MULTIPLES CAPAS

Como consecuencia de lo anterior es posible, por ejemplo, que un usuario de una aplicación IBM AS/400 centralizada desarrollada 100% con Genexus, quizás hace 15 años, pueda hacerla funcionar total o parcialmente en un ambiente JAVA o .NET sin tener que modificar los objetos originales.

En los últimos años se ha vuelto imperioso generar aplicaciones multi-plataforma, es decir, de ejecutar la misma aplicación en varios ambientes. Por ejemplo, la aplicación de un sistema bancario debe poder correr en un servidor iSeries o Linux en la oficina central y en una red de PCs en las sucursales del banco.

Pero eso no ha sido todo; con el uso progresivo de los ambientes Cliente/Servidor e Internet/Intranet/Extranet, ha surgido una nueva necesidad: la misma aplicación debe tener alguna de sus partes corriendo en una plataforma determinada y otras corriendo en otras plataformas. En estos casos, es también indispensable que exista una correcta intercomunicación entre las distintas partes de la plataforma.

El desarrollar aplicaciones con Genexus da la posibilidad de dividir una aplicación de manera tal que cada parte puede ser ejecutada en una plataforma diferente, utilizándose el lenguaje más apropiado para generar los programas en cada una de estas plataformas.

Esto ha dado lugar al advenimiento de las arquitecturas de múltiples capas, que a la vez optimizan el uso de los recursos disponibles.

4.1.2.1.4.2. PROTOTIPADO

En las tareas de diseño están implícitas las dificultades de toda comunicación humana:

- El usuario olvida ciertos detalles.
- El analista no toma nota de algunos elementos.
- El usuario se equivoca en algunas apreciaciones.
- El analista interpreta mal algunas explicaciones del usuario. Pero, además, la implementación de sistemas es, habitualmente, una tarea que insume bastante tiempo, por lo que:
    - Como muchos de estos problemas sólo son detectados en las pruebas finales del sistema, el costo (tiempo y dinero) de solucionarlos es muy grande.
    - La realidad cambia, por ello, no es razonable pensar que se pueden congelar las especificaciones mientras se implementa el sistema.
    - La consecuencia de la congelación de las especificaciones, es que se acaba implementando una solución relativamente insatisfactoria.

El impacto de estos problemas disminuiría mucho si se consiguiera probar cada especificación, inmediatamente, y saber cual es la repercusión de cada cambio sobre el resto del sistema.

Una primera aproximación a esto, ofrecida por diversos sistemas, es la posibilidad de mostrar al usuario formatos de pantallas, informes, etc. animados por menús. Esto permite ayudar al usuario a tener una idea de qué sistema se le construirá pero, posteriormente, siempre se presentan sorpresas.

Una situación bastante diferente sería la de poner a disposición del usuario para su ejecución, inmediatamente, una aplicación funcionalmente equivalente a la deseada, hasta en los mínimos detalles.

Esto es lo que hace Genexus: Un prototipo Genexus es una aplicación completa, funcionalmente equivalente a la aplicación de producción.

La diferencia entre prototipación y producción consiste en que la primera se hace en un ambiente de microcomputador, mientras que la producción se realiza en el ambiente objeto del usuario (IBM iSeries, servidor Linux, Cliente / Servidor, JAVA, .NET, etc.). El prototipo permite que la aplicación sea totalmente probada antes de pasar a producción. Durante estas pruebas, el usuario final puede trabajar con datos reales, o sea que prueba, de una forma natural, no solamente formatos de pantallas, informes, etc. sino también fórmulas, reglas del negocio, estructuras de datos, etc.

La filosofía de Genexus está basada en el concepto conocido como desarrollo incremental. Cuando se trabaja en un ambiente tradicional, los cambios en el proyecto hechos durante la implementación y, sobre todo, aquellos que son necesarios luego de que el sistema está implantado, son muy onerosos (y raramente quedan bien documentados).

Genexus resuelve este problema: construye la aplicación por medio de aproximaciones sucesivas que permite, una vez detectada la necesidad de cambios, prototiparlos y probarlos inmediatamente por parte del usuario, sin costo adicional.

4.1.2.1.4.3. IMPLEMENTACION

Genexus genera automáticamente el código necesario para:

- Crear y mantener la Base de Datos;
- Generar y mantener los programas para manejar los objetos descritos por el usuario.

El proceso de generación puede ser considerado en dos etapas: ESPECIFICACIÓN y GENERACIÓN. La especificación es totalmente independiente del ambiente objeto, pero la generación no. Esto significa que se puede ejecutar el mismo modelo en las diferentes plataformas de ejecución para las que se ha generado y cada una de estas versiones generadas puede ser optimizada de acuerdo con el ambiente en el cual correrá.

Los ambientes y lenguajes más importantes actualmente soportados hasta la fecha de elaboración de esta Tesis son:

BASES DE DATOS

IBM DB2 for iSeries y UDB, Informix,

Microsoft SQL Server, MySQL, Oracle y

PostgreSQL.

LENGUAJES

JAVA, C#, COBOL, RPG, Visual Basic.

SERVIDORES WEB

Microsoft IIS, Apache, WebSphere, etc.

PLATAFORMAS

Plataformas de ejecución

JAVA, Microsoft .NET, Microsoft

.NET Compact Framework

Sistemas Operativos

IBM OS/400, LINUX, UNIX,

Windows NT/2000/2003 Servers,

Windows NT/2000/XP/CE y

Windows Vista

Internet

JAVA, ASP.NET, Visual Basic (ASP),

C/SQL, HTML, Web Services

MULTIPLES ARQUITECTURAS

Arquitecturas de múltiples capas, basadas en web, Cliente/Servidor, centralizadas (iSeries)

Para conocer la lista completa de tecnologías soportadas hoy, visite:

http://www.Genexus.com/technologies

Además Genexus ofrece un conjunto de herramientas complementarias para:

- Workflow – GXflow (www.gxflow.com)
- Reporting – GXquery (www.gxquery.com)
- Business Intelligence – GXplorer (www.gxplorer.com)
- Portal Building – GXportal (www.gxportal.com)

4.1.2.1.4.4. MANTENIMIENTO

Esta es una de las características más importante de Genexus, y la que lo diferencia de manera más clara de las demás herramientas: el mantenimiento, tanto de la Base de Datos (estructura y contenido) como de los programas, es totalmente automático.

A continuación se explicará el proceso de mantenimiento, ante cambios en la descripción de algún objeto Genexus (visión del usuario):

Impacto de los cambios sobre la Base de Datos

Análisis de impacto

Una vez descritos los cambios de las visiones de usuarios, Genexus analiza automáticamente cual es el impacto de los mismos sobre la Base de Datos y produce un informe donde explica como debe hacerse la conversión de los datos y, si cabe, qué problemas potenciales tiene esa conversión (inconsistencias por viejos datos ante nuevas reglas, etc.). El analista decide si acepta el impacto y sigue adelante o no.

Generación automática de programas de conversión

Una vez que los problemas han sido solucionados o bien se ha aceptado la conversión que Genexus sugiere por defecto, se generan automáticamente los programas para hacer la conversión (estructura y contenido) de la vieja Base de Datos a la nueva.

Ejecución de los programas de conversión

A continuación, se pasa al ambiente de ejecución que corresponda (prototipo, producción Internet, producción Cliente / Servidor, etc.) y se ejecutan los programas de conversión.

Impacto de los cambios sobre los programas

Análisis de impacto

Genexus analiza el impacto de los cambios sobre los programas, y produce un diagnóstico informando qué programas deben generarse o re-generarse y proporcionando también, para el nuevo programa, o bien el diagrama de navegación o bien un pseudo-código, a elección del analista.

Generación de nuevos programas

A continuación el sistema genera o regenera automáticamente todos los programas.

4.1.2.1.4.5. DOCUMENTACION

Todo el conocimiento provisto por el analista, o inferido por Genexus, está disponible en un repositorio activo, que constituye una muy completa documentación online, permanentemente actualizada.

La documentación incluye la descripción de objetos específicos e información sobre la Base de Conocimiento resultante y sobre la Base de Datos diseñada.

La Base de Conocimiento de Genexus no solamente le permite acceder al conocimiento que almacena siempre que el desarrollador lo desee sino que también le habilita el acceso a toda la información inferida lógicamente (una regla de integridad referencial, un mapa de navegación en la Base de

Datos, un análisis de impacto de cambios, referencias cruzadas, diagramas E-R inferidos a partir del conocimiento almacenado, etc.).

4.1.2.1.4.6. CONSOLIDACION DE VARIAS APLICACIONES Y REUTILIZACION DEL CONOCIMIENTO.

Varias aplicaciones pueden ser diseñadas y prototipadas simultáneamente, por diferentes equipos, utilizando Genexus. Estos equipos pueden intercambiar especificaciones de diseño utilizando el Genexus Knowledge Manager. Este modulo le permite hacer lo siguiente automáticamente:

- Comenzar el diseño de una nueva aplicación basada en Objetos del Negocio, Patrones de Software, Dominios, Atributos y/o Estilos de un dominio público (consulte: http://www.gxopen.com.uy).
- Distribuir conocimiento desde una Base de Conocimiento corporativa a la Base de Conocimiento de otra aplicación.
- Verificar la concordancia entre la Base de Conocimiento de una aplicación y la corporativa.
- Consolidar dos aplicaciones (es especialmente útil consolidar el conocimiento de una aplicación dada a la Base de Conocimiento corporativa).

Esto permite una flexibilidad ideal: el analista trabaja con entera libertad en un ambiente de prototipo, con una pequeña Base de Conocimiento y, sólo cuando su aplicación está lista desde el punto de vista del usuario, debe tomarse en cuenta la Base de Conocimiento corporativa, que generalmente será muy grande.

En ese momento, con ayudas automáticas, se establece el impacto que tendrá la nueva aplicación, o la modificación de la preexistente, sobre el modelo corporativo y, si es el caso, se hacen los cambios para asegurar la consistencia, de una manera muy simple.

Con este esquema es posible reutilizar, por ejemplo, Bases de Conocimiento licenciadas de terceras partes.

Al comienzo es necesario usar una nomenclatura común entre las diferentes bases de conocimiento involucradas en la consolidación. No obstante, la funcionalidad "Adapt From" le permite definir un mapeo para la conversión de nombres para adaptarse a la nomenclatura objetivo.

También es importante señalar que la casa de software que otorga la licencia sobre la Base de Conocimientos Genexus puede mantener partes de la misma en privado; de esta manera podrá permitir su uso automático sin revelar sus fuentes.

Además, es posible que un objeto sea declarado como público o privado. Todos pueden ser usados automáticamente por Genexus, pero en el caso de los objetos privados, solo el dueño puede ver y/o modificar la fuente de alto nivel de Genexus.

Es de destacar la característica que permite sobre una misma Base de Conocimiento generar aplicaciones sobre varios idiomas lo que colabora mucho para que las aplicaciones puedan utilizarse internacionalmente.

4.1.2.1.5. CARACTERISTICAS DE GENEXUS

Genexus tiene algunas características que lo distinguen de las demás herramientas. Entre ellas pueden destacarse:

1. **El diseño parte de las visiones proporcionadas por los usuarios**. Debido a sus actividades diarias, ellos son quienes saben cómo deben y cómo no deben funcionar las cosas.
2. **La descripción de cada objeto es totalmente independiente de la de los demás** por lo que, en el caso de que se deba modificar la descripción de uno, ello no implicará la necesidad de modificar manualmente la descripción de cualquier otro. Esta característica exclusiva de Genexus (ortogonalidad de las descripciones) es la que permite un mantenimiento totalmente automático de las aplicaciones.
3. **La curva de aprendizaje** es corta.

4. **El diseño, creación y mantenimiento de la Base de Datos** son totalmente automáticos.
5. **La aplicación (Base de Datos y programas)** son siempre, sean cuales sean las modificaciones que haya sufrido, de la mejor calidad:
6. **La Base de Datos** es siempre la óptima (tercera forma normal).
7. **No se modifican programas**: cuando ya no son adecuados, se generan otros nuevos, óptimos y no parchados, que los sustituyen.
8. **Utilización los archivos o bases de datos preexistentes** como propios de Genexus.
9. **Lenguajes poderosos y de muy alto nivel** para la definición de Procesos, Work Panels y Web Objects. En estos lenguajes las descripciones de los procesos se hacen sin referirse a los archivos involucrados, los que son inferidos automáticamente en tiempo de generación. Esta característica permite una total independencia entre los datos y dichas especificaciones. Como consecuencia, las especificaciones de alto nivel de Genexus no necesitan modificaciones ante modificaciones de la Base de Datos.
10. **Mantenimiento 100% automático**: El conjunto de estos elementos permite a Genexus generar y mantener automáticamente el 100% de los programas en aplicaciones de negocios (comerciales, administrativas, financieras, industriales, etc.).
11. **Genexus funciona en PCs**, dejando al entorno de producción totalmente libre para el procesamiento de las aplicaciones.
12. **Fácil distribución del conocimiento corporativo** para facilitar el desarrollo de nuevas aplicaciones.
13. **Soluciones de Reportes y Data Warehousing** simples y potentes.

14. **Verificación automática de consistencia, y consolidación**, entre aplicaciones desarrolladas separadamente.
15. **Independencia** de plataforma y arquitectura.
16. **Simplicidad:** Genexus utiliza los recursos avanzados de la inteligencia artificial para que el analista y los usuarios, puedan usarlo de una forma muy simple.

4.1.2.1.6. FUNCIONALIDADES AVANZADAS.

4.1.2.1.6.1. COMPONENTES WEB.

Los Web Components permiten a los diseñadores de aplicaciones GeneXus Web un alto grado de reutilización de los mismos.

Son objetos web que tienen una propiedad en la cual se indica que son componentes. Es decir, pueden ser ejecutados en forma independiente (como cualquier otro objeto web) o pueden formar parte de otro objeto web (Web Panel o Web Transaction).

Cualquier parte de un objeto web que se repita en varios objetos de una aplicación web, puede ser definida como Web Component.

Los ejemplos más comunes son: menús, login, área que permite la personalización, etc.

La idea entonces es, en lugar de tener implementado, por ejemplo, la carga del menú en cada uno de los objetos web que requieren el mismo, programarla en un Web Component y reutilizarlo en cada Web Panel que requiere un menú.

Cómo definir un Web Component Para definir un objeto web como Web Component se debe configurar la propiedad "Type" (antes Web Component) con el valor "Component". De esta forma, se habilita la propiedad "URL access". La cual puede tomar los siguientes valores:

- **Yes**: permite que el objeto sea ejecutado desde la URL.
- **No**: no permite que el objeto sea ejecutado desde la URL.

Se debe notar que un objeto web definido como Web Component no pierde ninguna de sus características, por lo tanto, puede ser ejecutado en forma autónoma.

Los Web Components se generan dentro del mismo HTML del Web Panel que los contiene. Esto significa que el servidor resuelve la inclusión

del Web Component en tiempo de ejecución y devuelve al navegador el código HTML con el Web Component ya incluido.

4.1.2.1.6.2. OBJETO THEME.

Las aplicaciones Web necesitan tener una presentación bastante más sofisticada que las aplicaciones Win y los desarrolladores generalmente no tienen las habilidades y el tiempo para lograr un diseño con esas características.

Por esa razón, por lo general se opta por contratar servicios de diseño. Por lo cual, es necesario luego integrar ese diseño con la aplicación GeneXus. Esta integración se logra a través de los "Themes".

Para el diseño gráfico Genexus cuenta con el "Editor de Themes", una herramienta de libre distribución que se puede ejecutar en forma independiente de GeneXus.

Mientras que en versiones anteriores las propiedades de los controles en los "html forms" (Web Panels y forms html de las Transacciones) debían ser configuradas en los controles, hoy mediante los "Themes" se pueden definir "Clases" para los diferentes tipos de controles y asignarles propiedades a esas clases.

Luego, los controles se asocian a esas clases, y éstos heredarán las propiedades allí configuradas. Por lo tanto ya no es necesario establecer los valores de las propiedades para cada control en el form.

Las propiedades de los controles se configuran en un único lugar: en las clases del Theme.

Cualquier cambio de diseño que se requiera hacer, se realizará en el Theme (sin necesidad de generar/compilar absolutamente nada), de esta forma se logra la uniformidad estética del sitio con un bajo costo de mantenimiento (no es necesario cambiar cada uno de los objetos), y como consecuencia se obtiene una mayor productividad en el desarrollo de aplicaciones Web.

4.1.2.1.6.3. PATRONES (PATTERNS)

Patrones es una herramienta, que se puede utilizar tanto desde Genexus como en forma independiente. Permitiendo aplicar a una base de conocimiento cierto patrón (pattern), y que se generen todos los objetos

Genexus necesarios para implementar el patrón, para aquellas instancias que se hayan seleccionado.

Los patterns y las formas de utilizar dicha herramienta son un medio para obtener una mayor productividad en el desarrollo de las aplicaciones Web.

Se aplica esta herramienta en nuestra base de conocimiento para definir nuevos objetos Genexus.

Es común percibir al desarrollar aplicaciones que algunas partes son muy similares pero no exactamente iguales.

Por ejemplo, si se tienen clientes y productos, si bien los actores de la realidad (clientes y productos) son totalmente diferentes, los "Trabajar con" para cada uno de ellos tienen muchas cosas en común:

- Un Grid en el form
- Un conjunto de variables para utilizar en filtros
- Ordenamiento de los datos que se presentan.
- Acciones que se ofrecen de actualización de la base de datos, consultas, etc.

Esta situación es llamada "Pattern" (patrón) y es un tema muy popular en la industria del software.

Dado que con Genexus se trabaja a nivel de conocimiento y no de código, aplicamos el concepto de patrones a reusar el conocimiento.

Genexus a desarrollado un marco de trabajo con las siguientes características:

- Activo: Se generan objetos GeneXus que implementan cierto patrón para una o varias instancias elegidas.
- Reutilización de conocimiento: Se trata de un nuevo nivel de reutilización de conocimiento, ya que el mismo conocimiento puede utilizarse para diferentes situaciones.
- Abierto: Cualquier persona puede crear un patrón o modificar uno existente.
- De fácil uso: (A excepción de la creación de nuevos patrones).

Descripción de la herramienta

Patterns es una herramienta que es parte de Genexus (**Tools / Patterns**), que puede ejecutarse también como aplicación independiente.

El objetivo de esta herramienta es el incremento de la productividad y calidad en las aplicaciones.

Básicamente la herramienta Patterns permite aplicar a una base de conocimiento cierto patrón (Pattern), y generar todos los objetos GeneXus que implementen el Pattern para aquellas instancias que se hayan seleccionado. Es importante tener en cuenta que la base de conocimiento debe estar en diseño para poder utilizarla.

Existe un conjunto de Patterns muy útiles ya definidos (al instalar la herramienta se instalan) para que el desarrollador pueda comenzar a usar dichos patrones rápidamente.

A su vez la herramienta permite crear Patterns nuevos, siendo esta una tarea un poco más compleja.

Requerimientos

Esta herramienta se puede utilizar únicamente con los generadores .NET y Java. Para obtener más detalles de los requerimientos de software de instalación ver la siguiente página:

http://wiki.gxtechnical.com/wiki/tiki-index.php?page=PatternsInstallation

Patterns existentes

Hasta ahora los Patterns con los que cuenta Genexus son:

Work With: "Trabajar con" en Web; genera a partir de una Transacción (o para todas aquellas Transacciones que se deseen), todos los objetos necesarios para tener una aplicación Web: trabajar con, views, seguridad, etc..

Bill of materials: Permite generar a partir de una Transacción, otra que representa la relación compuesto–componente.

OAV: Objeto-Atributo-Valor; genera a partir de una Transacción otras dos Transacciones que permiten extender la original, con el objetivo de permitir definir atributos en tiempo de ejecución.

Además, seguimos trabajando en la definición de nuevos Patterns. Tenemos un catálogo con una lista de Patterns, algunos de los cuales ya están implementados y otros sugeridos para una futura implementación. Para acceder al catálogo:

http://wiki.gxtechnical.com/wiki/tiki-index.php?page=Business+Patterns+Catalog

4.1.2.1.6.4. SEGURIDAD

La seguridad en el desarrollo de una aplicación para Internet es un concepto muy amplio, la respuesta a esta pregunta no es sencilla. Sin embargo, dependiendo del tipo de aplicación que se está desarrollando, este punto puede ser crucial.

Se puede decir que al publicar una aplicación en Internet, se tienen 3 niveles distintos de seguridad:

- Seguridad a nivel del Servidor Web
- Seguridad a nivel de la Base de Datos
- Seguridad a nivel de la aplicación

A continuación trataremos cada uno de los puntos mencionados anteriormente.

SEGURIDAD A NIVEL DEL SERVIDOR WEB

Cuando se habla de la seguridad a nivel del servidor Web, se están enfrentando básicamente los siguientes riesgos:

- Intercepción de información enviada a través de la red desde el browser al servidor o viceversa,
- Errores o problemas de configuración del servidor Web, que permitan a usuarios no autorizados:
- Acceder a información confidencial que se encuentra en el servidor.
- Ejecutar comandos que afecten el comportamiento del servidor y les permita

ingresar al sistema.

Intercepción de información

El protocolo TCP/IP no fue diseñado para ofrecer servicios de comunicación seguros. En consecuencia, la información enviada desde el browser al servidor Web o viceversa puede ser interceptada por alguien. Es por esta razón que se debe agregar tecnología adicional para resolver problemas de seguridad.

Desde siempre, existe una única tecnología que provee los principios para resolver estos problemas:

CRIPTOGRAFIA.

Existen servidores Web, que pueden trabajar como servidores seguros. Un servidor seguro es un servidor Web, al cual se le habilitó la posibilidad de encriptar los datos que se envían al browser, así como desencriptar la información recibida del mismo.

El standard más conocido de autenticación y encriptación de datos para el Web es el SSL. Este standard fue propuesto por Netscape Comunications Corporation y es el que soportan la mayoría de los browsers (Internet Explorer, Netscape).

Para habilitar la encriptación de datos se necesita obtener un certificado de una autoridad certificadora (por Ej.: Verisign, El Correo (en Uruguay)), este certificado es instalado en el servidor Web, quedando habilitada la encriptación de datos. Finalmente, para activar la encriptación de los datos a transferir, todo lo que el usuario debe hacer es realizar la solicitud del recurso a través del protocolo HTTPS: en lugar de HTTP.

La seguridad ofrecida por el servidor Web, depende en parte también del largo del certificado soportado. Cuanto más largo sea el mismo más segura es la comunicación de los datos. Hay que destacar que los servidores seguros protegen "únicamente" la información confidencial de ser interceptada por terceros, pero sin seguridad a nivel del sistema el servidor Web sigue siendo vulnerable a ataques.

La property "Protocol Specification" aplica a la generación automática de links entre Web Panels, y el objetivo es identificar el protocolo por defecto a usar cuando se construyen los links (HTTP, HTTPS).

Acceso de usuarios no autorizados

La mayor parte de los riesgos que se presentan al tener un servidor Web público son asumidos por el administrador del sitio. Desde el momento que se instala un servidor Web en un sitio, se abre para la comunidad Internet una "ventana" a la red de área local. La mayor parte de los usuarios únicamente visitan el sitio, otros, sin embargo intentarán entrar por la "ventana"

abierta. Los resultados pueden ser variados, desde el simple descubrimiento que la página principal del sitio Web fue cambiada, hasta el robo de la base de datos que contiene la información de sus clientes.

Es entonces importante que el administrador del sistema defina políticas de seguridad, así como tome las medidas necesarias para evitar el acceso de usuarios no deseados. Estas medidas pueden ser a nivel del sistema operativo (restringir las operaciones que se pueden realizar, limitar los permisos sobre documentos y directorios, deshabilitar servicios no utilizados, etc.), como a nivel del servidor (aislamiento de la red, instalación de firewall).

4.1.2.1.6.5. ARQUITECTURA AJAX: Implementación con Genexus

Ajax es una arquitectura de software que transforma radicalmente la experiencia del usuario Web, y su nombre es el acrónimo de ***Asynchronous JavaScript + XML***.

Al cargar una página Web al inicio de la sesión, el navegador carga el motor Ajax que opera como intermediario entre la aplicación del usuario y el servidor. Este motor de Ajax es responsable de desplegar la interface que el usuario ve, y de comunicar esta interface con el servidor de acuerdo al comportamiento del usuario.

El motor de Ajax permite que la interacción del usuario con la aplicación sea asincrónica e independiente de la comunicación con el servidor, de manera que el usuario nunca va a tener que estar esperando que el servidor ejecute tal o cual acción para verla reflejada en su pantalla.

Cada acción del usuario toma la forma de un llamado Javascript al motor de Ajax. El motor se encarga de cualquier acción simple del usuario sin requerir una respuesta del servidor, por ejemplo para validar datos, editar datos en memoria o navegar. Si el motor necesita de alguna respuesta del servidor por el procesamiento de datos, para cargar código adicional a la interface, o para acceder a nuevos datos- el motor de Ajax realiza estos pedidos asincrónicamente, generalmente usando XML, sin detener la interacción entre el usuario y la aplicación a la espera de la respuesta del servidor.

Esto se comprueba en las aplicaciones web que se generan con GeneXus y sin necesidad de hacer nada extra en GeneXus para incluir estas

ventajas: las aplicaciones GeneXus se generan por defecto con esta funcionalidad.

Las funcionalidades basadas en Ajax que incluye GeneXus son:

- Input type "Values" or "Descriptions"
- Suggest (intellitips)
- Web client-side validation
- Dynamic combo boxes with filters

4.1.2.1.6.6. WEB SERVICES: Implementación con Genexus.

Un Web Service es una aplicación que puede ser descripta, publicada, localizada e invocada a través de una red, generalmente Internet. Combinan los mejores aspectos del desarrollo basado en componentes y la Web.

Al igual que los componentes, los Web Services son funcionalidades que se encuentran dentro de una caja negra, que pueden ser reutilizados sin preocuparse de cómo fueron implementados. A diferencia de la actual tecnología de componentes, no son accedidos por medio de protocolos específicos del modelo de objetos como ser RMI, DCOM o IIOP; sino que son accedidos utilizando protocolos Web como ser HTTP y XML.

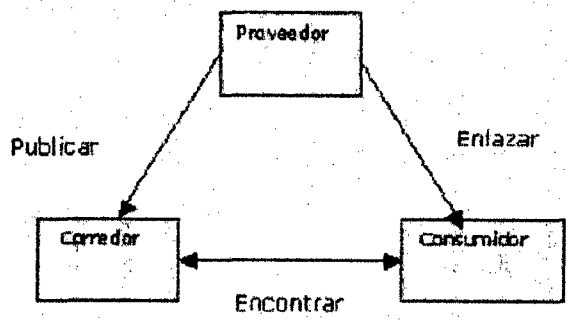
La interface de los Web Services está definida en términos de los mensajes que el mismo acepta y retorna, por lo cual los consumidores de los Web Services pueden ser implementados en cualquier plataforma y en cualquier lenguaje de programación, sólo tiene que poder crear y consumir los mensajes definidos por la interface de los Web Services.

El modelo de Web Services La arquitectura básica del modelo de Web Services describe a un consumidor, un proveedor y ocasionalmente un corredor (broker). Relacionados con estos agentes están las operaciones de publicar, encontrar y enlazar. Ver figura Nº 22

La idea básica consiste en que un proveedor publica su servicio en un corredor, luego un consumidor se conecta al corredor para encontrar los servicios deseados y una vez que lo hace se realiza un lazo entre el consumidor y el proveedor.

Cada entidad puede jugar alguno o todos los roles.

Figura Nº 22. Esquema de un Web service



Fuente. http://www.Genexus.com

A continuación se detallan los mecanismos para proveer y consumir Web Services con Genexus.

¿Como Genexus Provee Web Services.?

Para proveer Web Services con GeneXus se debe definir un objeto Genexus Procedimiento o Reporte, que debe ser de tipo main, y asociado a un generador Web.

Se debe configurar la propiedad de objeto "Call protocol" con el valor "SOAP".

Se debe publicar el objeto en un servidor Web, con lo que el Web Service queda operativo para ser invocado.

El WSDL del servicio queda disponible agregando el parámetro wsdl a la URL del mismo. Por ejemplo:

http://server/baseurl/aservice.aspx?wsdl

¿Cómo Consumir Web Services con Genexus?

Se utiliza el WSDL Inspector, que permite importar la definición del servicio que se desea consumir.

Se debe indicar el WSDL del mismo, y el utilitario definirá los tipos de datos necesarios para poder consumirlo desde GeneXus.

En el caso de que el Web Service utilice tipos de datos estructurados, los mismos serán agregados a la base de conocimiento al importar la definición con el WSDL Inspector.

Además, es necesario configurar la propiedad de objeto Location. Esto servirá luego para configurar, en los objetos GeneXus que vayan a invocar al Web Service, la ubicación del mismo (host, port, etc.).

4.1.2.2. PROTOTIPOS

Esta técnica se refiere a la creación de prototipos no desechables del sistema de software en fase de desarrollo desde las fases iniciales del proyecto. Permite el descubrimiento precoz de las deficiencias del sistema y permite a los futuros usuarios de del sistema de software llevar a cabo pruebas exhaustivas. De esta forma se concreta una buena participación de los usuarios, uno de los factores clave del éxito de MDDS-GX , o cualquier proyecto de Sistema de Desarrollo de software.

4.1.2.3. PRUEBAS

Uno de los objetivos importantes de la metodología MDDS-GX es la creación de un sistema de software de buena calidad. A fin de entregar una solución de buena calidad, MDDS-GX obliga a realizar las pruebas a través de cada iteración. MDDS-GX es independiente del método, herramienta o técnica de prueba, el equipo del proyecto es libre de elegir su propio método de gestión de pruebas.

4.1.2.4. TALLERES

Una de las técnicas que tiene como objetivo reunir a los diferentes actores del proyecto de software para discutir las necesidades, las funcionalidades y la comprensión mutua. En un taller los participantes se reúnen y discuten el proyecto.

4.1.2.5. MODELADO

Esta técnica es fundamental y se utiliza con el propósito de visualizar la representación esquemática de un aspecto específico del sistema o área de negocio que se está desarrollando. El Modelado proporciona al equipo de desarrollo MDDS-GX un mejor entendimiento del proyecto, sobre un dominio del negocio.

4.1.2.6. GESTION DE LA CONFIGURACION

Una buena implementación de la técnica de administración de la configuración es importante debido a la naturaleza dinámica de la MDDS-GX.

Ya que los productos se entregan con frecuencia a un ritmo muy rápido, por lo tanto los productos deben ser controlados estrictamente, hasta lograr su culminación.

La herramienta Genexus permite controlar las diferentes versiones que se realizan de un determinado sistema de software.

4.2. METODOLOGIA DE DESARROLLO DE SOFTWARE PROPUESTA.

En este capítulo se presenta la propuesta de este trabajo de Tesis, definición de Metodología Agil para Desarrollo de Software, Apoyada en la Herramienta CASE GENEXUS. A la que denominaremos ***"Metodología Dinámica para Desarrollo de Software Apoyada con Genexus"***, a la que en adelante la referenciaremos como **MDDS-GX**; se centra en el desarrollo de Sistemas de Software Comerciales Corporativos, tanto para escritorio como para la web tomando como base el Ciclo de Vida modificado de la Metodología Agil Dynamic Systems Development Method, la cual está en su cuarta versión y se dice que ahora las iniciales DSDM significan Dynamics Solutions Delivery Method, ya no se habla de Sistemas sino de Soluciones, y en lugar de priorizar el Desarrollo se prefiere enfatizar la entrega; MDDS-GX se complementa además con las mejores prácticas de las Metodologías de Programación Extrema (XP), Entrega Evolutiva, Prototipado Evolutivo, del desarrollo en ventanas temporales y sobre todo apoyándose en la Herramienta CASE GENEXUS, la cual está basada en una rama de la Inteligencia Artificial - la Teoría del Conocimiento.

MDDS-GX es una metodología de desarrollo de software que tiene un enfoque iterativo e incremental que enfatiza la continua participación de los usuarios.

Su objetivo es entregar sistemas de software oportunamente teniendo en cuenta las necesidades cambiantes que normalmente se presentan a lo largo del proceso de desarrollo así como satisfacer la hipótesis, objetivo principal y objetivos específicos de la investigación definidos en la sección 1.2.

El presente capítulo se estructura de la siguiente manera: en la sección 4.2.1. se describen los principios o prácticas en los que se basa; en la sección 4.2.2. se describen las fases de la Metodología, en la sección 4.2.3. se describen los roles, en la sección 4.2.4. se describe la naturaleza iterativa e incremental de la Metodología y posteriormente en la sección 4.2.5. se describen los factores críticos de éxito de la Metodología propuesta.

4.2.1. PRINCIPIOS Y PRACTICAS EN QUE SE BASA LA MDDS-GX.

1. Involucrar al usuario es la clave principal en la gestión de un proyecto de desarrollo de software eficiente y eficaz, donde los usuarios y desarrolladores comparten un lugar de trabajo, para que las decisiones se pueden hacer con precisión.
2. El equipo del proyecto debe estar facultado para tomar decisiones que son importantes para el avance del proyecto sin esperar a la aprobación de más alto nivel.
3. El sistema de software debe construirse usando herramientas CASE que soporten la generación automática de código.
4. Se desarrolla primero partes seleccionadas del sistema en un prototipo, aquellas áreas más relevantes del sistema, y luego se continúa aumentando y refinando el prototipo hasta que termina el desarrollo del resto del sistema, el prototipo evoluciona y se transforma en el sistema de software a entregar.
5. La construcción del software consiste en el desarrollo de un prototipo no desechable y su evolución hasta obtener el sistema de software final en estado operativo, para lo cual es necesario una fuerte implicación del usuario final, y revisiones constantes del sistema de software en desarrollo.
6. Desde un punto de vista de la motivación es esencial que el equipo de desarrollo cree su propia estimación tanto el tiempo que necesita como de la funcionalidad que puede construir e implementar dentro del periodo estimado.
7. Al principio del proyecto es necesario tener una idea fundamental del tipo de sistema de software que se va a construir, se comienza con una idea preliminar de lo que los clientes desean, y basándose en esto se crea la

arquitectura o un diseño global y el núcleo del sistema de software. Esta arquitectura y este núcleo sirven como base para el desarrollo del sistema de software.

8. Organizar y planificar un conjunto preliminar de entregas al comienzo del proyecto que probablemente contendrá alguna funcionalidad que claramente estará en el sistema final, alguna que probablemente estará, y otra que podría no estar. Sin embargo, se identificarán otras funcionalidades y se añadirán posteriormente. Asegurando de estructurar la planificación de entregas, de forma que desarrolle antes las funcionalidades más importantes. Puede mostrar el sistema a sus clientes, obtener, obtener su realimentación, y tener la seguridad de que el resto de funcionalidades es realmente lo que desean.
9. Antes de que comience la construcción del software es necesario definir el marco de las funciones y un diseño preliminar del sistema
10. Centrar la atención en la entrega frecuente de productos, con el supuesto de que para ofrecer algo "suficientemente bueno" antes, es siempre mejor que entregar todo "perfectamente" al final. Al entregar el producto con frecuencia desde las primeras fases del proyecto, el producto puede ser probado y examinado, el registro de pruebas y revisión de documentos pueden ser tenidos en cuenta en la siguiente iteración o fase.
11. Los principales criterios para la aceptación de una "entrega" son el presentar un sistema que responda a las necesidades empresariales actuales. Entrega de un sistema perfecto en el que se aborden todas las necesidades posibles del negocio es menos importante que centrarse en las funciones críticas.
12. El desarrollo es iterativo e incremental, de forma que pueda modificarse de manera inmediata, guiado y en respuesta a la realimentación del usuario final, para converger en una solución del negocio precisa.
13. Todos los cambios durante el desarrollo son reversibles.
14. El alcance de alto nivel y los requerimientos deberían ser la línea-base antes de que comience el proyecto.
15. La prueba está integrada a través de todo el ciclo de vida. La prueba también es incremental. Se recomienda particularmente la prueba de regresión, de acuerdo con el estilo evolutivo del desarrollo.

16. La comunicación y cooperación entre todas las partes interesadas en el proyecto es un prerrequisito importante para llevar a cabo un proyecto efectivo y eficiente.
17. Es esencial una estrategia colaborativa y cooperativa entre todos los participantes. Las responsabilidades son compartidas y la colaboración entre usuarios y desarrolladores no debe tener fisuras.
18. El equipo mínimo es de dos personas y puede llegar a seis, pero puede haber varios equipos en un proyecto. El mínimo de dos personas involucra que un equipo consista de un Desarrollador y un usuario.
19. Refactorización[1] contínua. Se refactoriza el sistema de software eliminando duplicación, mejorando la comunicación y agregando flexibilidad sin cambiar la funcionalidad. El proceso consiste en una serie de transformaciones que modifican la estructura interna preservando su comportamiento aparente. Se recomienda herramientas automáticas. En sus comentarios a [Highsmith, 2000].

[1)] ***El término refactoring, introducido por W.F. Opdyke en su tesis doctoral [Opdyke,*** *1992], se refiere al proceso de cambiar un sistema de software (orientado a objetos) de tal manera que no se altere el comportamiento exterior del código, pero que se mejore su estructura interna. Normalmente se utilizan herramientas automáticas para hacerlo (DMS, Genexus), y/o se implementan técnicas tales como aserciones (invariantes, pre- y poscondiciones) para expresar propiedades que deben conservarse antes y después de la refactoría. Otras técnicas son transformaciones de grafos, métricas de software, refinamiento de programas y análisis formal de conceptos [Tom Mens y Arie Van Deursen, 2003]. Al igual que sucede con los patrones, existe un amplio catálogo de refactorizaciones más comunes: reemplazo de iteración por recursión; sustitución de un algoritmo por otro más claro; extracción de clase, interfase o método; descomposición de condicionales; reemplazo de herencia por delegación, etcétera. La Biblia del método de Refactoring de Martin Fowler, Kent Beck, John Brant, William Opdyke y Don Roberts [Fowler, Beck, Brant, Opdyke y Roberts, 1999]*

4.2.2. METODOLOGIA MDDS-GX - CICLO DE VIDA, FASES

En el figura Nº 23, se muestra las fases del ciclo de vida de la MDDS-GX para el desarrollo de un sistema de software. Todas estas fases son iterativas e incrementales y se complementan unas a otras. Las fases son las siguientes:

1) Estudio del Sistema de software a Desarrollar,
2) Diseño
3) Prototipo
4) Producción
5) Implementación.

4.2.2.1. ESTUDIO DEL SISTEMA A DESARROLLAR.

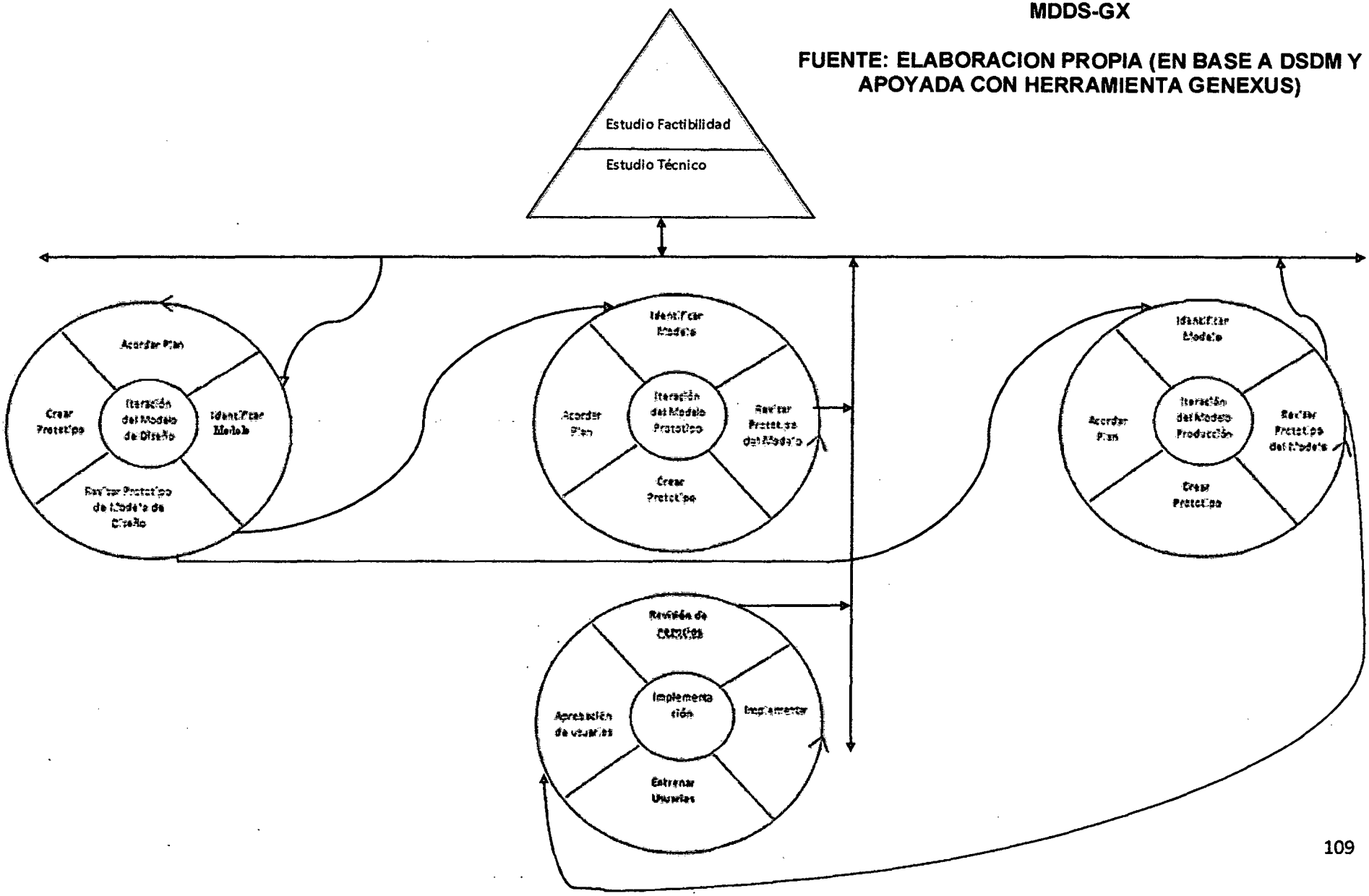
Esta fase se la ha dividido en 2 subfases:

4.2.2.1.1. ESTUDIO DE FACTIBILIDAD

En esta sub fase es donde se evalúa la idoneidad de la aplicación de la MDDS-GX. Teniendo en cuenta el tipo de proyecto, la organización y la gente, se toma la decisión, ya sea de usar o no la MDDS-GX. Se dirige a responder las preguntas siguientes: ¿Este proyecto de Desarrollo de Software puede satisfacer las necesidades que tiene la empresa?, ¿Este proyecto de Desarrollo de Software es adecuado para aplicar la MDDS-GX? y ¿cuáles son los riegos más importantes que están en juego?. La técnica más importante a usar en esta fase son los talleres, que tienen como objetivo reunir a los diferentes actores del proyecto de desarrollo de software para discutir sus necesidades, las funcionalidades y los requerimientos comunes. En un taller los participantes se reúnen y discuten el proyecto de Desarrollo de Software. Como resultado de esta fase se generará un informe de viabilidad, un esquema del plan global del proyecto de desarrollo de software y un registro de los riegos más importantes identificados.

FIGURA Nº 23 - CICLO DE VIDA DE LA METODOLOGIA MDDS-GX

FUENTE: ELABORACION PROPIA (EN BASE A DSDM Y APOYADA CON HERRAMIENTA GENEXUS)



4.2.2.1.2. ESTUDIO DEL NEGOCIO

En esta sub fase se estudian los procesos de negocio, reuniendo a grupos de usuarios involucrados y sus respectivas necesidades y deseos. Aquí también son los talleres una de las principales técnicas a usar, donde las diferentes partes interesadas se reúnen para discutir el sistema de software propuesto. La información obtenida de estas sesiones es: Una lista de requerimientos. Una propiedad importante de la lista de requerimientos es el hecho de que los requerimientos están priorizados. Teniendo en cuenta esta priorización, se elabora un bosquejo del Plan de Desarrollo el cual se constituye en una guía para el proyecto de desarrollo de software.

Una técnica importante que se utiliza en el desarrollo de este plan es la timeboxing. Esta técnica es esencial en la consecución de los objetivos de MDDS-GX, es decir, llegar a tiempo y dentro del presupuesto, garantizando la calidad deseada. Así también una arquitectura preliminar del sistema ayuda a guiar el desarrollo de las prestaciones del sistema de software. Así como una delimitación de la zona del negocio dentro del contexto de la empresa, que estará comprendida en el proyecto. Una definición de la arquitectura global inicial del sistema de software en fase de desarrollo, junto con un Plan de Desarrollo que describe los pasos más importantes en el proceso de desarrollo. En base de estos dos últimos documentos se encuentra la lista de requerimientos prioritarios. Esta lista indica todos los requerimientos del sistema,. Y por último, el registro de riesgos actualizado con los hechos identificados en esta sub fase de la MDDS-GX.

En esta fase se determinarán los objetos Genexus transacciones, las cuales provienen de los sustantivos identificados en las descripciones de las visiones de la realidad proporcionados por los usuarios expertos en el negocio, estos objetos conformarán la Base de Conocimiento.

En el caso de aplicaciones corporativas conformadas por varias aplicaciones, se seleccionará aquellas transacciones que son comunes a las diversas aplicaciones, Se determinarán los objetos correspondientes a cada módulo y se conformarán los equipos de desarrollo, los cuales estarán compuestos por uno o más desarrolladores, dependiendo del número de

objetos que conforman cada módulo así como por su grado de dificultad. por ejemplo: Para una aplicación corporativa que esté conformada por las aplicaciones de Compras y Ventas, deberemos identificar una Base de conocimiento núcleo, la que estará conformada por objetos Genexus comunes a ambas aplicaciones como son: bancos, agencias, facturas, etc. ; Esta Base de Conocimiento núcleo se entregará a cada equipo de desarrollo y servirá como punto de partida para el desarrollo de cada aplicación a su cargo.las cuales se integrarán en una Base de Conocimiento corporativa.

Se definirán el número de transacciones y el tiempo estimado a emplear en su implementación en cada iteración y el responsable de llevarlas a cabo.

4.2.2.2. FASE DE DISEÑO.

Corresponde a la representación lógica del sistema de software, permite describir la aplicación antes de su implementación. Con la herramienta Genexus, haciendo uso de la ingeniería inversa, se genera automáticamente, solo existe un modelo de diseño para una Base de Conocimiento y no está relacionado con ningún administrador de bases de datos ni lenguaje de programación.

Esto significa que no existirá una base de datos física asociada a este modelo, así como tampoco programas de aplicación ejecutables. Estos últimos existirán solo a un nivel conceptual o lógico.

Son los objetos Genexus que el analista haya definido dentro de este modelo los que principalmente describen al sistema. En particular, de las transacciones especificadas Genexus obtiene el diseño lógico de la base de datos, y ellas, en conjunto con el resto de los objetos definidos, constituyen la descripción lógica de los programas.

Los programas serán el resultado de la codificación de los objetos Genexus en el lenguaje de programación elegido por el analista, sobre una base de datos física.

Pero esta implementación (base de datos y programas), es realizada por Genexus en las fases de modelo de prototipo y de producción que se

definen dentro de la base de conocimiento. Cada uno de estos otros modelos, contendrán una implementación del sistema de software.

El modelo de diseño se puede subdividir en cuatro sub- fases:

- Identificar los objetos del Prototipo: Determinar los objetos que deberán considerarse en el prototipo de cada iteración.

- Concordar el cronograma: Ponerse de acuerdo en cómo y cuándo se deberán ingresar a la Base de Conocimiento estos objetos.

- Crear Prototipo de diseño: Incrementar la Base de Conocimiento con los objetos identificados en la fase anterior y obtener el prototipo de diseño. Investigar, perfeccionar y consolidar con el prototipo de diseño combinado de iteraciones de la fase anterior.

- Revisión del prototipo de diseño: Comprobar la veracidad de los prototipos desarrollados. Esto se puede hacer a través de pruebas realizadas por el usuario final y el desarrollador, a continuación, utilice los registros de prueba y la retroalimentación del usuario para generar el documento de revisión de prototipos de diseño.

Los resultados de esta fase son un modelo de diseño externo de la Base de Datos y un prototipo de diseño que en conjunto representan el diseño de software que se pueden obtener en esta iteración, preparado para ser probado por los usuarios. Junto a esto, la lista de objetos se actualiza, eliminando los objetos que se han realizado y replantear las prioridades de los objetos restantes. El registro de riesgos también se actualiza por tener que considerar ahora el análisis del riesgo de un desarrollo mayor.

4.2.2.2.1. Creación de una Base de Conocimiento.

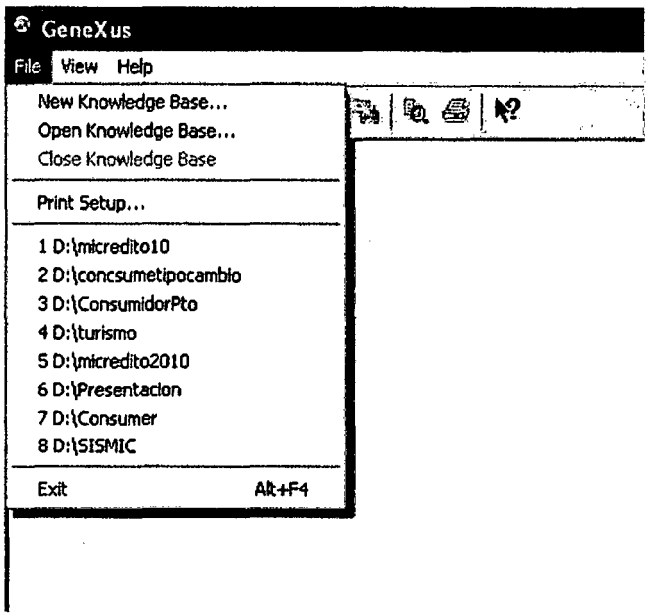
El primer paso para crear una aplicación con Genexus es crear una Base de Conocimiento, la cual contiene toda la información necesaria para generar una aplicación en múltiples plataformas.

Una base de Conocimiento está compuesta de varios Modelos:

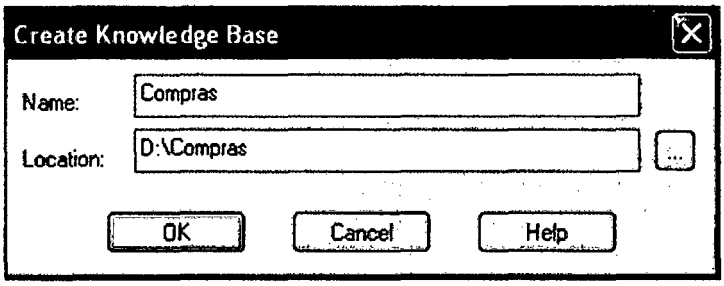
- **Modelo de Diseño**: Contiene todos los requerimientos de Datos de la aplicación. La información contenida en este Modelo es compartida por todos los otros Modelos en la base de Conocimiento. Siempre se comienza a diseñar la aplicación en el Modelo de Diseño.

- **Modelos de Prototipo** (generalmente uno o más): Contiene la información de diseño específica para uno o más ambientes de prototipo.
- **Modelos de Producción** (generalmente uno o más): Contiene la información de Diseño específica para uno o más ambientes de producción.

1. Ejecute Genexus. Luego de seleccionar **File**, se mostrará la pantalla siguiente:



2. Para crear una nueva base de conocimiento, en el menú **file,** seleccione y haga clic en **New Knowledge Base**
3. Digite el Nombre de la base de conocimiento a crear: Por ejemplo "Compras", luego presione clic en **OK** para continuar.

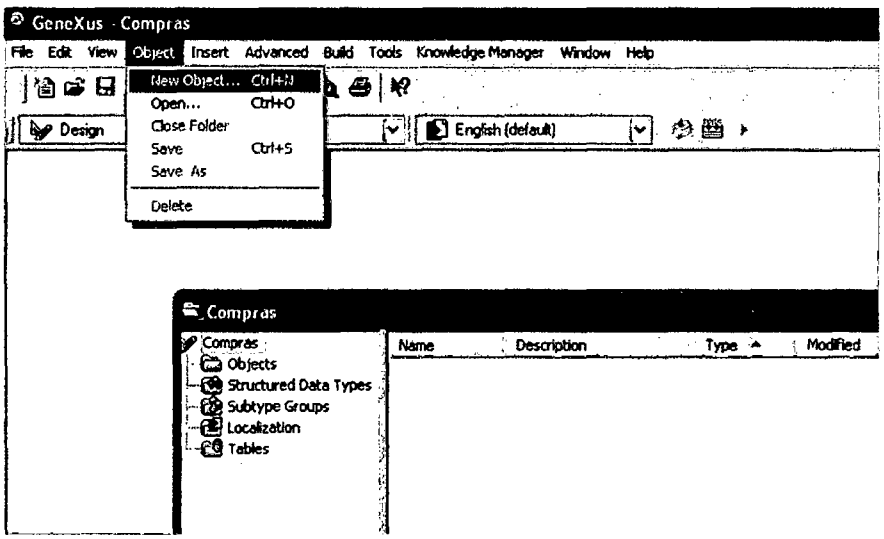


4.2.2.2.2. Creación de un Objeto Transacción.

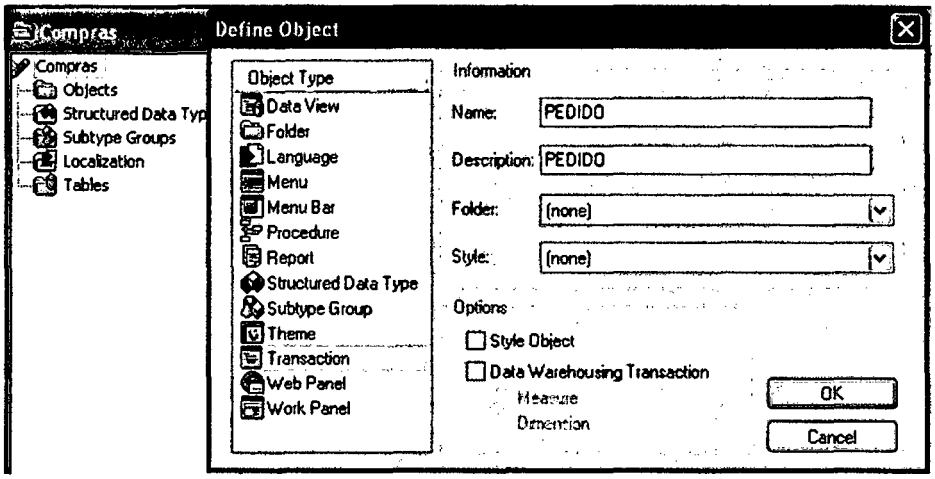
El Objeto Transacción: Representa a los Objetos de la realidad que maneja la aplicación. De las transacciones definidas se infiere el modelo de datos de la aplicación (3era forma normal). Genexus también usa el objeto Transacción para generar el programa de la aplicación que permitirá al usuario final, en forma interactiva, insertar, borrar, consultar y actualizar registros en la base de datos física. Las transacciones pueden ser creadas exclusivamente en el Modelo de Diseño.

Para crear una Transacción que representa un PEDIDO, seguiremos los siguientes pasos:

1. En el menú **Object**, seleccione **New Object.** Se mostrará la pantalla siguiente:



2. Seleccione el objeto que quiere crear: en este caso **Transacción**
3. Digite un nombre al Objeto que quiere crear: Por ejemplo PEDIDO.



4. Haga clic en **OK**

4.2.2.2.3. Descripción de la Estructura de la Transacción (structure):

Describimos la estructura PEDIDO definiendo que atributos integran la misma y cómo están relacionados.

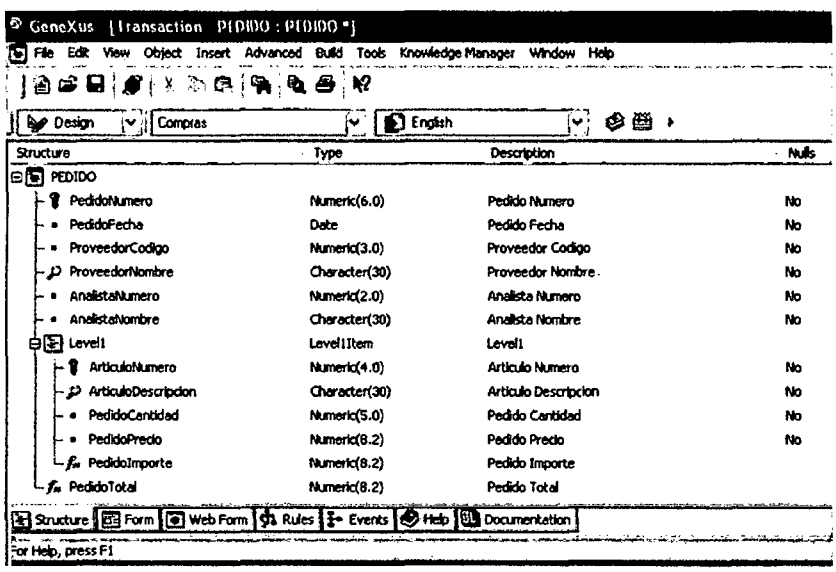
La estructura del objeto transacción es una descripción de los datos requeridos para conocer el objeto real que este representa. En la estructura, debemos declarar los atributos (campos) que forman la transacción (los datos con los que el usuario interactuará) y las relaciones entre ellos. En base a esta estructura, Genexus diseña y mantiene automáticamente la Base de Datos correspondiente (tablas, claves, índices, restricciones de integridad, etc.) en 3era forma normal.

Los elementos clave para definir la transacción son los siguientes:

- **Atributos:** Cada atributo es definido por su nombre tipo de dato y descripción.
- **Niveles:** Los atributos se agrupan en uno o más niveles, y estos niveles pueden ser anidados y paralelos (pueden haber múltiples niveles anidados). Por ejemplo: Las líneas de un PEDIDO representan un nivel anidado a nivel raíz. El nivel de las líneas del PEDIDO demuestra el hecho de que un PEDIDO puede tener muchas líneas, es decir, define una relación de una a muchas entre el PEDIDO y las líneas del PEDIDO.
- **Atributos de Clave Primaria (PK):** En cada nivel, uno o más atributos pueden ser elegidos como Clave Primaria del nivel.
  - La Clave Primaria es un identificador de cada instancia del nivel.

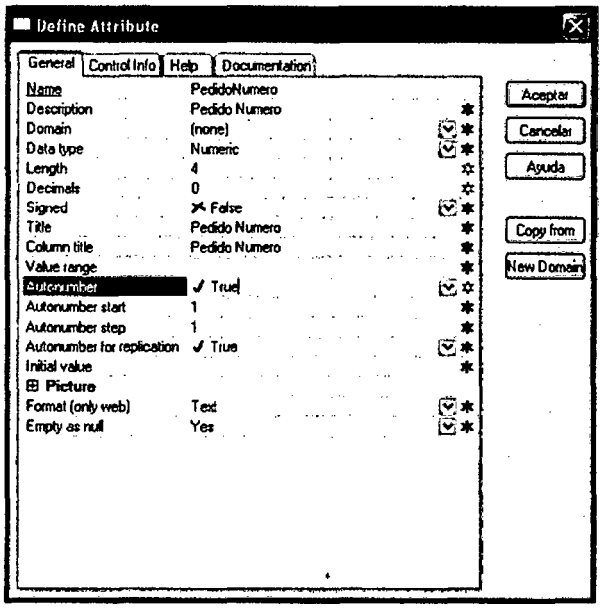
- Los valores de la Clave Primaria son únicos y una vez que se ingresan no pueden ser actualizados.
- Si no existe una Clave Primaria llamémosle "natural" para el objeto, debe crearse una "artificial"; por ejemplo, **ProveedorCodigo**.

1. Ingrese el nombre, tipo de datos y descripción de los atributos en la solapa **Structure** de la transacción PEDIDO. Use la tecla **TAB** para moverse entre en nombre, el tipo de dato y la descripción del atributo. Use la tecla **ENTER** para agregar un nuevo atributo. Use la **tecla Control + flecha a la derecha** para agregar un nuevo nivel a la estructura de datos y la **tecla control + la tecla flecha a la izquierda** para retornar a un nivel anterior. Luego de ingresar algunos de ellos tendremos la pantalla siguiente:



El primer atributo de cada nivel es definido por defecto como clave primaria de ese nivel, pero esto se puede cambiar haciendo clic derecho sobre el atributo y eligiendo la opción **Toggle Key** (Ctrl+K).

2. Haga clic derecho en el atributo **PedidoNumero** y seleccione **properties.**
3. Configure la propiedad **autonumber** de **PedidoNumero** en **True.** De este modo, los programas generados asignarán automáticamente un valor de **PedidoNumero** a cada instancia del PEDIDO.



4. Salve la nueva estructura de la transacción oprimiendo el botón con forma de diskette (**Save**) en la barra de herramientas estandart.

Universal Relationship Assumption (URA)

Un elemento clave de la Metodología Genexus es la hipótesis de que los atributos con el mismo nombre son el mismo atributo. Esto se llama Universal Relationship Assumption (URA), y según dicha hipótesis:

- Todo lo que es conceptualmente igual debe tener el mismo nombre.
- Los conceptos diferentes NO deben tener el mismo nombre.

Esto nos permitirá usar el mismo atributo en otros objetos Genexus (otras transacciones, procedimientos, reportes, etc.) simplemente haciendo referencia a su nombre. Genexus establece las relaciones (claves foráneas) entre las tablas del modelo de datos en base a los nombres de los atributos.

Nomenclatura Incremental Basada en Conocimiento de Genexus (GIK)

ARTech ha definido una nomenclatura de atributos flexible pero estándar -**GIK**- que es muy usada por la comunidad Genexus. En esta nomenclatura, el nombre de un atributo está formado por 4 componentes. Algunos de ellos son opcionales:

Objeto + Categoría [+ Calificador] [+ Complemento]

• **Objeto**: Es la entidad real descripta por el atributo. Generalmente (pero no siempre) es el nombre de la transacción en la cual un atributo ha sido

definido para que pueda ser almacenado en una de las tablas asociadas a esa transacción (esto significa que no debería inferirse en esa transacción específica). Ejemplos: Pedido, Proveedor, Articulo, Analista.

- **Categoría**: Es la categoría semántica del atributo. Define el rol que el atributo tendrá dentro del objeto y en el ambiente de la transacción. Ejemplos: Identificación, Fecha, Pedido, Descripción, etc.
- **El Calificador y el Complemento** son componentes opcionales. Ejemplos: Inicio, Fin, Mínimo, Máximo, etc.

4.2.2.2.4. Definición de campos calculados: Fórmulas.

Fórmulas: Son atributos que pueden inferirse del valor de otros atributos. Un atributo fórmula es definido de la misma manera que un atributo "normal", es decir, tiene un nombre, un tipo de datos y una descripción, además de una fórmula que define como se calcula.

- A menos que se especifique lo contrario, los atributos definidos como fórmulas no son almacenados en la base de datos (son **atributos virtuales**).
- Las fórmulas son **globales**; son válidas en toda la base de conocimiento y no solamente dentro de la transacción en la cual fueran definidas. Esto significa que la formula es calculada cada vez que el atributo es invocado desde una transacción o desde otros objetos Genexus (Reportes, Work Panels, etc.)
- **Variables Definidas por el Usuario**. No pueden participar en una fórmula porque son locales respecto a los objetos en donde han sido definidas y no tienen valor fuera de ellos.

Ahora definiremos los siguientes atributos fórmula:

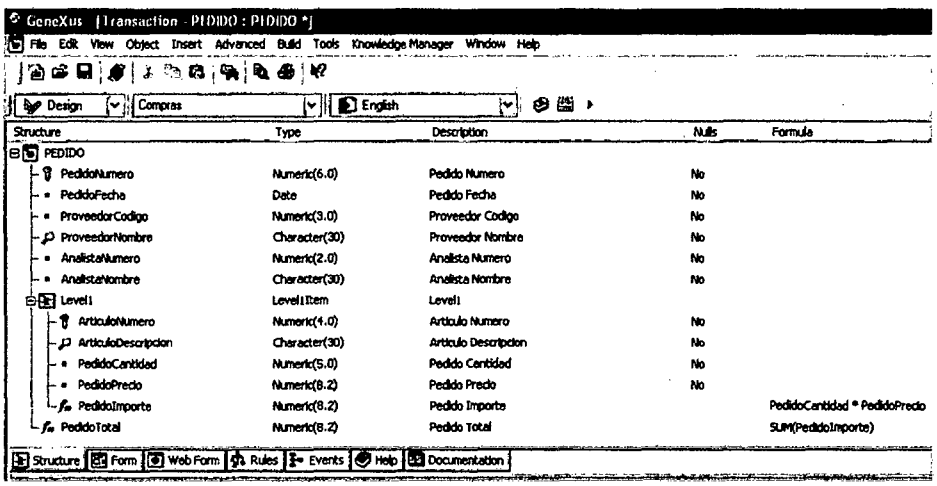
PedidoImporte = PedidoCantidad * PedidoPrecio

PedidoTotal= SUM(PedidoImporte)

1. Haga doble clic en el campo Formula del atributo PedidoImporte (a la derecha de la descripción del atributo).

2. Escriba lo siguiente: "PedidoCantidad * PedidoPrecio". CONSEJO: También se puede hacer clic derecho sobre el campo de la fórmula y seleccionar la opción **Editar Fórmula**... para abrir el Editor de Fórmula.
3. Repita los Pasos 1 y 2 para el resto de las fórmulas que aparecen en la lista al principio de esta sección.
4. Haga clic en **Save** para salvar las nuevas fórmulas.

Luego tendremos una pantalla como sigue:



4.2.2.2.5. Visualización del Modelo de datos Inferido por Genexus

Usted puede ver el Modelo de Datos inferido por Genexus e incluso modificarlo.

Generación de Modelo de Datos

Siempre que se haga clic en el botón Salvar, Genexus inferirá el modelo de datos óptimo (3era forma normal sin redundancias) que soporte a las entidades del usuario final representadas por los objetos de su transacción Genexus. En base a este modelo de datos, Genexus generará una base de datos física cuando usted defina una DBMS objetivo para un modelo de prototipo o producción.

Generación de Modelo de Datos Inteligente: La estructura de los objetos transacción determina las tablas e índices a ser creados:

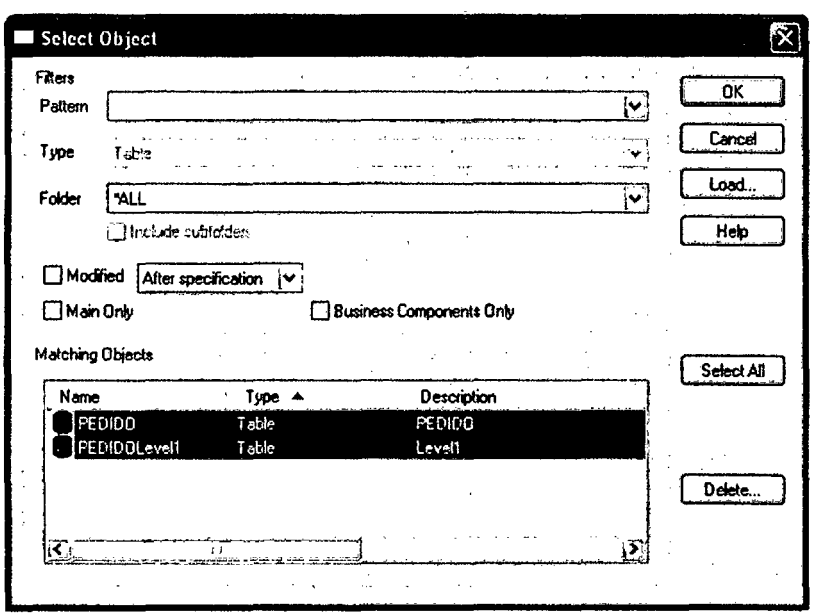
• Los nombres de las tablas e índices son asignados automáticamente por Genexus por el nombre de la transacción, pero pueden ser modificados si es necesario.

• Genexus infiere un modelo de datos en 3era forma normal, sin redundancias. No obstante, pueden definirse redundancias que serán automáticamente administradas por Genexus.

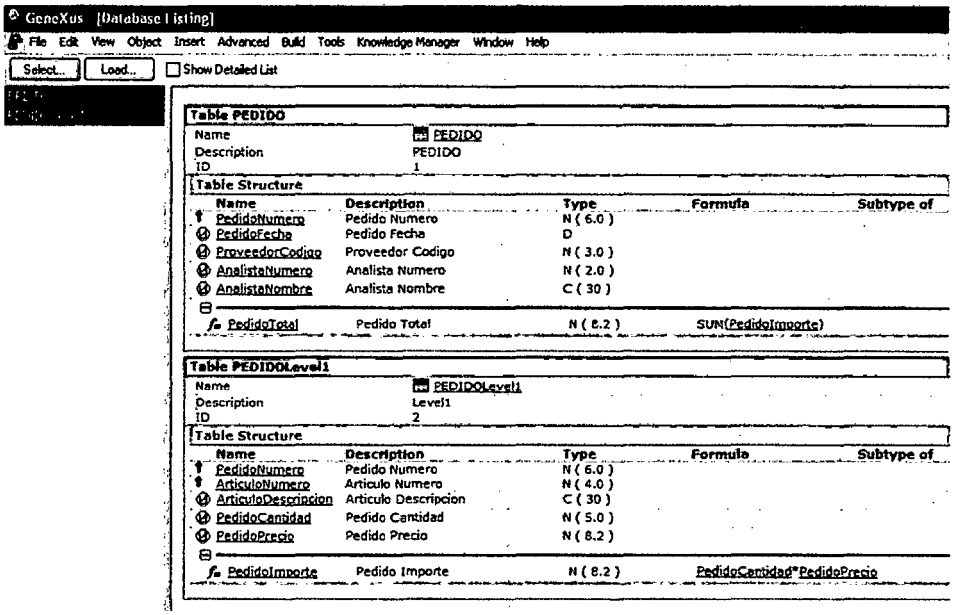
• La clave primaria de la tabla correspondiente a una transacción de nivel N se obtiene concatenando identificadores de los N-1 niveles anidados previos con el identificador de nivel N.

1. En el menú **Tools**, haga clic en **List Database**.
2. Quite la opción **Modified** si estuviera seleccionada.
3. En el **Dialog box Select Object** haga clic en **Select All** y oprima **OK**. Se generará un reporte con el Listado de la Base de Datos.

Se presentará la pantalla siguiente:



Al dar clic en Ok, se presentara la pantalla siguiente, mostrando el listado de la Base de Datos (del Modelo de Datos) generada por Genexus.



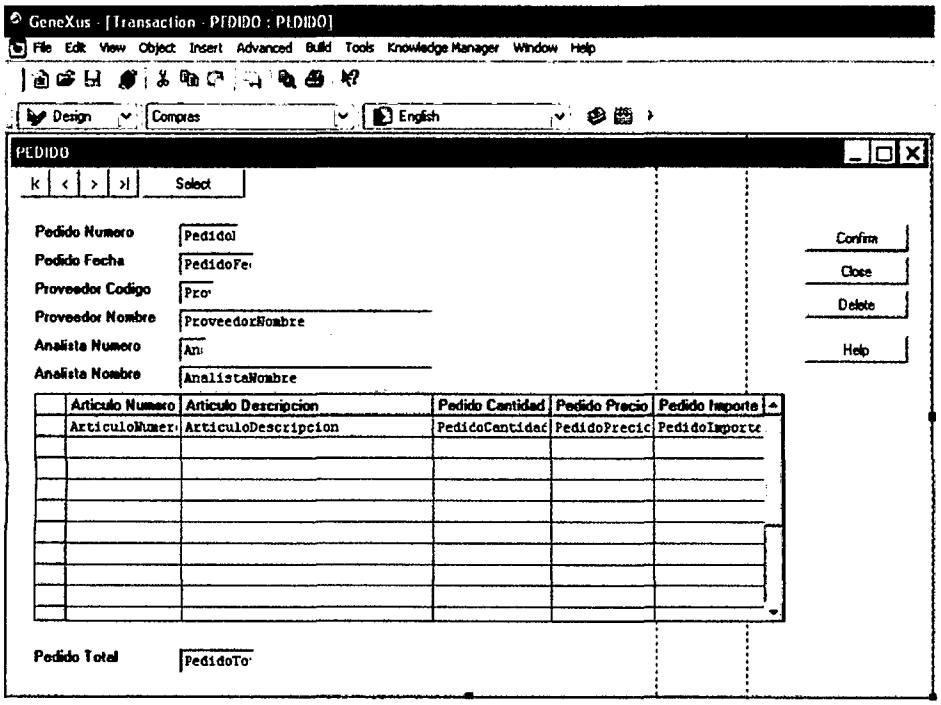
4.2.2.2.6. Visualización de los Formularios (Forms) del Objeto Transacción

Visualice los formularios GUI y Web predeterminados que han sido generados automáticamente por Genexus para el objeto transacción recientemente creado.

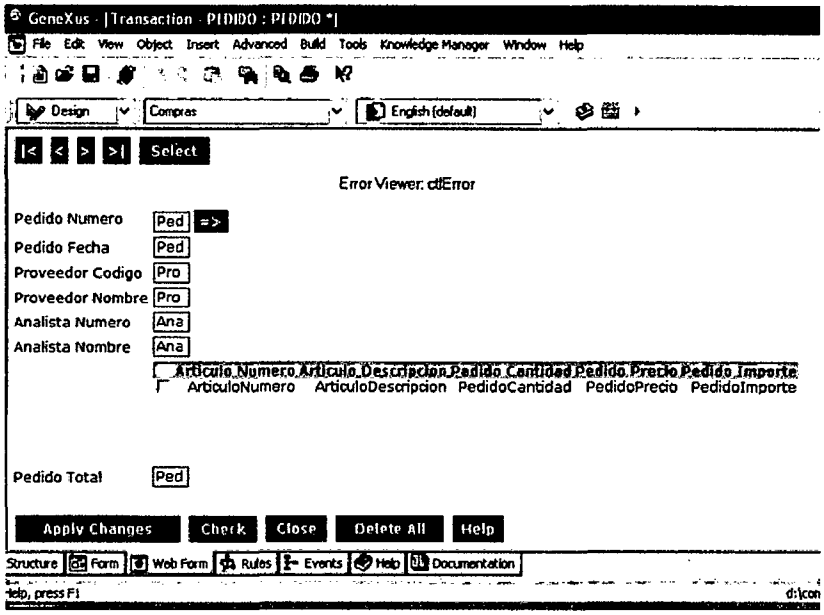
Formularios GUI y Web: Después de salvar un nuevo Objeto Transacción, Genexus crea automáticamente un Formulario (GUI) y un Formulario Web predeterminados para especificar como accederá el usuario final a los datos en las aplicaciones GUI y Web respectivamente. Ambos formularios pueden ser personalizados por el analista del negocio más adelante.

Para ver ambos formularios siga los pasos siguientes:

1. Seleccione la solapa **Form** de la transacción PEDIDO. Se mostrará el formulario GUI (predeterminado) de la Transacción PEDIDO en la pantalla siguiente:



2. Seleccione la solapa **Web Form** de la transacción PEDIDO. Se mostrará el formulario Web (predeterminado) de la Transacción PEDIDO en la pantalla siguiente:



"Error Viewer: ctlError" es el control predeterminado donde se despliegan mensajes de error. Puede ponerse en cualquier lugar del formulario y configurársele propiedades. Los mensajes posibles son los que se despliegan en **Msg** y **reglas de Error**, y los controles automáticos de Genexus (es decir, integridad referencial, errores tipo de datos, etc.).

Estos formularios habilitarán a los usuarios finales a ingresar nuevas facturas que se insertarán como nuevos registros en las tablas correspondientes. Los usuarios también podrán actualizar o eliminar los PEDIDOS existentes, siempre que tengan los derechos para hacerlo.

El analista de Genexus no necesita programar ninguna de estas acciones porque están implícitas en la lógica de la transacción. Genexus generará automáticamente el código nativo correspondiente en el lenguaje seleccionado.

Tener en cuenta que cuando se definen transacciones Genexus se está:

- Explícitamente: describiendo la interfase de usuario para la presentación y captura de datos.
- Implícitamente: diseñando el modelo de datos de la aplicación (tablas, índices, etc.).

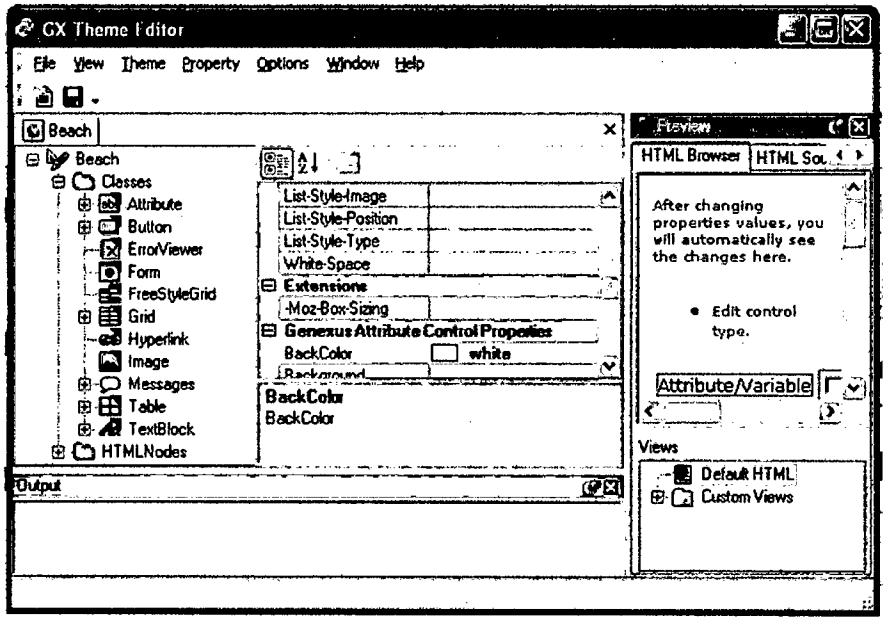
4.2.2.2.7. Creación de Formularios atrayentes: Temas

En esta sección se usted creará un nuevo Tema basado en una plantilla predefinida y lo configurará como el Tema de su aplicación.

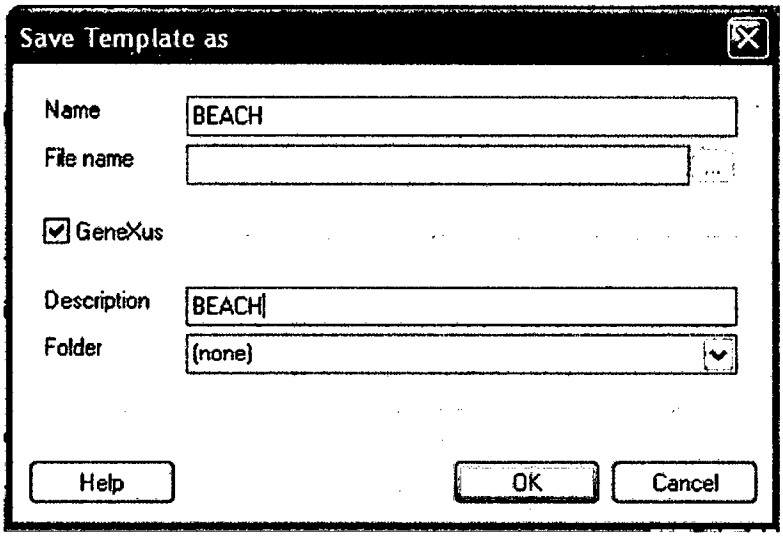
Objeto Tema: El objeto Genexus Tema mejora el desarrollo y mantenimiento de las aplicaciones Web separando las tareas del analista del negocio de las del diseñador Web.

- Se crean Formularios Web en base a un Objeto Tema predeterminado.
- Los Temas son definidos usando **Genexus Theme Editor**, una herramienta distribuida con Genexus.
- Usted puede crear nuevos Temas y asociarlos a la totalidad de su Base de Conocimiento o a objetos específicos dentro de ella.

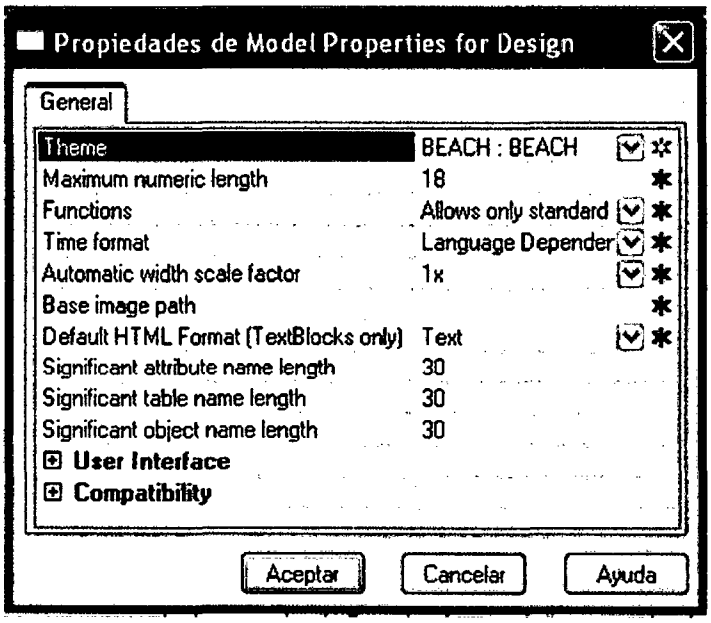
1. En el menú de **Tools** de Genexus, seleccione **GX Theme Editor**.
2. En el menú **File** del **GX Theme Editor** selecciones **Open / Template**.
3. Abra la plantilla **Beach.xml** del directorio C:\ProgramFiles\ARTech\Genexus\Genexus90Trial\KBTData\Templates. Se mostrará la pantalla siguiente:



4. Cambie el color de fondo de los Forms seleccionando **Classes / Form** y seleccionando el color **Light Yellow** (Amarillo Claro) en el tab Web.
5. En el menú **File** del **GX Theme Editor** seleccione **Save As**. Esto salvará el template como un Tema de su Base de Conocimiento. Se mostrará la pantalla siguiente:

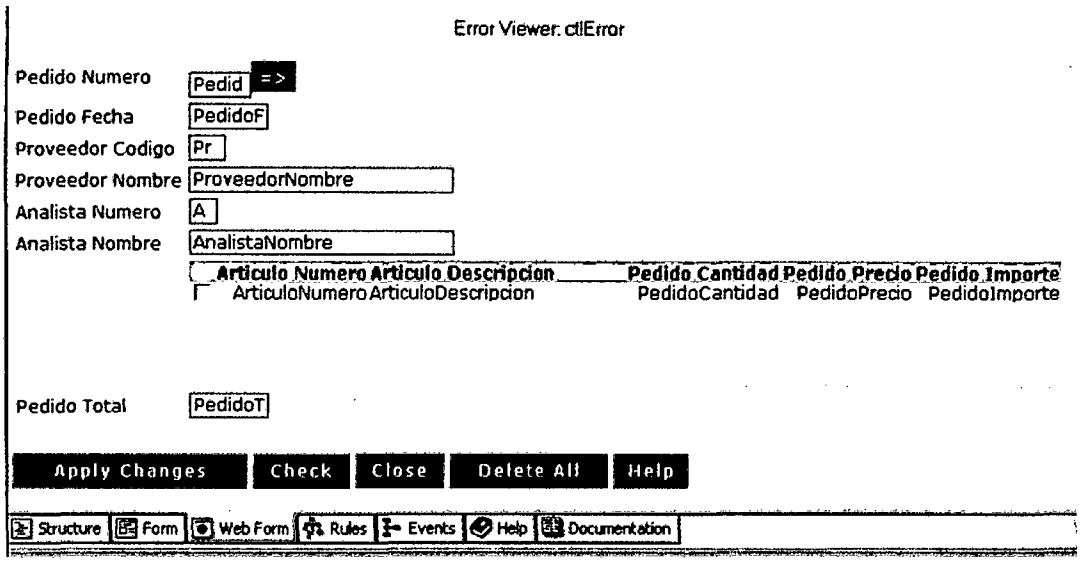


6. Cierre el **GX Theme Editor**.
7. En el menú **File**, seleccione **Edit Model** y después **Properties**.
8. Configure **Beach** como el nuevo Tema del Modelo. Se mostrará la pantalla siguiente:



9. Seleccione la solapa **Web Form** de la transacción PEDIDO. Se desplegará el Formulario Web previo, que ahora incluirá el nuevo Tema.

   Nota: debe cerrar y abrir la transacción para ver los cambios si esta estaba abierta de antemano.



4.2.2.2.8. Agregar Reglas del Negocio: Reglas

Agreguemos algo de lógica básica de negocios a la aplicación.

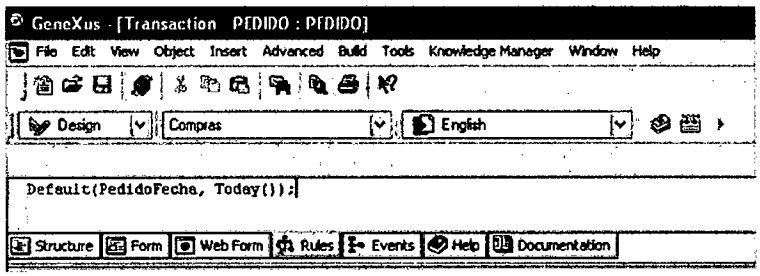
Reglas Genexus: Las Reglas Genexus son el medio para definir la lógica del negocio asociada a cada objeto. Son escritas en forma declarativa y Genexus decide de manera inteligente que regla aplicar y cuando aplicarla.

Las reglas juegan un rol muy importante en los objetos transacción porque permiten programar su comportamiento (por ejemplo: asignando valores predeterminados, definiendo controles de datos, etc.).

- Pueden involucrar atributos definidos en la estructura de la transacción así como variables y funciones.
- Las Reglas son programadas de manera declarativa, es decir, el orden en el cual se escriben no es necesariamente el orden en el cual se van a ejecutar. El orden de ejecución adecuado es automáticamente determinado por Genexus.
- Sólo son válidas en la transacción en la que han sido definidas. Por eso decimos que son locales.

Ahora agregaremos una regla simple que configura por defecto la Fecha de la Factura como la fecha del día actual:

1. Seleccione la solapa **Reglas** de la transacción PEDIDO
2. En el menú **Insertar**, haga clic en **Regla**.
3. Seleccione la regla **Default** (la primera de la lista) que asigna un valor predeterminado a un atributo o variable.
4. Complete la fórmula del modo siguiente: ***"Default(PedidoFecha, Today());"*** lo que indica que el valor predeterminado de la Fecha del PEDIDO será la fecha actual.
5. Haga clic en el botón de **salvar**.



4.2.2.2.9. Creación del Objeto Transacción PROVEEDOR

1. Cree el objeto Transacción PROVEEDOR siguiendo lo descrito en la sección 4.1.9.1.2. Creación de un Objeto Transacción y en la sección 4.1.9.1.3: Describir la Estructura de la Transacción.
2. Agregue los siguientes atributos a la Estructura PROVEEDOR:

3. Nótese que cuando comienza a escribir los atributos **ProveedorCodigo** y **ProveedorNombre**, Genexus le indica el nombre completo del atributo y su tipo y descripción. Esto sucede porque estos atributos ya están definidos en su base de datos.

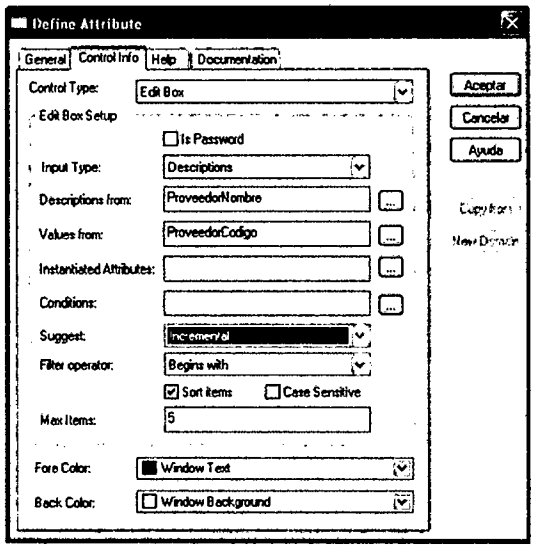
ATRIBUTO	TIPO	DESCRIPCION
ProveedorCodigo	----------	----------
ProveedorNombre	----------	----------
ProveedorDireccion	Carácter (50)	Cliente Dirección
ProveedorEmail	Carácter (50)	Cliente Email

4. Presione el botón derecho del mouse sobre el atributo **ProveedorCodigo** y seleccione **Propiedades**.
5. En la solapa General, configure la propiedad **Autonumber** de **ProveedorCodigo** como **True**. De esta forma, los programas generados asignarán automáticamente un valor **ProveedorCodigo** a cada nueva instancia de Cliente.
6. En la solapa **Control Info** configure las siguientes propiedades:
   - **Input Type** = Descriptions
   - **Descriptions from** = ProveedorNombre
   - **Suggest** = Incremental

Así, en vez de ingresar el código de un proveedor para identificarlo, podremos ingresar su nombre y la aplicación automáticamente inferirá su ID. La propiedad **suggest** le sugerirá todos los nombres de clientes que concuerden con el nombre ingresado por el usuario.

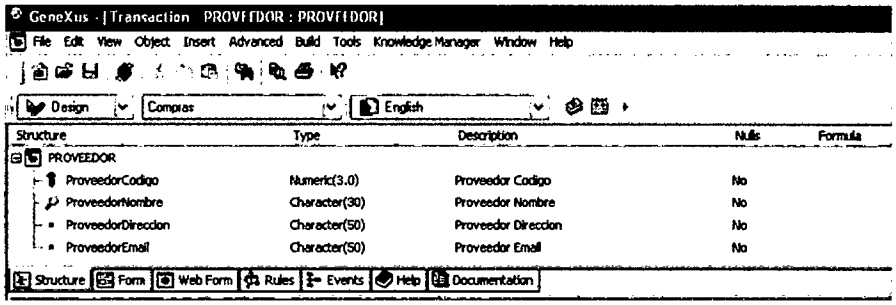
Estas propiedades son parte de la implementación de **AJAX** que hace Genexus automáticamente.

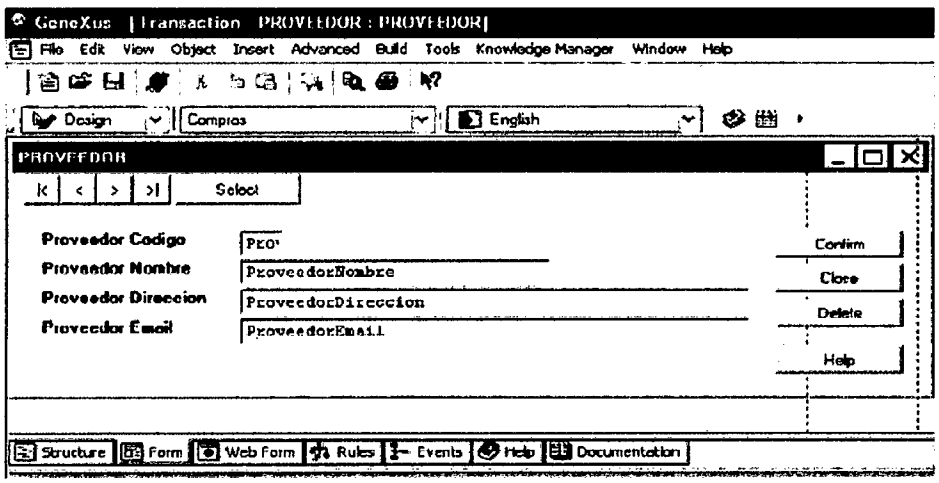
Ver figura siguiente:

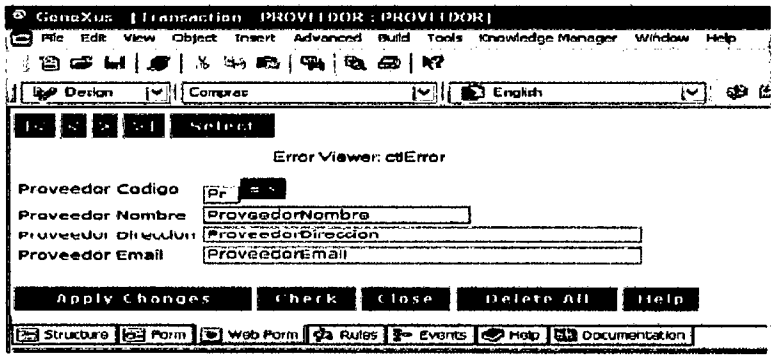


La Estructura, el Formulario (Windows) y Formulario Web de la Transacción PROVEEDOR se verán como se muestra en las siguientes figuras.

Estructura de la Transacción PROVEEDOR:

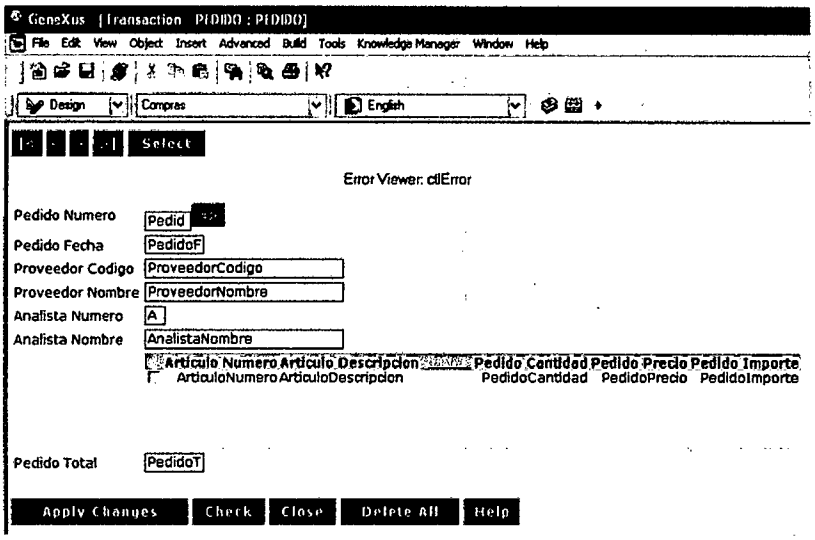






Nótese (abajo) que el Formulario Web del PROVEEDOR también ha cambiado, reflejando los cambios en las propiedades del atributo **ProveedorCodigo**: el atributo **ProveedorNombre** es ahora el la descripción del atributo **ProveedorCodigo**. Esto mejorara notablemente la usabilidad de la aplicación como veremos rápidamente.

Estructura Web de la transacción PEDIDO



4.2.2.2.10. Revisión de los Cambios efectuados al Modelo de Datos

Revise el nuevo modelo de datos inferido por Genexus (recuerde Salvar su base de conocimiento para que el modelo de datos sea automáticamente inferido). Para ello:

1. En el menú **Tools**, haga clic en **List Database**.
2. Quite la selección de **Modified** si estuviera seleccionada.

3. En el **Dialog box** Seleccionar **Objeto**, haga clic en **Seleccionar Todo** y después en **OK**. Se generará un informe con el Listado de la Base de Datos.

Comprobará que Genexus ha normalizado automáticamente su modelo de datos después de haber incluido la nueva transacción.

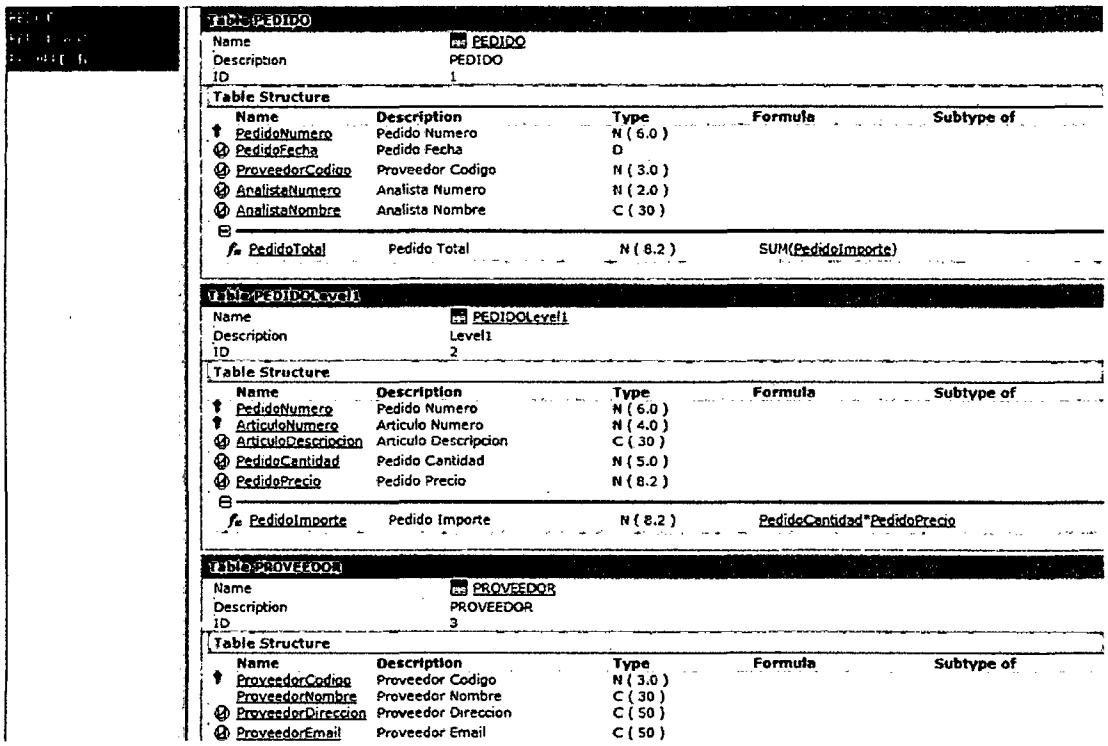


Table PEDIDO

Name	PEDIDO
Description	PEDIDO
ID	1

Table Structure

Name	Description	Type	Formula	Subtype of
PedidoNumero	Pedido Numero	N (6.0)		
PedidoFecha	Pedido Fecha	D		
ProveedorCodigo	Proveedor Codigo	N (3.0)		
AnalistaNumero	Analista Numero	N (2.0)		
AnalistaNombre	Analista Nombre	C (30)		
PedidoTotal	Pedido Total	N (8.2)	SUM(PedidoImporte)	

Table PEDIDOLevel1

Name	PEDIDOLevel1
Description	Level1
ID	2

Table Structure

Name	Description	Type	Formula	Subtype of
PedidoNumero	Pedido Numero	N (6.0)		
ArticuloNumero	Articulo Numero	N (4.0)		
ArticuloDescripcion	Articulo Descripcion	C (30)		
PedidoCantidad	Pedido Cantidad	N (5.0)		
PedidoPrecio	Pedido Precio	N (8.2)		
PedidoImporte	Pedido Importe	N (8.2)	PedidoCantidad*PedidoPrecio	

Table PROVEEDOR

Name	PROVEEDOR
Description	PROVEEDOR
ID	3

Table Structure

Name	Description	Type	Formula	Subtype of
ProveedorCodigo	Proveedor Codigo	N (3.0)		
ProveedorNombre	Proveedor Nombre	C (30)		
ProveedorDireccion	Proveedor Direccion	C (50)		
ProveedorEmail	Proveedor Email	C (50)		

Nótese que:

• Genexus agregó una nueva tabla: la tabla **Proveedor** (asociada a la transacción Proveedor).

• El atributo **ProveedorNombre** se ha eliminado de la tabla **Pedido** (este atributo está almacenado únicamente en la tabla **Proveedor**), dejando la base de datos normalizada.

• En la tabla **Proveedor**:

  • No pueden haber dos clientes con el mismo **ProveedorCodigo.**

  • Por cada **ProveedorCodigo** hay un único valor de **ProveedorNombre**, **ProveedorDirección** y **ProveedorEmail**.

• En la tabla **Pedido**:

- No pueden haber dos PEDIDOS con el mismo PedidoCodigo.
- El atributo **ProveedorCodigo** en la tabla **Pedido** es una clave foránea tomada de la tabla **Proveedor**. Por lo tanto, existe una relación de uno a muchos entre **Proveedores y Pedidos**:
  - Por cada **PedidoNumero** existe únicamente un **ProveedorCodigo**.
  - Cada **ProveedorCodigo** puede tener muchos Pedidos (**PedidoNumero**).

• Si usted selecciona la opción **Mostrar Lista Detallada**, Genexus le mostrará los índices de tabla que usa para mantener la integridad referencial de su base de datos y para acceder a las tablas de manera eficiente

4.2.2.3. FASE DE PROTOTIPO

4.2.2.3.1. Generación Automática de la Base de Datos.

Hasta el momento, hemos descrito los requerimientos de los datos de nuestra aplicación como una serie de objetos Genexus llamados transacciones. Genexus usa esta información para inferir el Modelo de Datos óptimo (3era forma normal) requerido para soportar sus transacciones.

Hemos seguido todos los pasos previos dentro del **Modelo de Diseño**. El Modelo de Diseño es un modelo independiente de la plataforma usada, que utilizamos para definir el modelo de datos de nuestra aplicación. Para generar una aplicación de trabajo (esquema de Base de Datos + Código Fuente + Programas Ejecutables), usaremos los **Modelos de Prototipo y Producción.**

4.2.2.3.2. Prototipando de la aplicación.

Modelos de Prototipo y Producción: Para generar y mantener una aplicación de trabajo en una plataforma de software específica, debemos definir un Modelo de Prototipo o Producción. Para esto, especificamos un DBMS, un lenguaje meta y algunos parámetros adicionales por cada Modelo de Prototipo o Producción. Genexus generará y mantendrá el esquema de la base de datos y todos los programas en la plataforma seleccionada. De esta manera, el

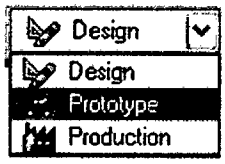
analista de Genexus no necesita tener un conocimiento profundo de la plataforma objetivo.

4.2.2.3.3. Prototipando de la Aplicación en .NET con SQL Server 2005 Express Edition

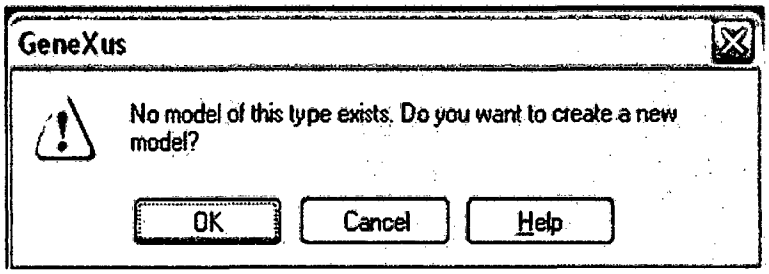
Antes de continuar, asegúrese de que tiene todo el software requerido para ejecutar la aplicación.

En este paso, generaremos un Modelo de Prototipo en Visual C# usando SQL Server 2005 Express Edition como nuestro DBMS. Alternativamente, usted puede usar cualquiera de los DBMS soportados por Genexus.

1. Nótese que Genexus no crea una nueva Base de Datos. Por lo tanto, antes de crear los Modelos de Prototipo o Producción deberá crear una nueva Base de Datos en su DBMS y asegurarse de que tiene los derechos necesarios para usarla. En este ejemplo usaremos la Base de Datos "Master" que se crea durante la instalación de SQL Server 2005 Express.
2. Seleccione el ambiente de **Prototipo** en la Barra de Herramientas del Modelo.

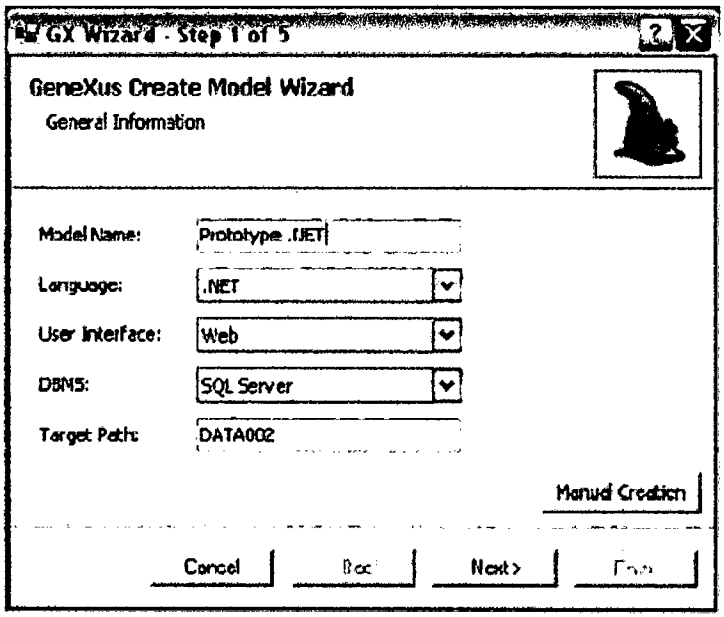


3. Se le indicará que cree un nuevo modelo de prototipo. Haga clic en **OK**.

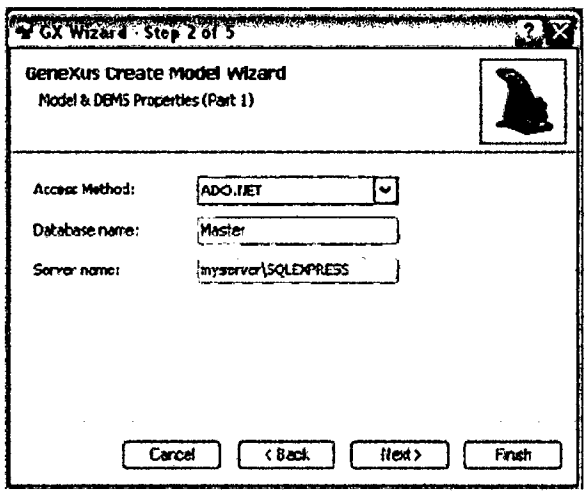


4. El Ayudante Para la Creación de Modelos Genexus lo guiará en la configuración de los parámetros del nuevo modelo. Configure lo siguiente:
   - Nombre del Modelo : Prototype .NET
   - Lenguaje : .NET
   - Interfase de Usuario : Web

- DBMS : SQLServer
- Ruta Objetivo : Deje el valor predeterminado

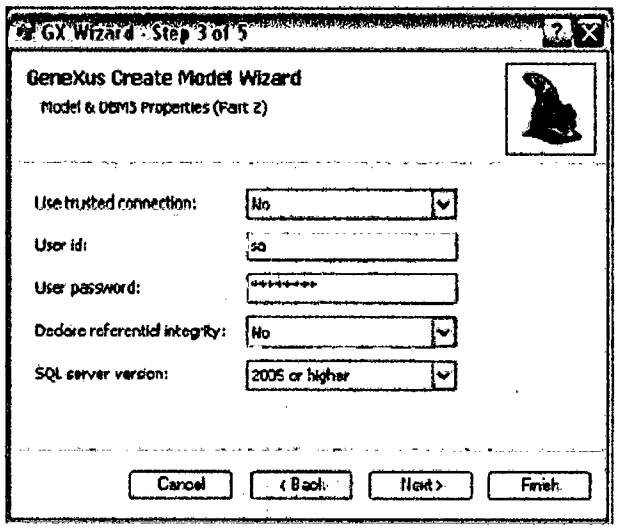


5. Haga clic en **Next**.
6. En el Paso 2 del Ayudante para la Creación de Modelos Genexus, configure lo siguiente:
   - Método de Acceso : ADO.NET
   - Nombre de la Base de Datos : Master 10
   - Nombre del Servidor : <Machine Name>\SQLEXPRESS

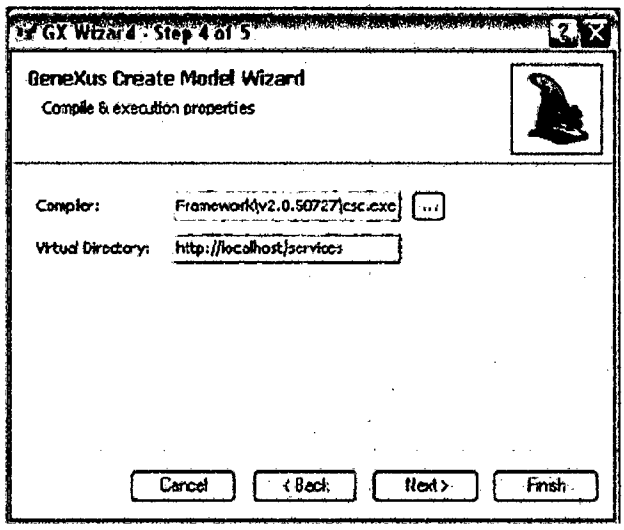


7. Haga clic en **Next**.
8. En el Paso 3 del Ayudante para la Creación de Modelos Genexus configure lo siguiente:

- Usar conexión segura : No
- ID de Usuario : sa
- Contraseña de Usuario : 18854554
- Declarar integridad referencial : No
- Versión de SQL server : 2005 o superior

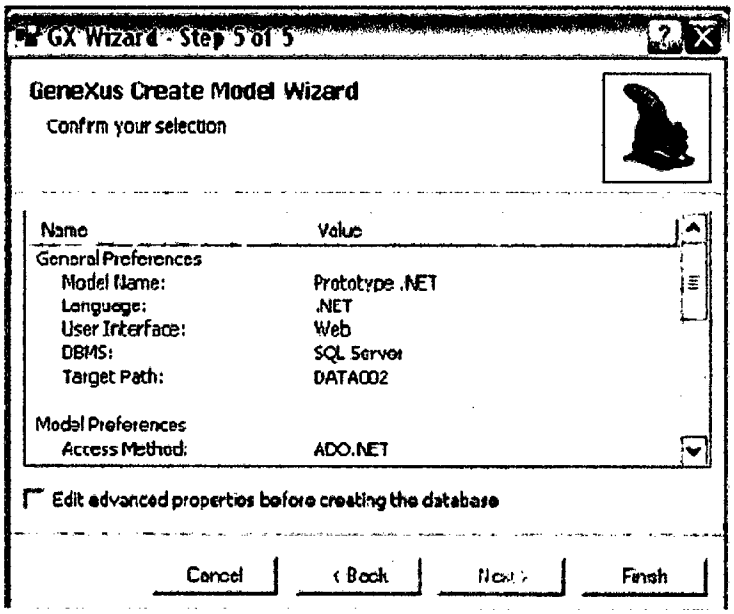


9. Haga clic en **Next.**
10. En el Paso 4, verifique que las rutas del Compilador y el Directorio Virtual de su máquina son las correctas.

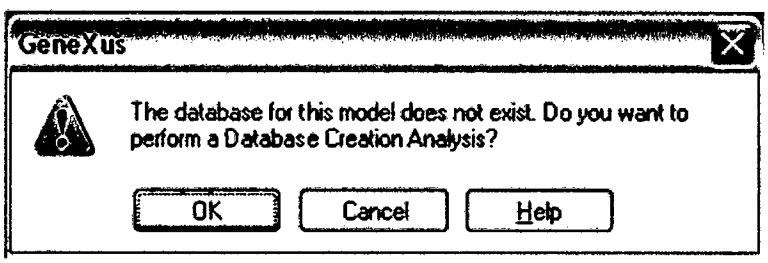


11. Haga clic en **Next**.
12. El paso 5 es un resumen de lo que usted ha seleccionado antes, de modo que deberá

verificar si sus opciones son correctas (puede retroceder si lo necesita) y, una vez verificado esto, hacer clic en **Finish**.



13. Haga clic en **OK**. Debido a que este es un nuevo modelo, se le indicará que cree el esquema de la base de datos física en base al modelo de datos inferido por Genexus.



4.2.2.3.4. Visualización del informe de creación de la base de Datos.

Es conveniente que lea el Informe de Creación de la Base de Datos antes de crear la base de datos del modelo de Prototipo.

Generación Automática de la Base de Datos

• Cuando usted está creando un nuevo modelo de Prototipo o Producción, Genexus genera los programas ejecutables requeridos para crear su base de datos en el DBMS seleccionado (en el Modelo de Prototipo o Producción) en base a el modelo de datos inferido (desde el Modelo de Diseño).

• Cuando usted está actualizando un modelo de Prototipo o Producción, Genexus genera los programas ejecutables requeridos para reorganizar la

base de datos del modelo; es decir, crea un nuevo esquema y convierte los datos del viejo esquema al nuevo.

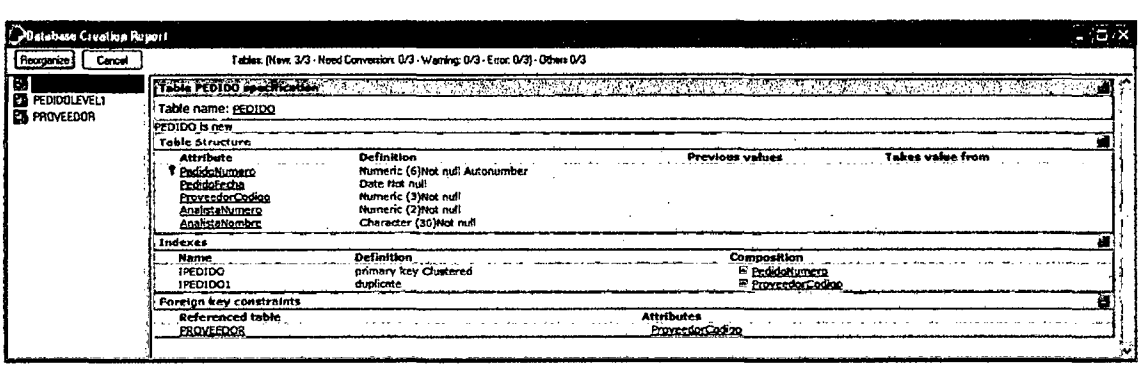
En estos casos se desplegará un Reporte de Creación de Base de Datos o un Reporte de Análisis de Impacto respectivamente, que le mostrarán lo que hará Genexus.

Reporte de Creación de Base de Datos: Es el reporte que describe el esquema de la base de datos que Genexus generará en al DBMS objetivo de su Modelo de Prototipo o Producción.

Contiene toda la información sobre el modelo de datos inferido y el esquema de base de datos propuesto para ser generado. La información sobre cada tabla está dividida en cuatro secciones:

- **Cabezal**: Contiene la nombre de la tabla, las acciones a realizar en él, advertencias y errores, Si el modelo de datos contiene errores, el botón de Reorganización estará deshabilitado.
- **Estructura de la Tabla**: Muestra los atributos de la tabla, sus relaciones y las acciones a realizar sobre ellos.
- **Índices:** Describe los índices de la tabla que Genexus usa para mantener la integridad referencial de su base de datos y para acceder a las tablas eficientemente.
- **Restricciones de la Clave Foránea**: Describe las restricciones de integridad de la tabla.

1. Lea el Reporte de Creación de la Base de Datos. Verá que puede seleccionar las especificaciones de la tabla que quiera visualizar.

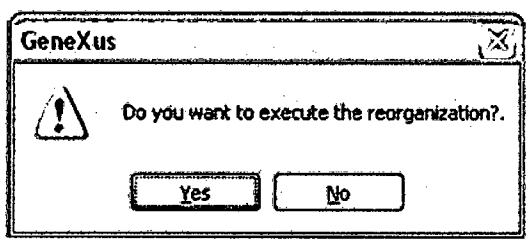


2. Si está de acuerdo con la base de datos propuesta, haga clic en **Reorganizar.**

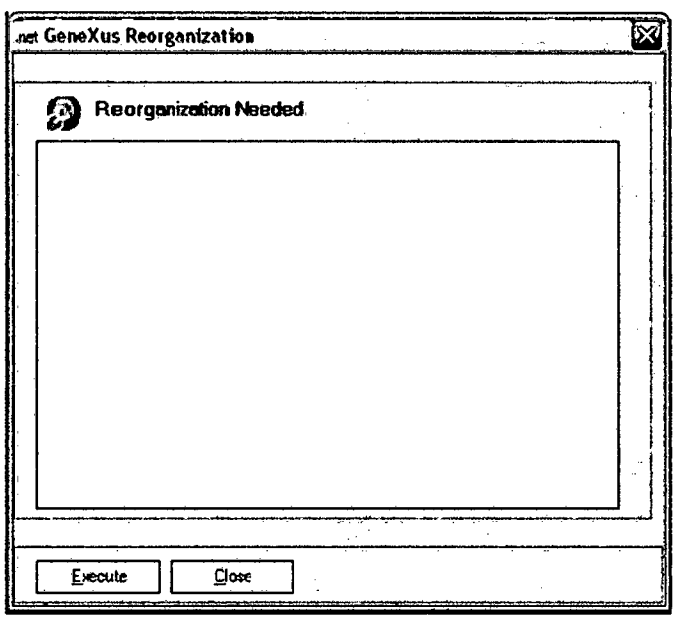
4.2.2.3.5. Creación de la Base de datos del modelo de prototipo. Reorganización de Genexus (Base de Datos): Cuando usted hace clic sobre el botón **Reorganizar** del Reporte de Creación de la Base de Datos, Genexus genera los programas de creación de la Base de Datos. La ventana de Reorganización de Genexus es el front end de estos programas y lo guiará a través del proceso de creación de la Base de Datos.

El mismo proceso se repetirá cuando ocurran cambios en su Modelo de Datos que requieran la reorganización de su Base de Datos. En este caso, usted generará y ejecutará programas de reorganización de la Base de Datos.

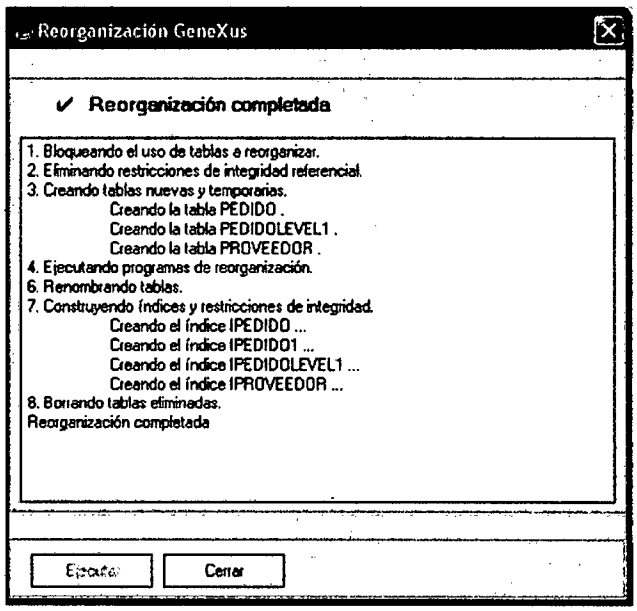
1. Cuando los programas de creación de la base de datos está listos para ejecutarse, se le indicará que los ejecute. Haga clic en **Yes.**



2. Los programas de creación de la base de datos le indican que debe reorganizar la base de datos de su modelo de Prototipo. Haga clic en **Execute.**



3. Cuando aparece el mensaje "**Reorganización terminada**" en la ventana de Reorganización de Genexus, haga clic en **Close**.



4.2.2.3.6. Generación Automática de Código.

Hasta el momento, hemos creado un nuevo esquema de base de datos que soporta el modelo de datos inferido por Genexus para su modelo de Prototipo. A continuación generaremos el código fuente para su aplicación en el lenguaje de su preferencia.

4.2.2.3.7. Especificación y Generación Automatica de Código Fuente: Comando Build.

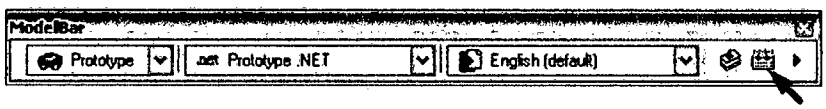
En este paso generaremos el código para el modelo de Prototipo. Para esto, usaremos el comando **Build.Building** es el proceso de creación del código fuente de las aplicaciones. Consta de dos pasos consecutivos:

1º. Especificación: Este proceso genera un archivo de especificación por cada objeto Genexus en el Modelo de Prototipo o Producción. El archivo de especificación describe el comportamiento del objeto Genexus y un lenguaje intermediario que es independiente del lenguaje objetivo de la aplicación. Estos archivos tienen extensión "**spc**". Por cada archivo de especificación, Genexus genera un Reporte de Especificación (que veremos en el próximo paso) que describe la lógica del objeto y muestra advertencias y errores. Una vez que se ha especificado un objeto (o un grupo de objetos), el analista puede indicar a Genexus que genere los programas de la aplicación.

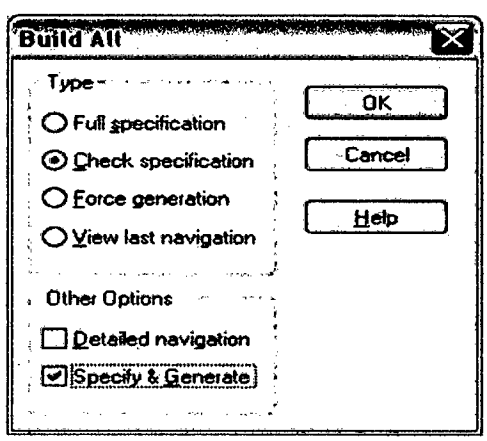
2º. Generación: Este proceso genera el código fuente para los programas de la aplicación en el lenguaje objetivo seleccionado para el **Modelo de Prototipo**

o **Producción**. Esto se hace en base a la información contenida en los archivos de especificación.

1. En el menú Build, seleccione Build All. También puede hacer clic en el acceso directo a **Build All** en la Barra de Herramientas del Modelo.



2. Seleccione el Tipo (Type) de especificación que usaremos: Check Specification.
3. En Other Options, seleccione: Specify & Generate.



4. Haga clic en OK.

4.2.2.3.8. Visualización del Reporte de Especificación:

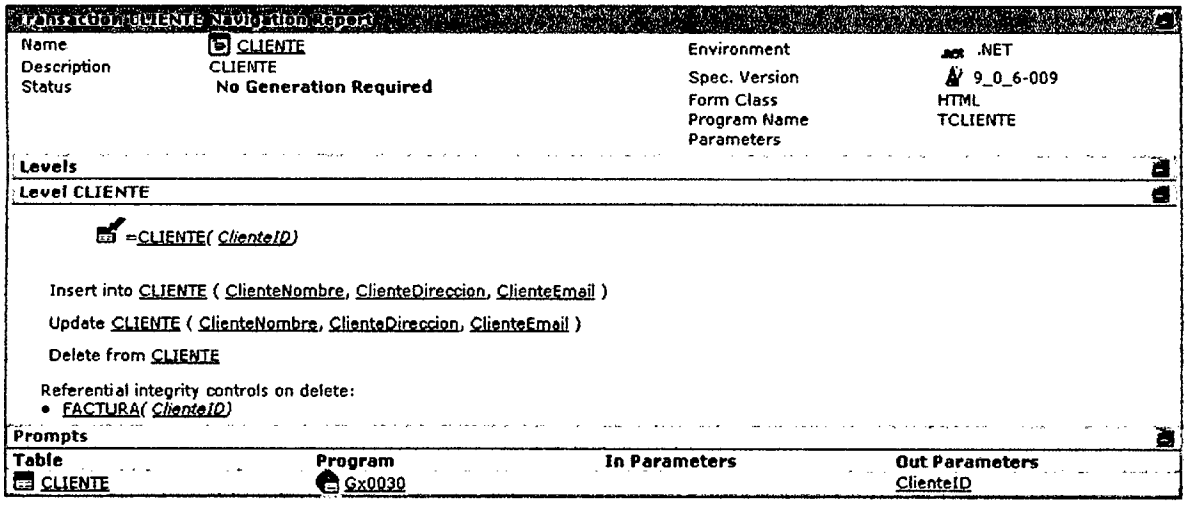
El Reporte de Especificación describe como se ejecutará el programa, a que tablas accederá (y como) y que operaciones realizará.

Los programas de la Transacción (programas asociados con los objetos transacción) permiten al usuario final Insertar, Actualizar y Eliminar registros de la base de datos. Aseguran la unicidad de la clave primaria y el manejo de la integridad referencial y el loqueo de registros. Todo esto sin que usted deba escribir ni una sola línea de código.

El mantenimiento de la integridad referencial implica evitar las inconsistencias de datos debidas a actualizaciones, por ejemplo:

- La eliminación de un registro padre con registros hijos.
- La inserción de un registro hijo sin un registro padre.

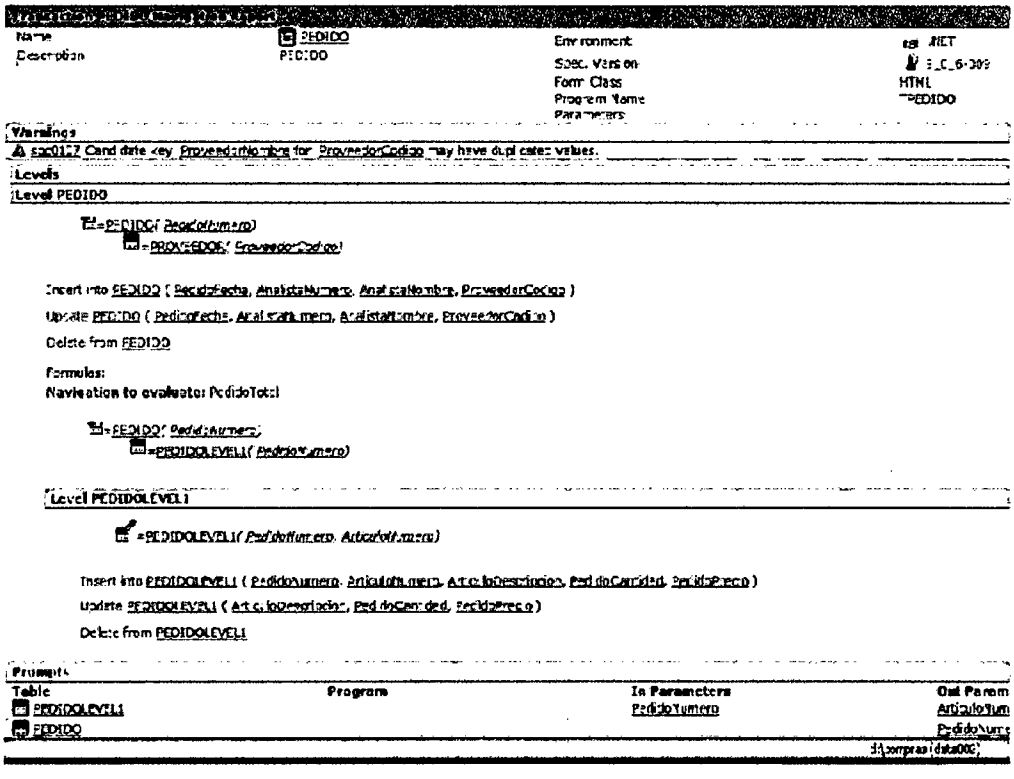
Genexus despliega un Reporte de Especificación compuesto por una serie de Reportes de Navegación, uno para cada programa que vaya a generar.



Reporte de Especificación de la Transación Pedido.

"Referential Integrity controls on delete" (Controles de integridad referencial al eliminar) significa que cuando usted elimina un Proveedor de la Transacción Proveedor, el programa verificará que no existan Pedidos para ése Proveedor. Para realizar esta búsqueda con eficiencia, se usa el índice foráneo **ProveedorCodigo** de la tabla Pedido.

Reporte de Especificación de la Transacción Pedido



Control	Type	Navigation
ClienteID	1	-CLIENTE(*ClienteID*) INTO ClienteNombre
		Where ClienteNombre. toupper() like @ClienteNombre. toupper() Order ClienteNombre

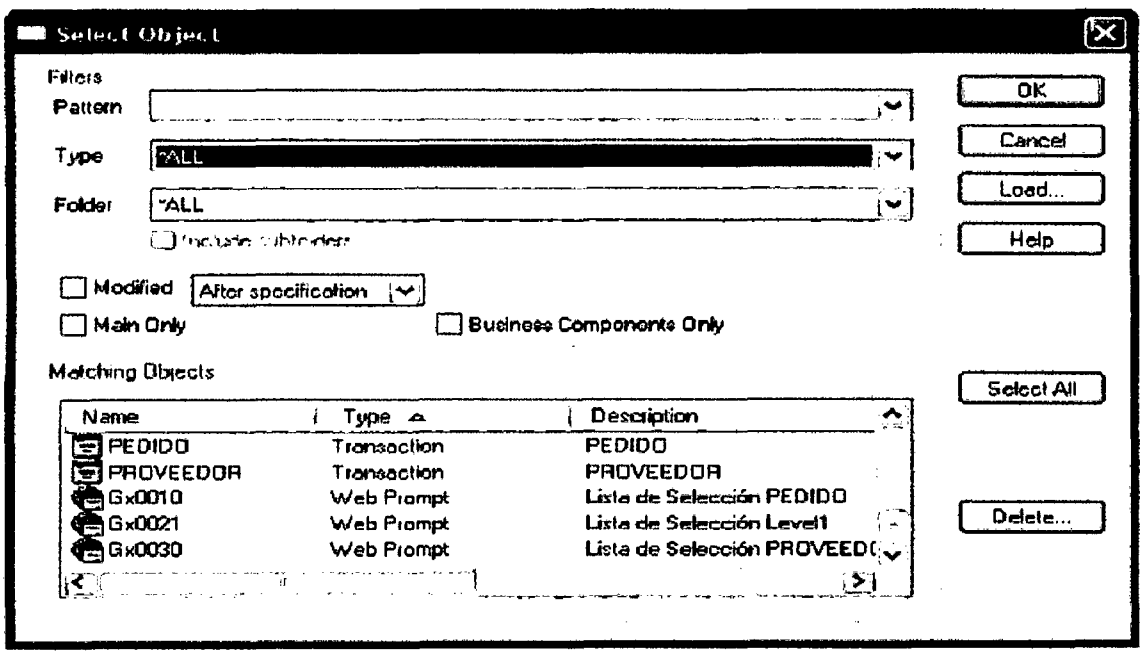
Integridad Referencial en la Transacción Pedido: La clave foránea **ProveedorCodigo** en el nivel **PEDIDO** significa que cuando usted inserta o actualiza un PEDIDO usando la Transacción **PEDIDO** se controlará automáticamente si el valor ingresado en la clave foránea **ProveedorCodigo** ya existe como clave primaria de un registro en la tabla **Proveedor**. Para realizar esta búsqueda con eficiencia, se usa el índice primario **ProveedorCodigo** de la tabla **Proveedor.**

El control de integridad referencial valida que un atributo ingresado (por ejemplo, **ProveedorCodigo** en una PEDIDO) es válido, pero no proporciona información sobre cuáles son los valores válidos. Para facilitar la búsqueda de los valores válidos, Genexus crea objetos Lista de Selección (prompts) que muestran el grupo completo de valores válidos para elegir los que correspondan.

Los objetos Lista de Selección agregados a la solución son los siguientes:

- Lista de Selección PEDIDO
- Lista de Selección PEDIDO1
- Lista de Selección PROVEEDOR

Lista para seleccionar objetos Genexus



4.2.2.3.9. Generación de Prototipo Funcional Definido con Genexus

A diferencia del modelo de Diseño que es creado automáticamente por Genexus, este modelo es creado en forma explícita por el analista. Al hacerlo, éste debe ingresar los datos de la plataforma o ambiente de implementación correspondiente (lenguaje de programación, DBMS, etc.) para que Genexus pueda implementar el software para este modelo.

Los objetos Genexus definidos en el modelo de Diseño se copian al nuevo modelo y es para éste modelo que Genexus genera código en forma automática de acuerdo a su plataforma.

Por cada base de conocimiento, habrá uno y solo un modelo de Diseño, cuya característica fundamental es que representa al sistema conceptualmente.

Los otros dos tipos de modelos se parecen mucho entre sí, dado que todo modelo de Prototipo o Producción tiene asociada una plataforma o ambiente que le permite implementar físicamente el sistema, siendo ésta la característica fundamental que diferencia a estos modelos del de Diseño. La principal diferencia entre ellos es conceptual (no funcional) y radica en el uso que se hace de los mismos:

- Un modelo de Prototipo se utiliza en la etapa de desarrollo, justamente para prototipar la aplicación–de allí su nombre- probando lo modelado, haciendo modificaciones, volviendo a probar, con la participación directa de los usuarios.

Generación de Prototipo Funcional Definido con Genexus

Hemos diseñado nuestra aplicación como un grupo de objetos transacción que mapean los objetos de la vida real. Genexus infirió automáticamente el mejor modelo de datos requerido para suportar nuestros requerimientos de datos. Después creamos un nuevo esquema de base de datos para una aplicación de prototipo. Finalmente especificamos y generamos el código fuente de los programas de nuestra aplicación de prototipo Estamos listos para probar nuestra aplicación en el ambiente de prototipo.

Prototipo de la Aplicación con Genexus

Se puede definir tantos Modelos de Prototipo y Producción como se quiera. Aún cuando ambos Modelos son idénticos con respecto a las funcionalidades soportadas, se recomienda especialmente que se defina por lo menos un Modelo de Prototipo por cada Modelo de Producción.

Porque es necesario usar prototipos:

El proceso de diseño está expuesto a todos los inconvenientes de la comunicación humana:

- El usuario olvida ciertos detalles.
- El analista no nota algunos puntos.
- El usuario transmite algunos enfoques erróneos.
- El analista interpreta mal algunas explicaciones del usuario.

Los prototipos reducen el riesgo de fallas en la implementación de varias maneras:

- Sin prototipo, los problemas de diseño solo se detectarán durante las pruebas finales del sistema. El costo de resolverlos en esta etapa será muy alto.
- La realidad cambia, de modo que no es razonable pensar que las especificaciones de su sistema puedan permanecer incambiadas durante la implementación del sistema.
- Mantener incambiadas las especificaciones durante la etapa de diseño seguramente conducirá a una solución insatisfactoria.

El prototipado cierra la brecha entre el diseño y las etapas de implementación de un sistema de software.

Un Prototipo Genexus es una aplicación "lista para trabajar" que es funcionalmente equivalente a la aplicación final de producción. El prototipo está ideado para correr en ambientes PC, pero puede ejecutarse en cualquier plataforma seleccionada. Genexus es capaz de generar código para los siguientes lenguajes: C# (para .NET Framework y .NET Compact Framework), Java, C/SQL, Cobol para iSeries, RPG para iSeries, Visual Basic (standalone y C/S), Embedded Visual Basic, y Visual Fox Pro (standalone y C/S). El prototipo le permite probar la funcionalidad de una aplicación antes de ponerla en producción. El usuario final puede fácilmente probar pantallas, reportes, fórmulas, reglas del negocio, estructuras de datos, etc.

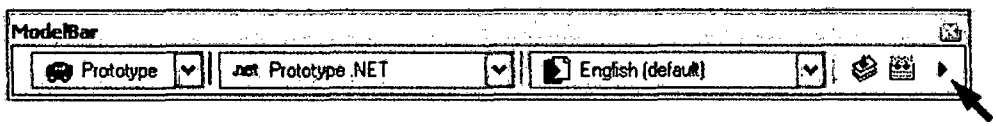
Trabajar con un Prototipo consiste en lo siguiente:

- Administrar la base de datos física asociada con el Modelo de Prototipo.
- Ejecutar la aplicación del Modelo de Prototipo con fines de evaluación.

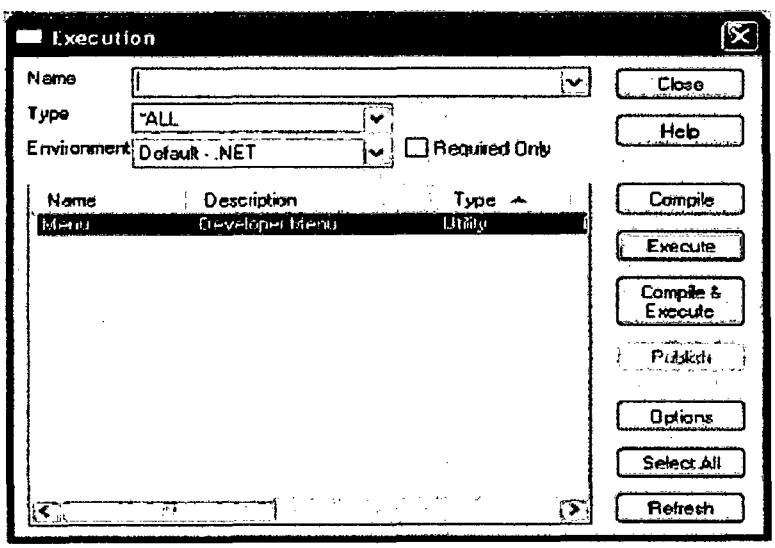
4.2.2.3.10. Ejecución de la Aplicación

1. En el menú **Build**, haga clic en **Run**. También puede hacer clic en el acceso directo a **Run** en la Barra de Herramientas del Modelo (último botón a la derecha), o simplemente presionar **F5.**

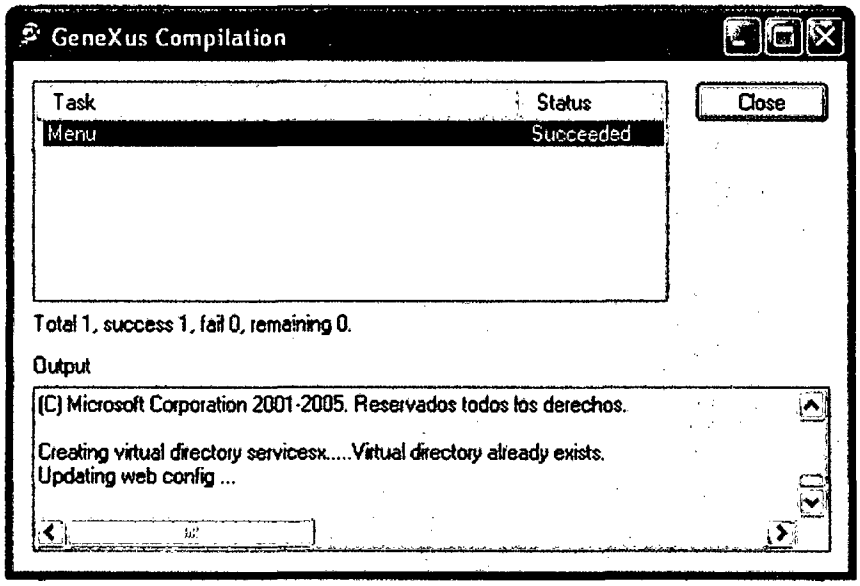
Barra de Herramientas del Modelo



2. Haga clic en Compile en el **Execution dialog box** que se desplegará para compilar la aplicación.



3. Aparecerá una ventana de Compilación Genexus. Cuando el Estado (**Status**) de la Tarea (**Task**) cambie a **Succeeded**, presione **Close.**

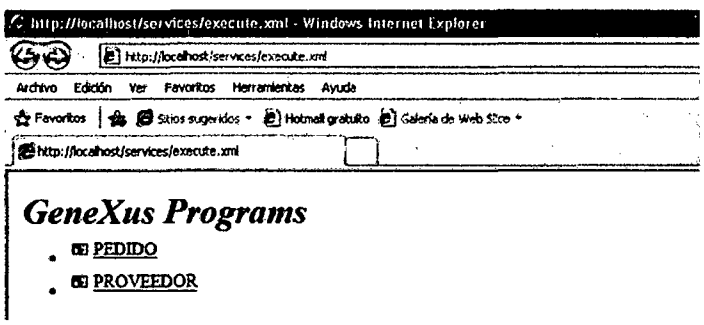


4. Haga clic en **Execute** en el **Execution dialog box** para ejecutar la aplicación

4.2.2.3.11. Prueba de la Aplicación.

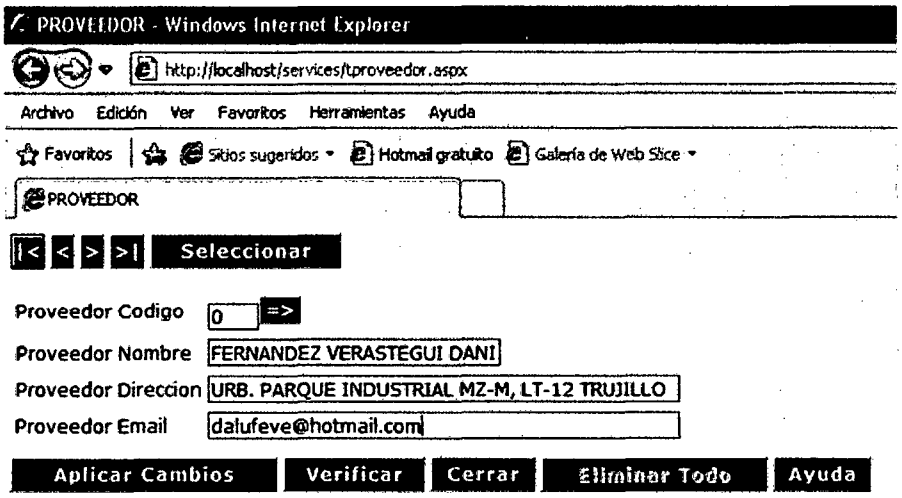
Cuando ejecute la aplicación compilada desde el modelo de prototipo, se ejecutará un prototipo completamente funcional.

1. El Menú del Desarrollador es un archivo XML que incluye a todos sus objetos ejecutables. Es un menú auxiliar para prototipar su aplicación. Haga clic en la opción **Cliente** (Si está usando el Internet Explorer abra una nueva ventana del mismo haciendo clic derecho sobre la opción **Cliente** y seleccionando la opción Abrir en nueva ventana).



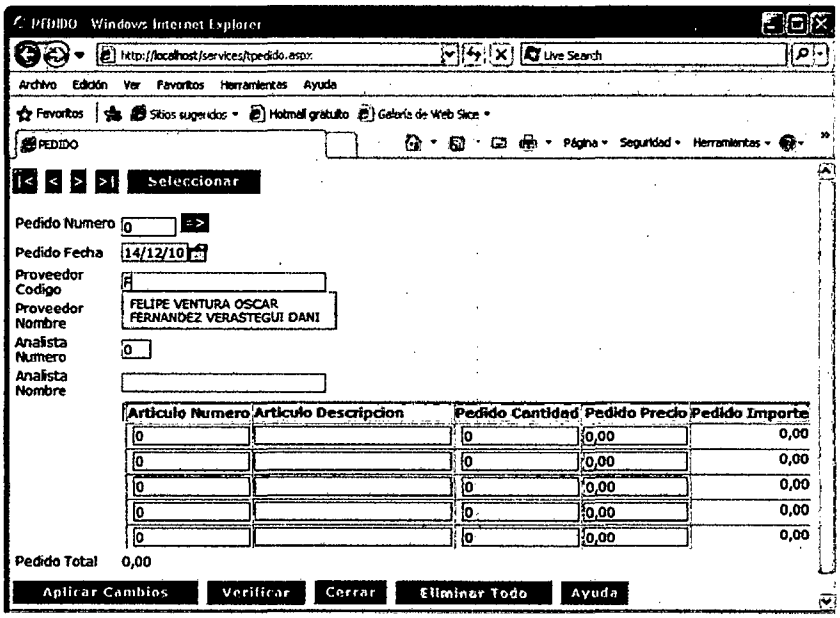
Menú del Desarrollador

2. Ingrese algunas instancias del cliente para usarlas en futuros Pedidos.



Instancia de la Transacción Proveedor.

3. Cuando haya finalizado haga clic en el botón Cerrar.
4. Haga clic en la opción Pedido (Si está usando el Internet Explorer abra una nueva ventana del mismo haciendo clic derecho sobre la opción Factura y seleccionando la opción Abrir en nueva ventana).
5. Ingrese algunos datos en el Pedido. Mientras hace esto, fíjese en las siguientes cosas:
   - La fecha del Pedido ya ha sido configurada como la fecha actual gracias a la regla Predeterminada definida anteriormente.
   - No necesita ingresar o seleccionar un Proveedor por su Código, simplemente escriba las primeras letras de su nombre y la aplicación generada le mostrará los nombres que comienzan con esas letras. Esto es gracias a la generación de AJAX. Recuerde que previamente definimos que el atributo ProveedorCodigo sería descrito por el atributo ProveedorNombre y le pedimos a Genexus que la aplicación nos sugiriera valores.
   - Las fórmulas se calculan automáticamente cuando usted hace clic en Aplicar Cambios por primera vez.
   - Debe hacer clic en el botón Aplicar Cambios dos veces para confirmar en ingreso de sus datos.



Instancia de la transacción Pedido

6. Cuando haya finalizado, haga clic en el botón **Cerrar.**

4.2.2.3.12. Desarrollo Incremental y mantenimiento de la Aplicación.

Hasta el momento hemos creado una aplicación de trabajo en base a algunos objetos transacción y reglas del negocio. Ahora veremos como mantener una aplicación Genexus simplemente editando los objetos Genexus existentes y/o agregando nuevos y luego actualizando su base de datos y regenerando los programas de la aplicación en forma automática.

Desarrollo Incremental con Genexus

A medida que la realidad cambia, los requerimientos del sistema evolucionan y esto se traduce en cambios en los objetos Genexus y/o nuevos objetos. En base a los nuevos objetos, Genexus actualizará su base de datos automáticamente (creando un nuevo esquema de base de datos y migrando los datos del viejo esquema al nuevo) y regenerará los programas de la aplicación que deban ser modificados.

Decimos que Genexus es incremental porque se basa en la presunción de que el desarrollo del sistema es un proceso iterativo que pasa por sucesivas aproximaciones. El desarrollo incremental es posible porque Genexus pude mantener el esquema de su base de datos y los programas de su aplicación automáticamente.

4.2.2.3.12.1. Inclusión de Nuevos objetos en el Modelo: Objeto Transacción Artículo.

Habrá notado que en la prueba de la aplicación, debía ingresar la Identificación, Descripción y Precio de Artículo en cada línea del Pedido. Esto no es lo que esperaría de un sistema de software de compras. Y ni siquiera de un sistema simple como esta aplicación de ejemplo.

Debemos contar con la posibilidad de agregar, actualizar y eliminar Artículos de nuestro sistema de software.

Para hacerlo, agregaremos la transacción Artículo en nuestra Base de Conocimiento:

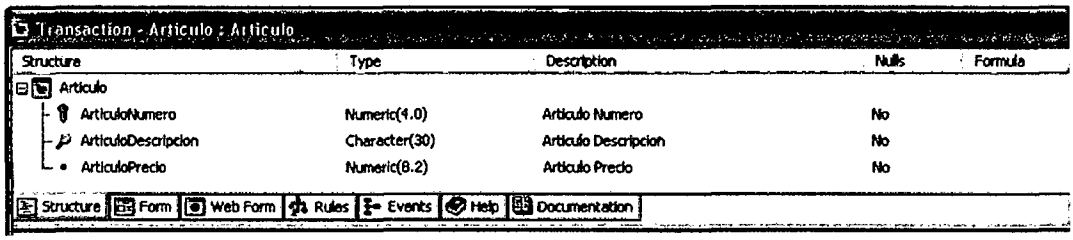
Seleccione Diseño en el menú desplegable de la **Model toolbar** para volver al Modelo de Diseño. Recuerde que todo el modelado de datos se hará únicamente en el Modelo de Diseño (la opción de crear nuevas transacciones está deshabilitada en todos los modelos de prototipo y producción).

Cree la transacción Artículo siguiendo lo descrito en la sección 4.1.9.1.2.: Creación de un Objeto Transacción y en la sección 4.1.9.1.3.: Descripción de la Estructura de la Transacción . Inserte los siguientes atributos en la Estructura de la Transacción Producto:

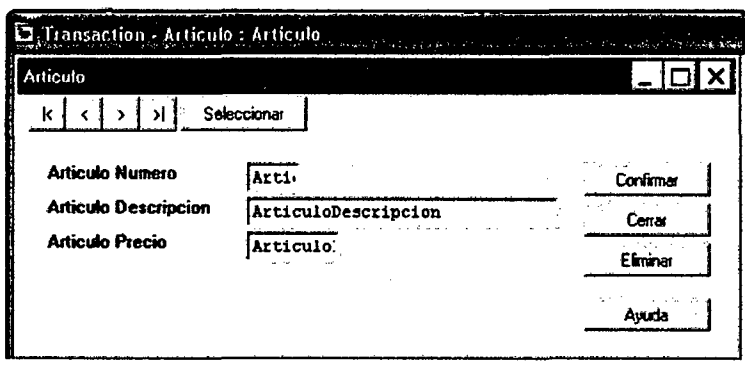
ATRIBUTO	TIPO	DESCRIPCION
ArticuloNumero	.....	.....
ArticuloDescripcion	.....	.....
ArticuloPrecio	.....	.....

Notará que tan pronto como empiece a escribir estos nombres de atributo, Genexus le indicará el nombre completo. Esto sucede porque estos atributos ya están definidos en su Base de Conocimiento.

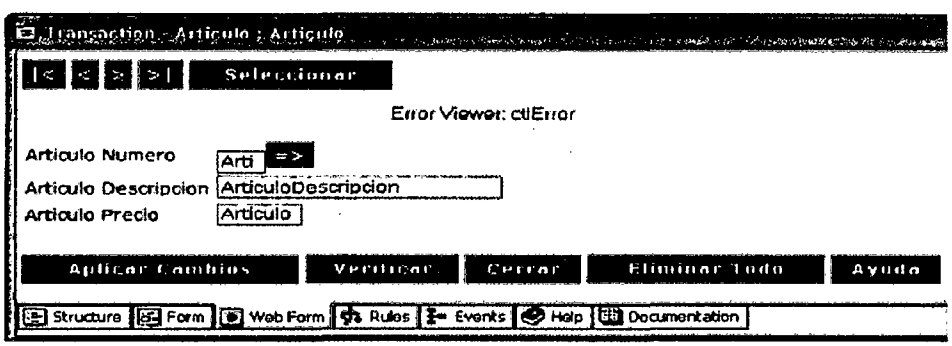
La estructura (Structure) de la transacción Articulo, su Formulario (Windows), y su Formulario Web se verán como sigue.



Estructura de la transacción Articulo



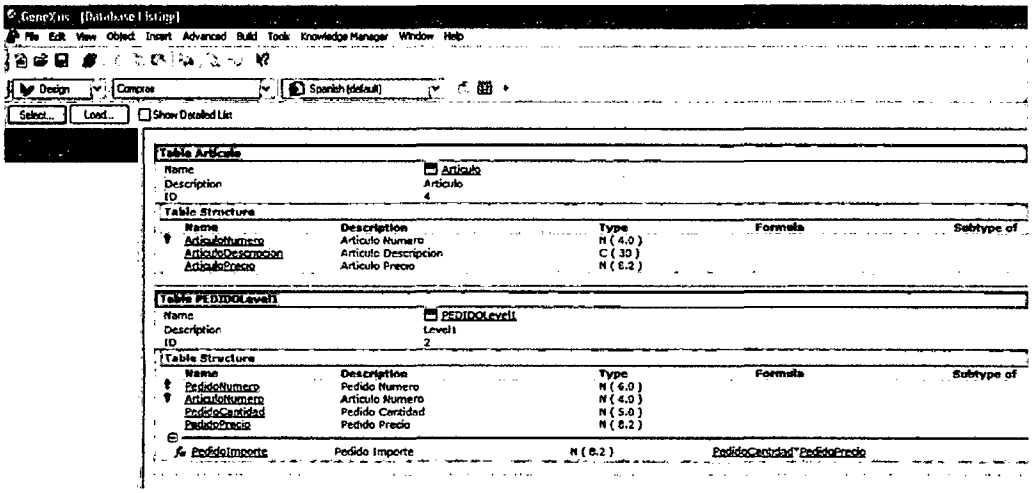
Formulario Windows de la transacción Articulo.



Formulario Web de la transacción Articulo

4.2.2.3.12.2. Revisión de los cambios efectuados en el Modelo de Datos.

Cuando usted salve el objeto Transacción Articulo, Genexus normalizará el Modelo de Datos nuevamente. La revisión del Modelo de Datos (opción de menú Tools / List Database) revela que Genexus ha normalizado automáticamente el Modelo de Datos moviendo los atributos ArticuloDescripcion y ArticuloPrecio desde la tabla Pedido1 (Detalle de Pedido) a la nueva tabla Articulo.



Listado de Base de Datos (Modelo de Datos) y Transacción Articulo

4.2.2.3.12.3. Análisis de Impacto y Reorganización de la Base de Datos.

Vuelva a su Modelo de Prototipo y efectúe un Análisis de Impacto en el mismo.

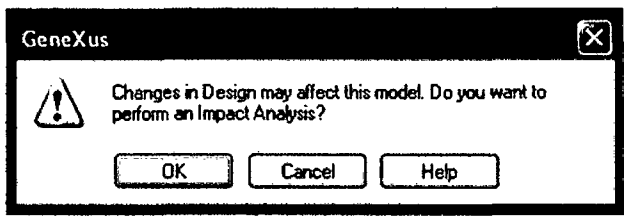
Análisis de Impacto: Siempre que vaya del Modelo de Diseño a un Modelo de Prototipo o Producción (modelo objetivo), Genexus estima si el modelo objetivo debe ser actualizado para que coincida con el modelo de datos del Modelo de Diseño. Si es así, Genexus analiza el impacto de los cambios en la base de datos del modelo. Esto se llama Análisis de Impacto y produce un Reporte de Análisis de Impacto que contiene lo siguiente:

- Una descripción de la conversión de los datos (reorganización) a efectuar.
- Advertencias sobre problemas posibles que pueden darse durante el proceso de reorganización (inconsistencias producidas por nuevas reglas aplicadas a viejos datos, etc.)

En base a la información presentada en el Reporte de Análisis de Impacto, usted puede decidir si continúa con el proceso de reorganización o no.

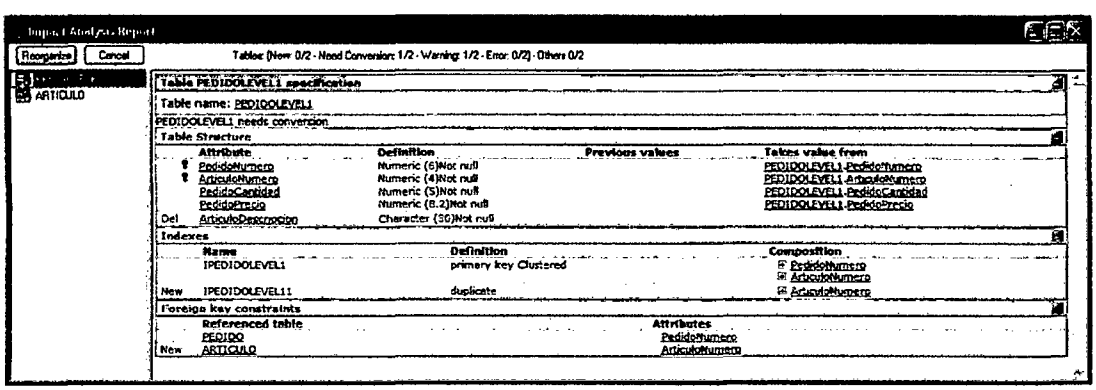
Reorganización o Programas de Conversión: Cuando usted está listo para proceder con la reorganización de la base de datos en el modelo objetivo, usted crea los Programas de Reorganización y los ejecuta. Los programas de reorganización crean un nuevo esquema de base de datos en la base de datos física del modelo objetivo y transportan los datos desde el esquema viejo al nuevo. Este proceso es generalmente considerado como una refactorización de la base de datos efectuada automáticamente por Genexus.

1. Seleccione Prototipo en el menú desplegable de la Model toolbar.
2. Se le indicará que efectúe un Análisis de Impacto. Haga clic en OK.



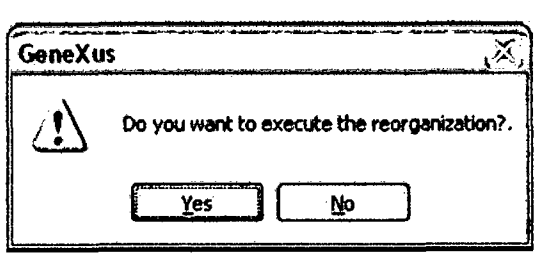
Caja de Diálogo del Análisis de Impacto

3. El Informe de Análisis de Impacto describe los cambios requeridos en la Base de Datos física del Modelo de Prototipo para que coincida con el Modelo de Datos actual en el Modelo de Diseño.



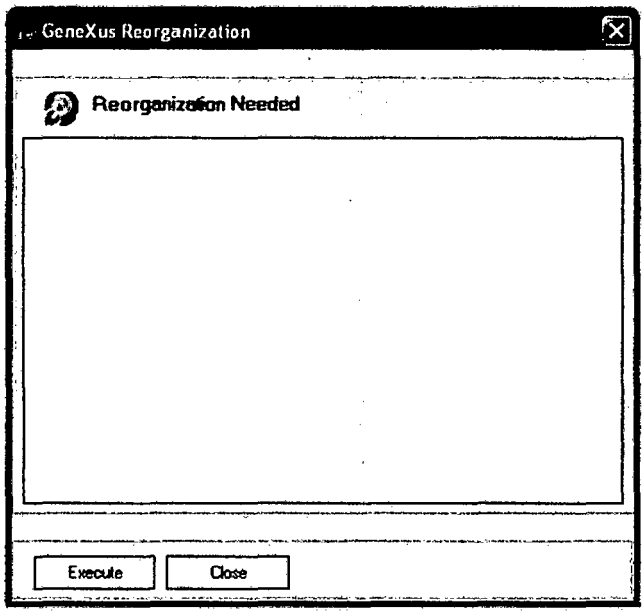
Reporte del Análisis de Impacto

4. Haga clic en el botón **Reorganize** del Informe del Análisis de Impacto para generar los programas de reorganización.
5. Cuando los programas de reorganización estén listos para ejecutarse, se le indicará que los ejecute. Haga clic en **Yes**.



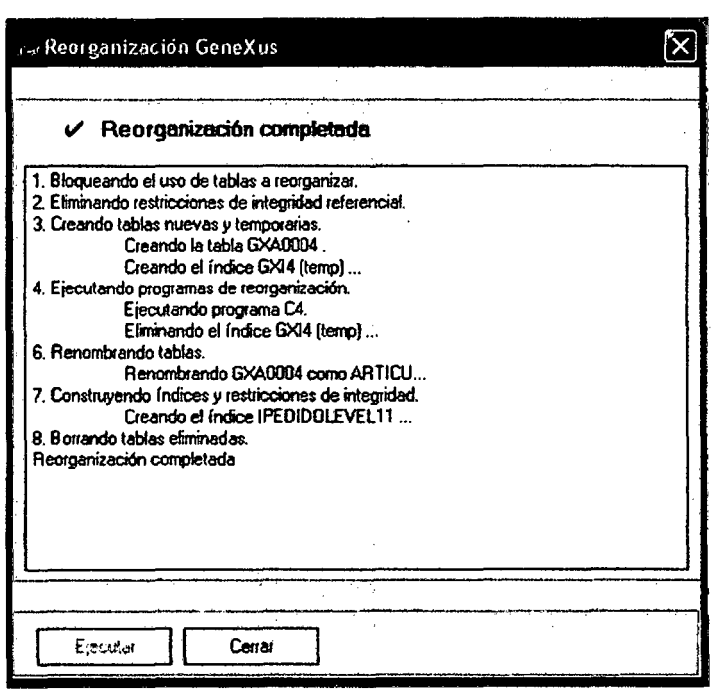
Caja de Diálogo para la Ejecución de la Reorganización

6. Los programas de reorganización le indicarán que debe reorganizar la base de datos de su Modelo de Prototipo. Haga clic en **Execute** (la reorganización).



Caja de Diálogo de Reorganización de Genexus

7. Cuando aparece el mensaje "Reorganization completed" en la ventana de Reorganización de Genexus, haga clic en **Close**. Ahora la base de datos física de su Modelo de Prototipo coincide con el Modelo de Datos definido en su Modelo de Diseño.



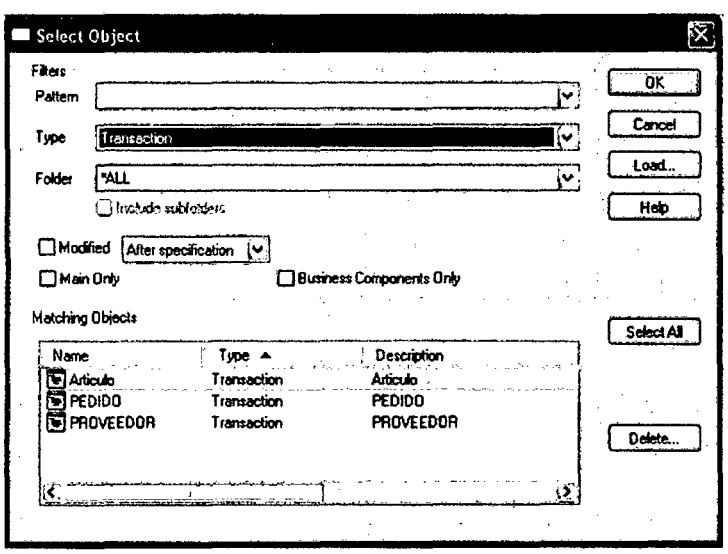
Caja de Diálogo de Reorganización de Genexus

4.2.2.3.12.4. Regeneración de los programas de la Aplicación.

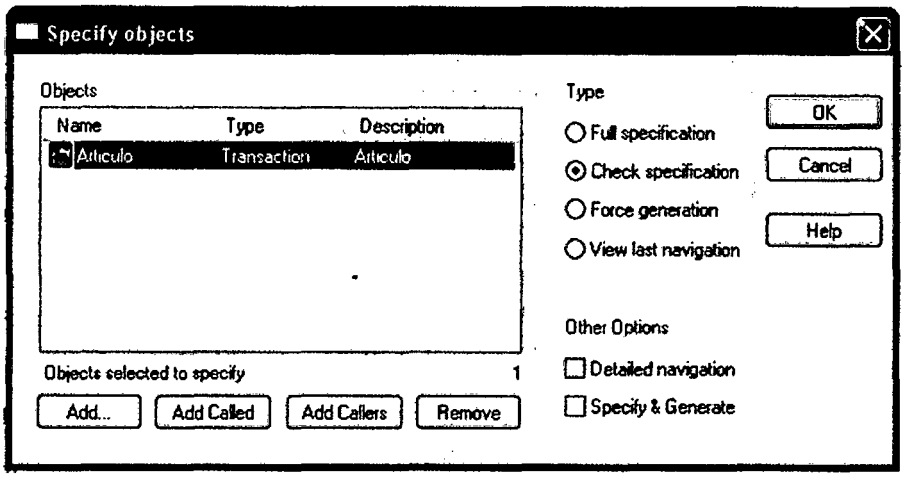
Una vez que la base de datos de su Modelo de Prototipo (o Producción) está sincronizada con el modelo de datos del Modelo de Diseño, usted generará los programas de la aplicación de su Modelo de Prototipo (o Producción). Por más información consulte la sección sobre Generación Automática de Código.

Para regenerar los programas de la aplicación:

1. En el menú **Build** haga clic en **Specify** o presione **SHIFT + F8** para desplegar el Dialog box Seleccionar Objeto.
2. Seleccione la opción Modificado Después de Especificación para especificar solo los objetos que hayan cambiado después de la última especificación.

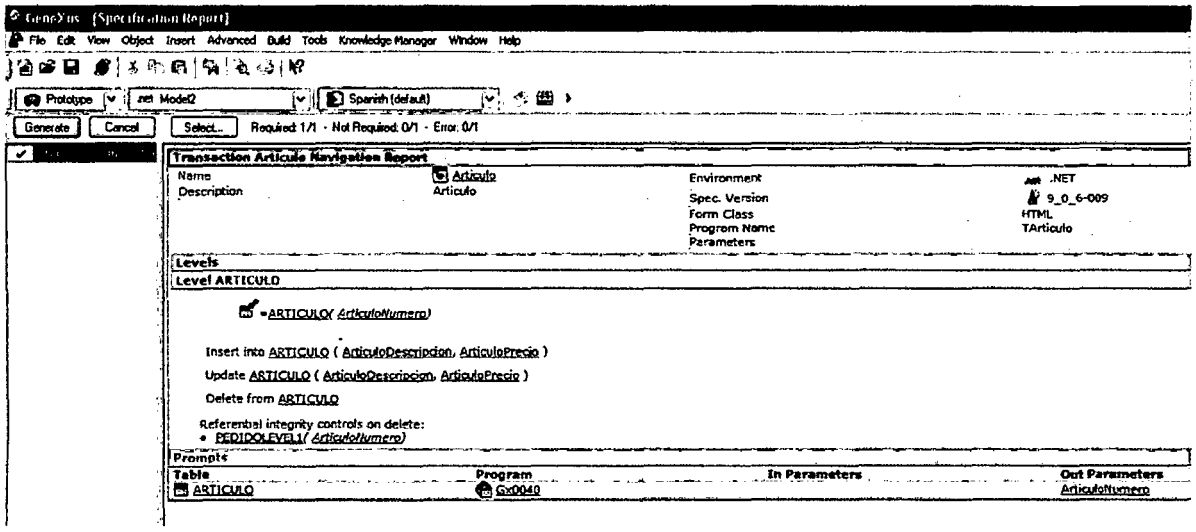


3. Caja de Diálogo para Seleccionar Objeto
4. Haga clic en OK.
5. En el Dialog box Especificar Objetos, seleccione la opción **Check Specification** y haga clic en **OK**.



Caja de Diálogo para Especificar Objeto

6. En el Reporte de Especificación resultante, haga clic en **Generate** para generar los programas asociados a la Transacción Articulo. Notará que Genexus ha agregado automáticamente un objeto **Lista de Selección** para ser especificado y generado. Estos programas se generarán en el código nativo correspondiente del Modelo de Prototipo (o Producción).



Caja de Diálogo del Reporte de Especificación

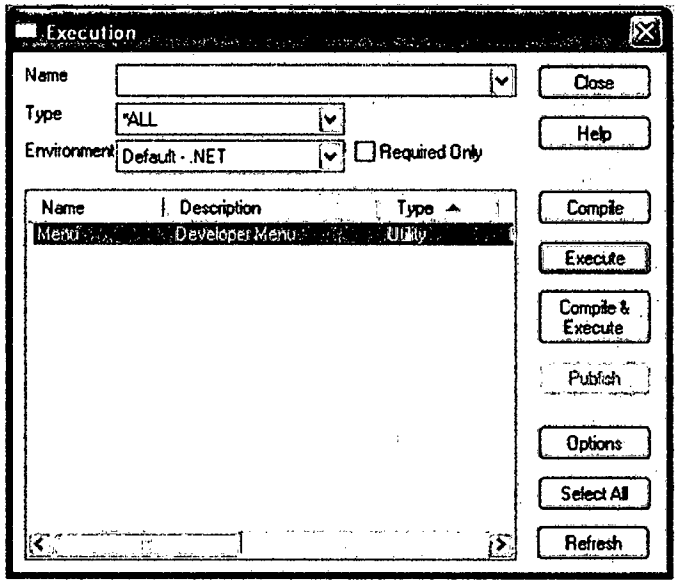
Los Programas de la Aplicación están listos para ser compilados y ejecutado

4.2.2.3.12.5. Compilación y Ejecución de la Aplicación.

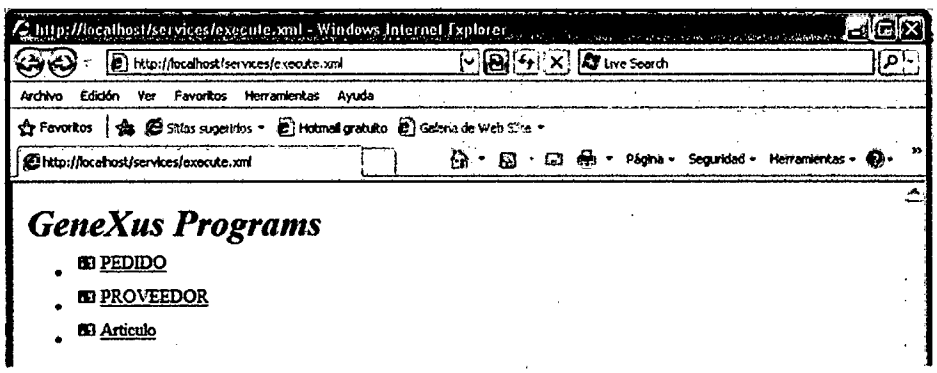
La base de datos del Modelo de Prototipo ya ha sido actualizada y el código de los programas de la aplicación han sido generados.

Ahora es tiempo de compilar y ejecutar la aplicación. Para ello, siga estas instrucciones:

1. Presione F5, y haga clic en Compilar en el Dialog box de Ejecución.



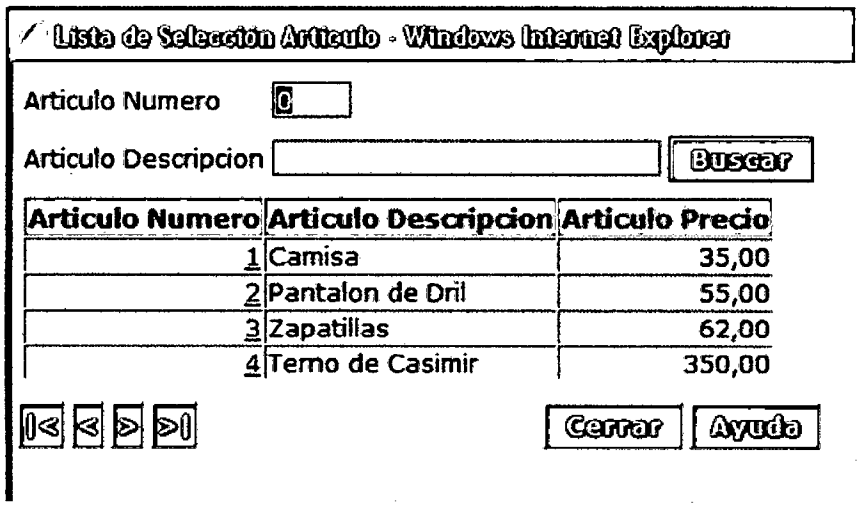
2. Cierre la ventana de compilación de Genexus.
3. Haga clic en Ejecutar en el Dialog box de Ejecución.
4. Caja de Diálogo de Ejecución
5. Haga clic en la opción Producto en el Menú del Desarrollador.



Menú del Desarrollador.

6. Haga clic en el botón Seleccionar para ver los ítems que usted había ingresado originalmente en el Pedido.

Esto significa que los programas de reorganización no solo han cambiado la estructura de la base de datos sino que también han mantenido la información que estaba almacenada la misma.



4.2.2.3.13. Construcción de Procesos no Interactivos (Reportes y Procedimientos)

Reportes y Procedimientos

Hasta ahora hemos trabajado con los Objetos Transacción que son objetos Genexus que requieren la intervención del usuario para insertar, actualizar y eliminar registros en la Base de Datos. No obstante, muchas veces necesitamos realizar tareas sin la intervención del usuario. Para ello, usamos otros dos tipos de Objetos Genexus:

• **Reportes**: Definen procesos no interactivos para consultar a la Base de Datos. La salida del reporte es usualmente enviada a una impresora o desplegada en la pantalla. Los reportes no actualizan la Base de Datos.

• **Procedimientos**: Definen procesos no interactivos para consultar y actualizar la Base de Datos (los procedimientos pueden hacer todo lo que hacen los reportes además de actualizar la Base de Datos). Los procedimientos se usan para definir funciones y subrutinas.

Una funcionalidad clave de los Reportes y Procedimientos Genexus es que ambos están basados en el conocimiento que usamos para definir los Objetos Transacción. Esto significa que la definición de estos objetos se basa en nombres de atributos y no en las tablas de la Base de Datos donde dichos atributos son almacenados. Por lo tanto, la definición de sus Reportes y

Procedimientos permanece válida siempre que los atributos declarados dentro de los mismos estén presentes en la Base de Conocimiento.

En el ejemplo crearemos un Reporte simple (uno que muestre todos los datos de Pedido). Téngase en cuenta que los Reportes y Procedimientos constituyen una de las funcionalidades más potentes de Genexus.

4.2.2.3.13.1. Creación e Invocación de Un Reporte.

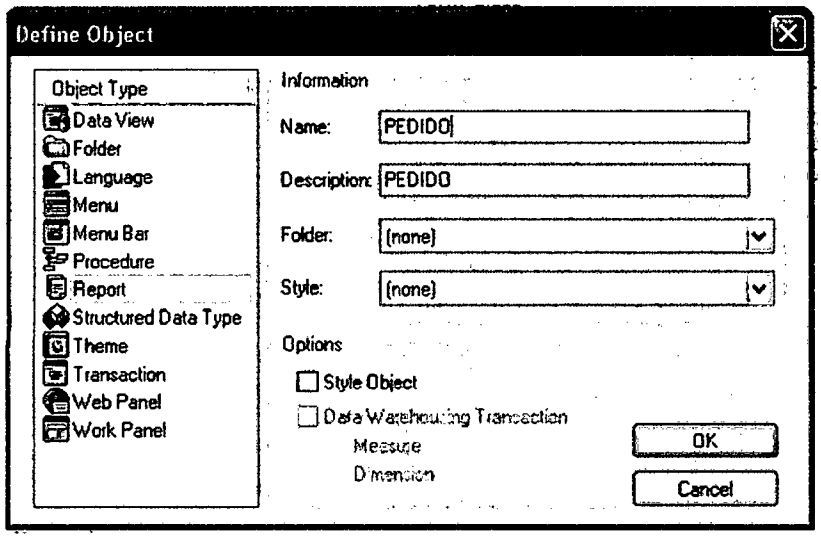
Reportes y Procedimientos

Los Reportes y Procedimientos Genexus comparten las siguientes funcionalidades clave:

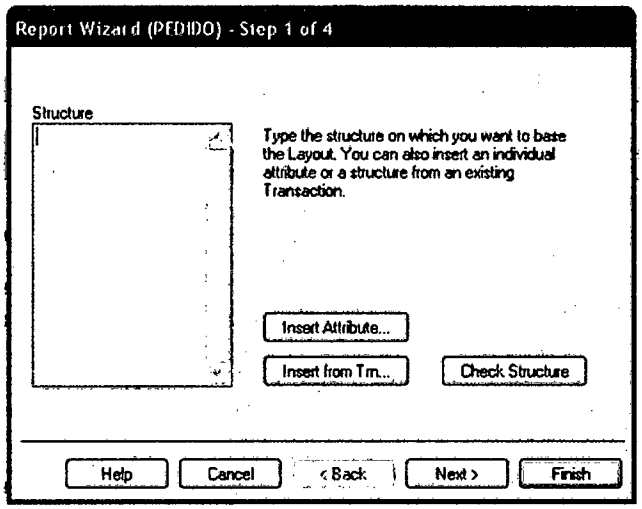
- Proceso de alto nivel: Los Reportes y Procedimientos se especifican en un proceso de muy alto nivel. La secuencia de ejecución es determinada por el analista usando un lenguaje de proceso simple que incluye control, impresión, acceso a datos y otros comandos.
- Basados en Conocimiento: El código fuente de los Reportes y Procedimientos se refiere a nombres de atributos en la Base de Conocimiento, no a su ubicación en la Base de Datos física (del Modelo de Prototipo o Producción). Por lo tanto:
- Genexus sabe dónde encontrar los atributos en la base de datos física.
- Genexus conoce las relaciones entre las tablas en la base de datos física.
- Los Atributos Fórmula son inferidos automáticamente por Genexus.
- Los cambios en la Base de Datos física no afectan el comportamiento de los Reportes y Procedimientos.

Debido a que no habrá ningún cambio en el esquema de la Base de Datos, crearemos un reporte en el Modelo de Prototipo; invocaremos a ese reporte agregando un botón de impresión al objeto Factura ya existente.

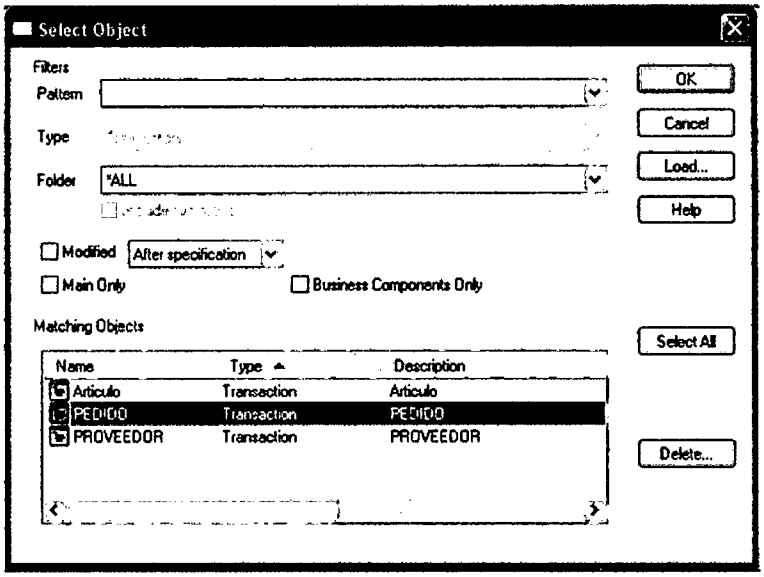
1. En el menú **Object**, haga clic en **New Object**.
2. Seleccione el Tipo de Objeto que quiere crear: Report.
3. Nombre al Objeto: "Pedido". Haga clic en **OK**.



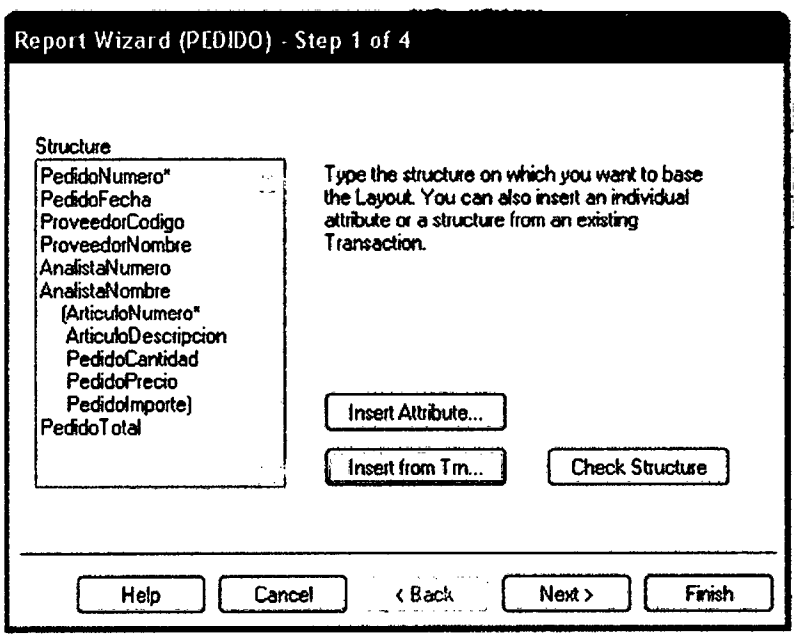
Caja de Diálogo para la Definición del Objeto



4. Haga clic en **Insert from Trn**...(Transacción) en el ayudante de reportes que se desplegará. La opción **Inserte desde Transacción** es en realidad un acceso directo para crear un reporte cuya estructura es exactamente igual a la de la transacción seleccionada. No obstante, usted puede crear reportes compuestos de atributos de muchas transacciones en cualquier orden de significación. Ver pantalla siguiente.



5. Seleccione la transacción Pedido y haga clic en OK.



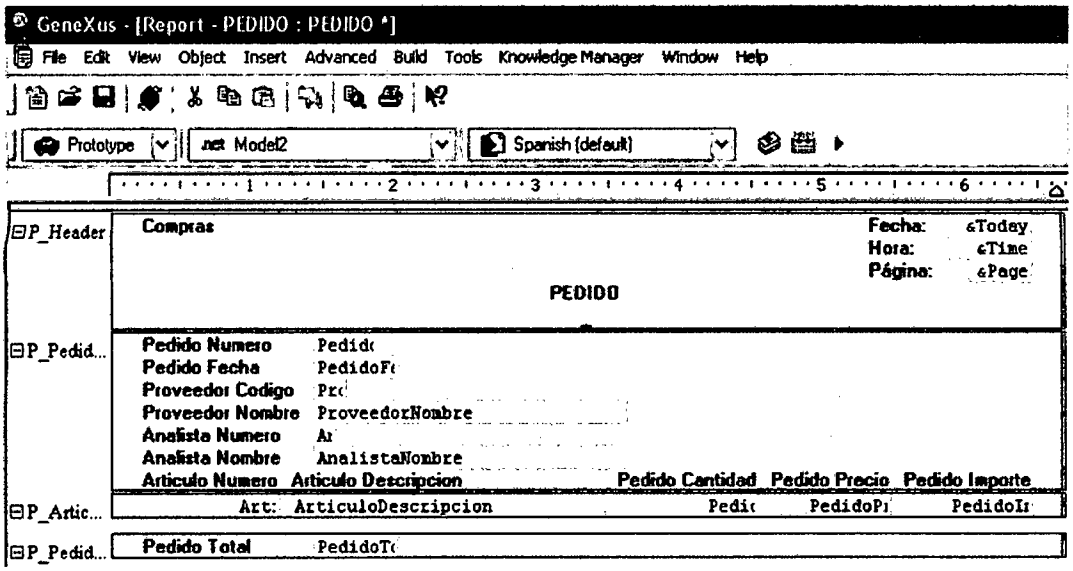
Caja de Diálogo para Seleccionar Objeto

6. Haga clic en Finalizar.

   Estamos salteando pasos del Ayudante de Reportes. Estos pasos nos permiten especificar diversas configuraciones de formato y composición. En este caso aplicamos las configuraciones predeterminadas.

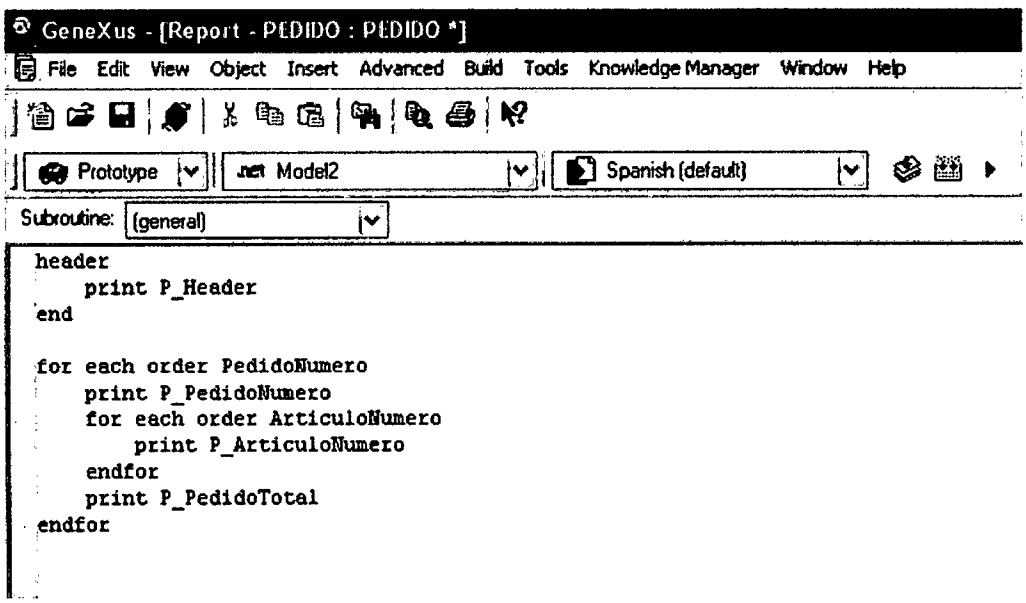
7. Haga clic en la solapa de **Layout** del Reporte de Factura para ver sus Bloques de Impresión. Cada Bloque de Impresión puede

contener un grupo de controles tales como atributos, variables, etiquetas, etc.



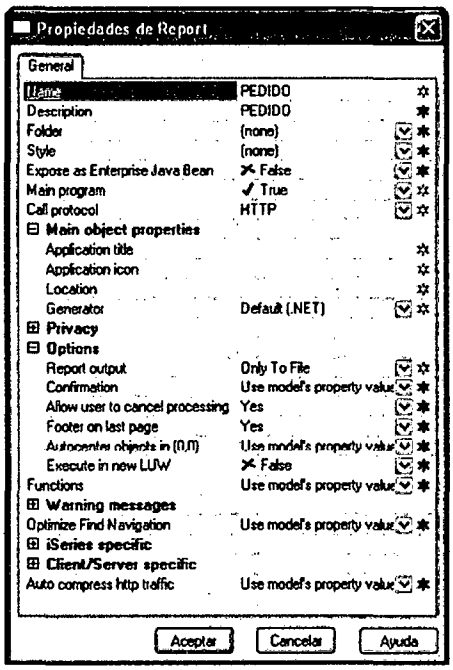
Solapa de Composición del Reporte de Pedido (Layout)

8. La estructura de navegación (que datos se listarán y en qué orden) se define en la solapa **Source**. En este caso, el Código Fuente del Reporte fue generado automáticamente por Genexus.



Solapa Source del Reporte de Pedido

9. En el menú Objeto, seleccione **Propiedades** para ver las propiedades del reporte.
10. Configure la propiedad **Main program** como **True**.
11. Configure la propiedad **Report Output** (debe expandir las opciones del grupo para verla) en **Only to File**.
12. Configure el **Call protocol** como **HTTP** y haga clic en **OK**:



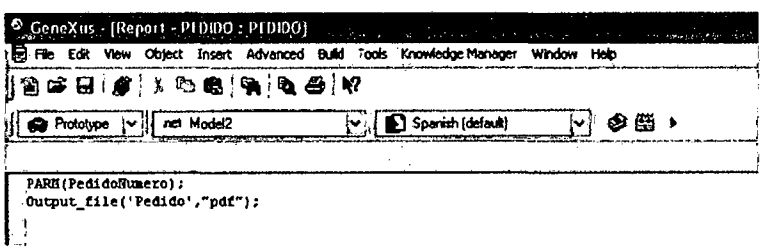
Ventana de las Propiedades del Reporte

Finalmente, para invocar la Reporte de Pedido desde la Transacción Pedido, usaremos dos reglas simples:

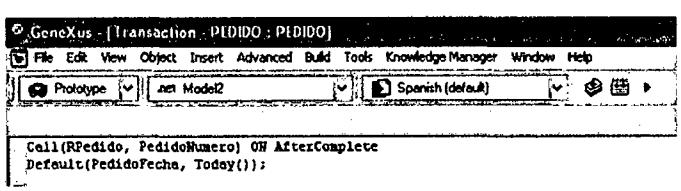
13. En el reporte Pedido seleccione la solapa **Rules**.
14. Escriba:

Parm(PedidoNumero);

Output_file('Pedido',"pdf");



15. Haga clic en **Salvar.**
16. En la Transacción Pedido, seleccione la solapa **Reglas.**
17. Escriba: **Call(RPedido, PedidoNumero) ON AfterComplete**;



Solapa Reglas de la Transacción Pedido

En este punto es importante introducir dos conceptos clave de Genexus: El de tabla Extendida y el del comando For Each.

TABLA EXTENDIDA

Dada una tabla base, su tabla extendida es el grupo de atributos que son directa o indirectamente determinados por la clave de la tabla base:

- Atributos que pertenecen a la tabla base.
- Atributos que pertenecen a todas las tablas que están directa o indirectamente relacionadas en una relación N a 1 con la tabla base.

Usos:

- El concepto de tabla extendida permite que los reportes y procedimientos Genexus permanezcan válidos cuando la estructura de la base de datos cambia.
- La tabla extendida también es también usada por los objetos transacción que pueden insertar, actualizar y eliminar atributos que pertenecen a las tablas extendidas de las tablas base referenciadas en la estructura de la transacción.

Para encontrar la tabla extendida de una tabla dada se puede usar el Diagrama de Bachmann del Modelo de Datos de la Base de Conocimiento:

1. En el menú **Tools** seleccione **Diagrams**
2. Seleccione el tipo de diagrama **Tables** y haga clic en **New**
3. Seleccione las tablas que desea desplegar y haga clic en OK.

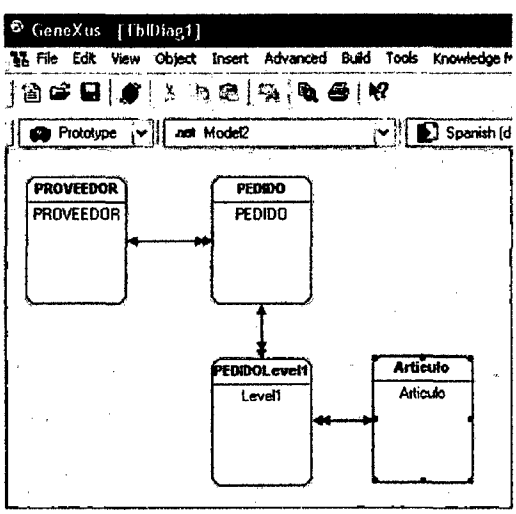


Diagrama de Bachmann del Modelo de Datos

En base a este diagrama podemos identificar la tabla extendida de cada una de las tablas de la aplicación.

Tabla Base	Tabla Extendida
Proveedor	Proveedor
Articulo	Articulo
Pedido	Pedido, Proveedor
Pedido1	PedidoLabel1, Pedido, Proveedor, Articulo

COMANDO FOR EACH

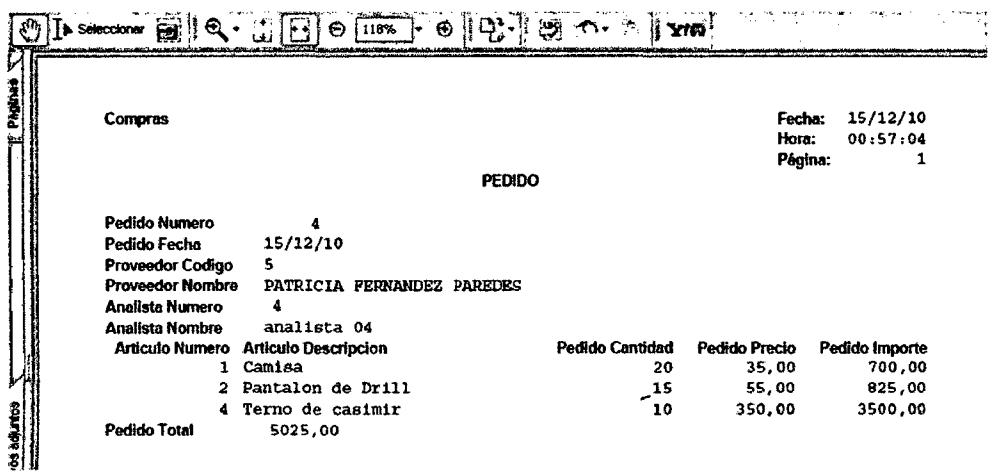
- El comando "**For Each**" es el corazón del lenguaje de procedimientos usado por Genexus para definir reportes y procedimientos. Recupera y actualiza información (mediante procedimientos online) desde su Base de Datos.
- Con el comando **For Each** usted puede definir la información que a la que desea acceder y nombrar los atributos que desea usar. Genexus inferirá automáticamente las tablas apropiadas en tiempo de generación. Cuando la estructura de la Base de Datos cambie, las definiciones del **For Each** permanecerán válidas.
- La sintaxis básica del **For Each** es la siguiente:

```
for each
        print <something>
endfor
```

- Cada comando **For Each** tiene una tabla de Modelo de Datos asociada que será navegada. Está compuesto de la tabla base del For Each y su tabla extendida (las tablas que Genexus necesita para buscar los atributos mencionados en el comando).
- Las tablas que serán accedidas son determinadas por los atributos contenidos en el comando **For Each**.
- Para este grupo de atributos, Genexus inferirá la mínima tabla extendida.
- La tabla base de este grupo de atributos será la tabla base del **For Each**.

4.2.2.3.13.2. Especificación, Generación y Ejecución de la Aplicación.

1. Especifique y ejecute su aplicación siguiendo lo descrito en la sección 4.1.9.1.3.1.: Especificación y Generación Automática de Código- Comando **Build** y en la sección 4.1.9.1.4.1.: Ejecución de la Aplicación.
2. En el Menú del Desarrollador seleccione **Pedido** e ingrese un nuevo Pedido o seleccione un **Pedido** existente. Luego haga clic en **Aply Changes** dos veces. La regla que invoca al reporte se disparará en su navegador.



Compras

Fecha: 15/12/10
Hora: 00:57:04
Página: 1

PEDIDO

Pedido Numero 4
Pedido Fecha 15/12/10
Proveedor Codigo 5
Proveedor Nombre PATRICIA FERNANDEZ PAREDES
Analista Numero 4
Analista Nombre analista 04

Articulo Numero	Articulo Descripcion	Pedido Cantidad	Pedido Precio	Pedido Importe
1	Camisa	20	35,00	700,00
2	Pantalon de Drill	15	55,00	825,00
4	Terno de casimir	10	350,00	3500,00

Pedido Total 5025,00

Reporte de Pedido

4.2.2.3.14. Consultas y Diálogos Interactivos (Work Panels y Web Panels)

4.2.2.3.14.1. Creación de un Web Panel: Trabajar con Proveedores.

Work Panels y Web Panels

Los Work Panels son usados para hacer diálogos y consultas interactivos a la base de datos en ambientes GUI o Windows.

Los Web Panels son equivalentes a los Work Panels para los ambientes Web. No obstante, existen algunas importantes diferencias entre ellos debido a naturaleza desconectada de Internet.

En los Web Panels, los resultados de los diálogos y las consultas son formateados como páginas HTML en tiempo de ejecución y enviados al navegador.

Work y Web Panels son programados usando un lenguaje simple dirigido por eventos.

Programación Dirigida por Eventos: Es un estilo de programación en el cual las aplicaciones contienen código que permanece inactivo hasta ser llamado para responder a eventos disparados por el usuario o por el sistema.

Eventos Definidos por el Usuario: Son eventos creados por el analista. Estos eventos tienen un nombre y código que se ejecutan solo cuando el usuario final hace clic en la tecla o botón asociado al evento correspondiente.

Eventos del Sistema: Los siguientes eventos están predefinidos en Work Panels y Web Panels.

• **Start:** Ocurre solo una vez, cuando se inicia la ejecución del Work Panel o Web Panel.

Cada vez que ejecutamos un work panel, lo primero que se ejecuta es el código asociado al evento **start**.

• **Refresh**: Ocurre solo una vez, justamente antes de que los datos sean cargados desde la Base de Datos a la grilla.

• **Load:** Ver "Carga de Datos" más adelante.

• **Enter:** El código asociado a este evento es ejecutado cada vez que el usuario presiona la tecla **Enter** o el botón **"Confirmar".**

• **Exit**: Ocurre solo una vez, al finalizar la ejecución del Work Panel o Web Panel. Cada vez que se cierra un Work panel, la última cosa que se ejecuta es la código asociado con el evento **exit**. Los Work Panels y Web Panels pueden cerrarse con el botón **Cerrar** o la tecla **Exit** (ESC o F12), o si se ejecuta el comando "retornar" incluido en un evento.

Carga de Datos: Genexus recupera los datos a ser desplegados en los Work Panels y Web Panels con el mismo procedimiento usado en los Objetos Reportes y Procedimientos. En este caso, Genexus realiza un **For Each** implícito en base a los siguientes atributos:

- Todos los atributos desplegados en el formulario.
- Todos los atributos usados en los eventos con excepción de aquellos incluidos en comandos **For Each** definidos por el analista en los **Work Panels** o **Web Panels** (estos no se considerarán en el For Each implícito).

- Todos los atributos de la grilla que estén escondidos, es decir, los que hayan sido escondidos usando el botón "Esconder" en la solapa "General" del diálogo de propiedades de la grilla.
- Todos los atributos de la grilla que han sido insertados en la pantalla "Orden de la Grilla". Esta pantalla es accedida mediante la opción "Orden" en el acceso directo desplegado haciendo clic derecho sobre la grilla.
- Todos los atributos de la grilla que se han usado en las condiciones de la grilla. Con todos estos atributos, Genexus infiere la tabla base correspondiente. Entonces, navega en la tabla base inferida y accede a las tablas extendidas para recuperar los valores de los atributos de la grilla, tanto los visibles como los escondidos. Por cada registro encontrado, se ejecuta el código asociado al Evento Load y después el registro es cargado en la grilla.

Manejo de Eventos en Web Panels: Cuando un usuario Web accede a una página del servidor para verla, el Navegador baja la página al cliente. Después de esto, es imposible saber que está haciendo el usuario en el cliente, hasta que el usuario dispara un evento, como por ejemplo el **evento Enter**. Cuando el usuario hace clic en **Enter**, la página modificada es enviada (presentada) al servidor para continuar con el proceso

La primera vez que se ejecuta el Web Panel (Get), los eventos correspondientes se disparan en el siguiente orden:

1. **Start**, 2. **Refresh**, 3. **Load**

Después, cuando se presiona el botón asociado al evento **Enter** o a un evento de usuario, o cuando se ejecuta un evento asociado a una **imagen** (que no llama a otro web panel) haciendo clic en el mismo, el web panel será ejecutado nuevamente y el orden de disparo será el siguiente (Post):

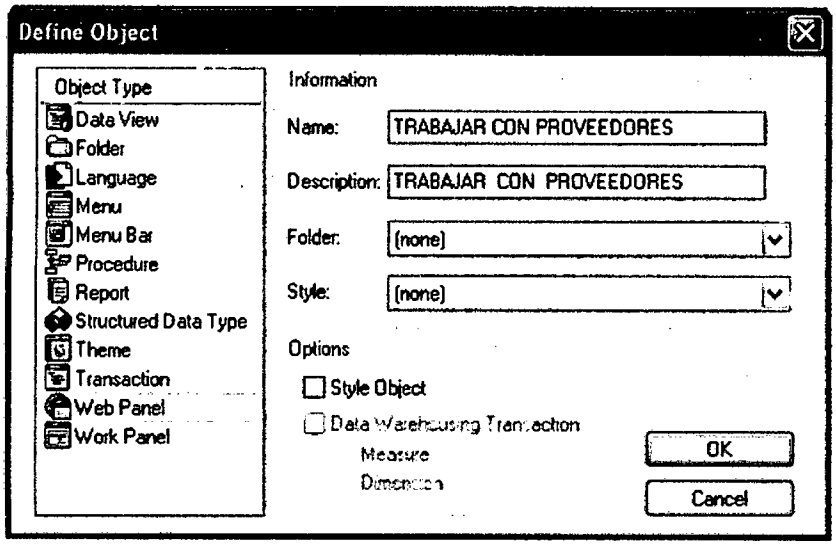
1. Start
2. Leer variables en la pantalla
3. Evento Enter o eventos definidos por el usuario
4. Refresh
5. Load

El orden de los eventos aclara el concepto de "presentar" los valores ingresados por el usuario.

Para crear un Web panel, siga los pasos siguientes:

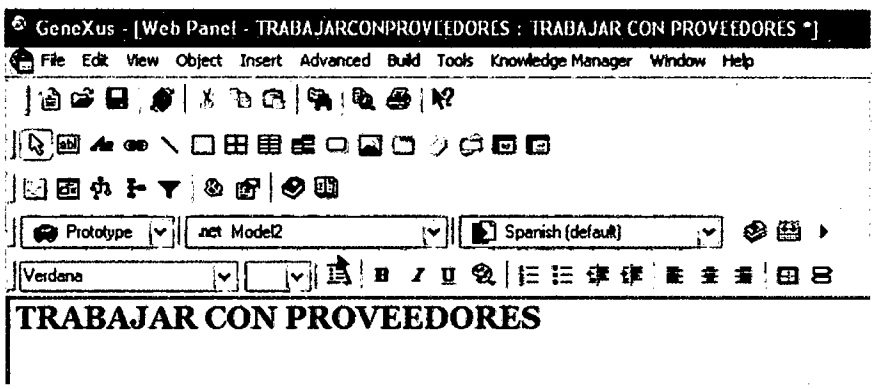
1. En el menú **Objeto** haga clic en **Nuevo Objeto**.

2. Seleccione el **Tipo de Objeto** que desea crear: **Web Panel.**
3. Nombre al Objeto: "Proveedor"
4. Describa al Objeto como: "Trabajar con Proveedores".
5. Haga clic en **OK.**



Caja de Diálogo para Definir Objeto

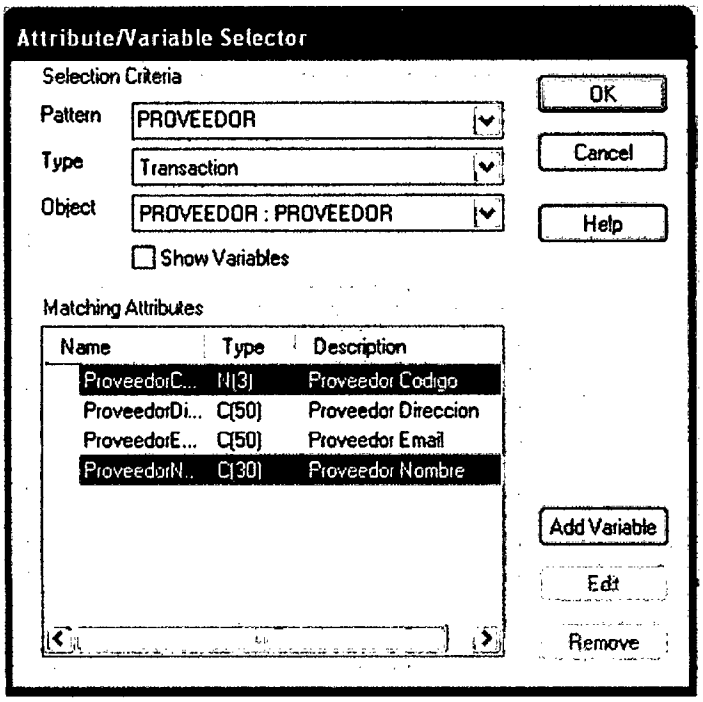
6. Aparecerá el Formulario vacío del Web Panel.Escriba: **Trabajar con Proveedores**
7. Use el botón **Format** de la **Barra de Herramientas de Formato** para configurar el texto como **Título 1.**



Formulario de Web Panel con Barra de Herramientas de Formato

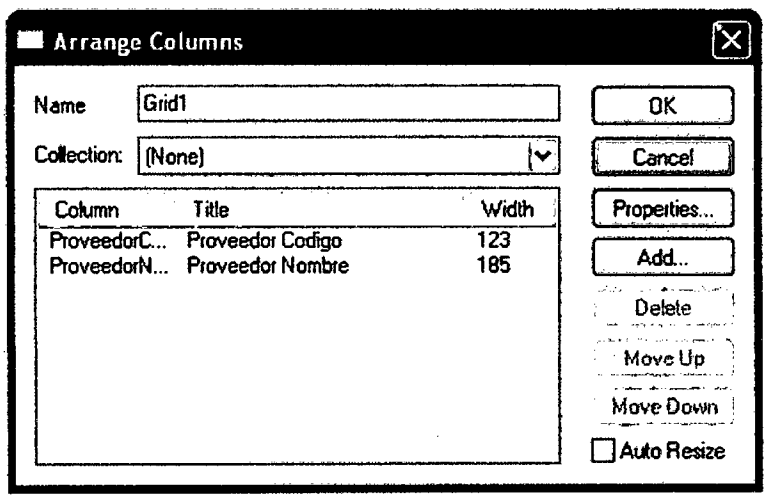
8. Presione **Enter**.
9. En el menú Insertar seleccione Grilla. Aparecerá la ventana Selector de Atributo /Variable.
10. Escriba "Proveedor" como patrón (filtro).
11. Elija **Transaction** como tipo de objeto a filtrar.

12. Seleccione los atributos **ProveedorCodigo** y **ProveedorNombre** y haga clic en **OK**.



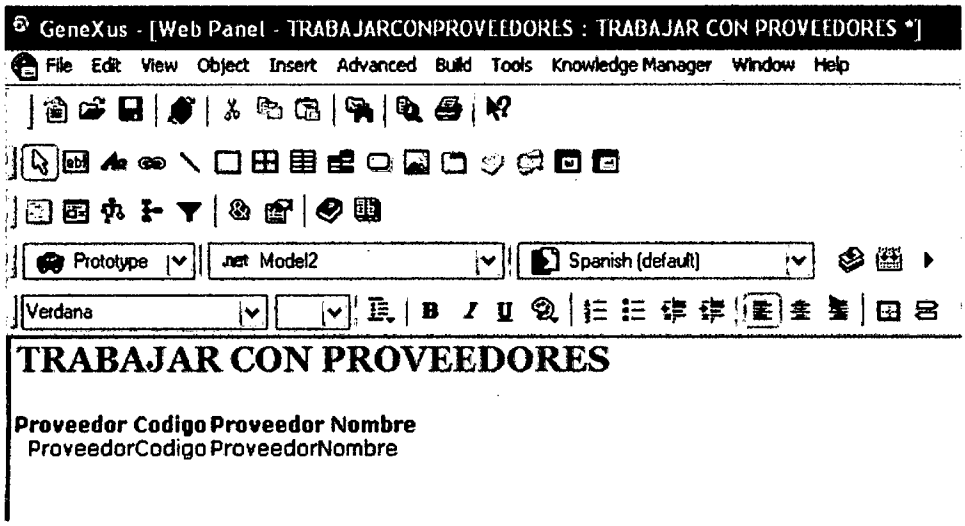
Ventana del Selector de Atributo /Variable

13. En la ventana **Arrange Columns** (Arreglar Columnas) haga clic en **OK.**
14. Haga clic fuera del área de la grilla para eliminar la selección.



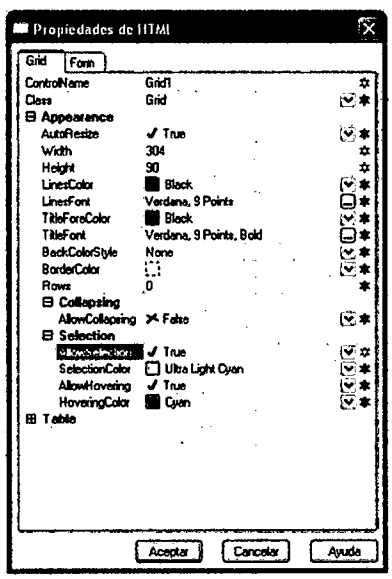
Ventana Arreglar Columnas

16. Haga clic en el botón **Show Borders** (Mostrar Bordes) de la Barra de Herramientas de Formato para ver los bordes de la grilla.



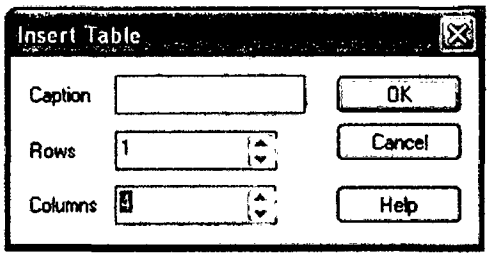
Formulario de Web Panel con grilla

16. Haga clic derecho sobre la Grilla y seleccione **Properties**.
17. Configure la propiedad **AllowSelection** en True y haga clic en **OK**. Esto le permitirá seleccionar clientes de la grilla sencillamente hacienda clic sobre ellos.
18. Haga clic fuera del área de la grilla para eliminar la selección y presione **Enter** una vez.



Ventana de Propiedades de la Grilla

19. En el menú **Insertar** seleccione **Table** (Tabla).
20. Aparecerá la ventana **Insertar Tabla**. Cree una tabla con 1 fila (Row) y 4 columnas (Columns) sin ningún literal.

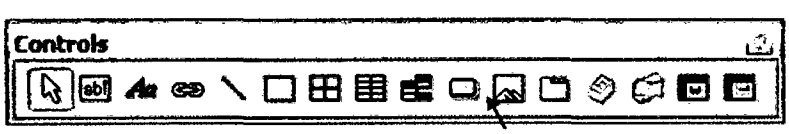


Ventana para Insertar Tabla

21. Cambie el tamaño de la tabla arrastrando los puntos de una esquina de la misma con el mouse.

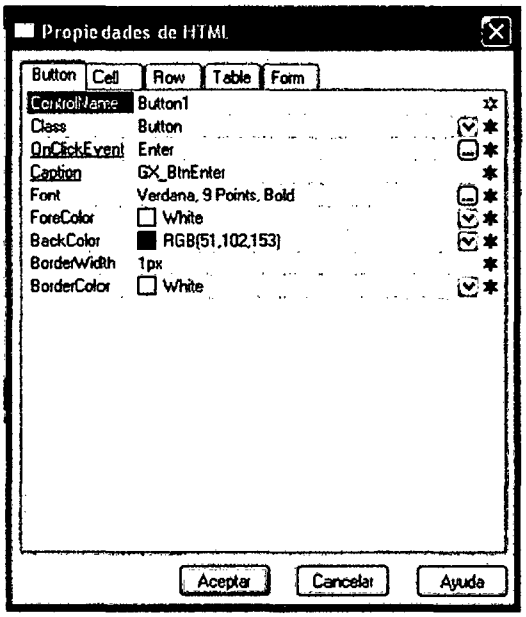
Formulario de Web Panel con grilla y tabla

22. En el menú **View**, seleccione **Toolbars / Palette**
23. Coloque el cursor en la primera columna de la tabla y después presione el icono **Botón** en la Barra de Herramientas de la Paleta (Controles).



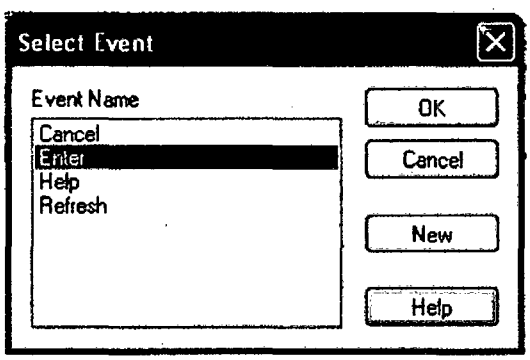
Barra de Herramientas de la Paleta (Controles) – Icono del **Botón**

24. Haga clic derecho sobre el botón y seleccione **Properties**. Aparecerá la ventana con las propiedades HTML del botón.
25. Haga clic en la **elipsis** (...) de la propiedad **OnClickEvent.**



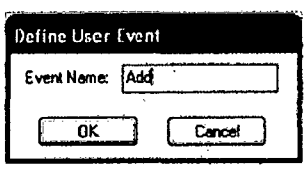
Propiedades HTML del Botón

26. Haga clic en el botón **New.**



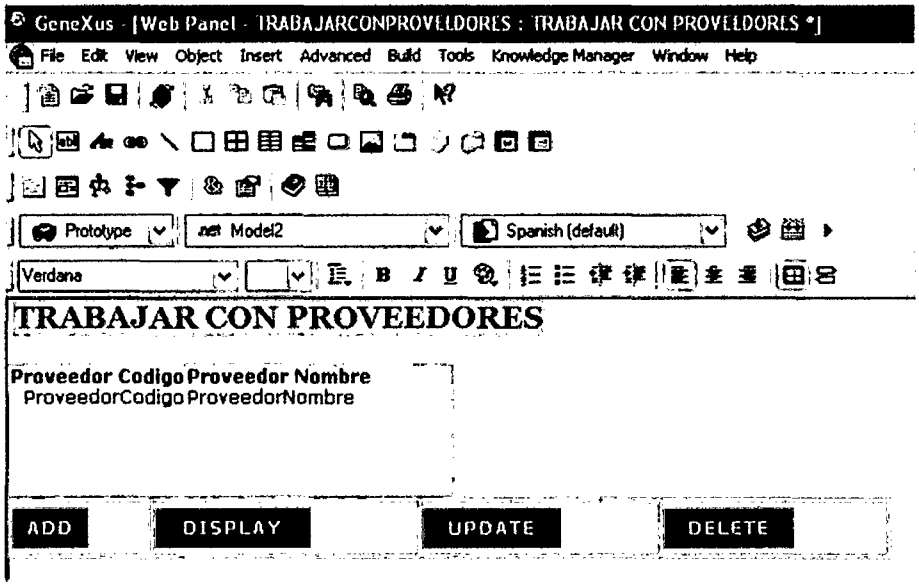
Evento Select

27. Escriba **Add**, para definir un botón que agregará una instancia del cliente y haga clic en **OK**.
28. Haga clic en **OK**.

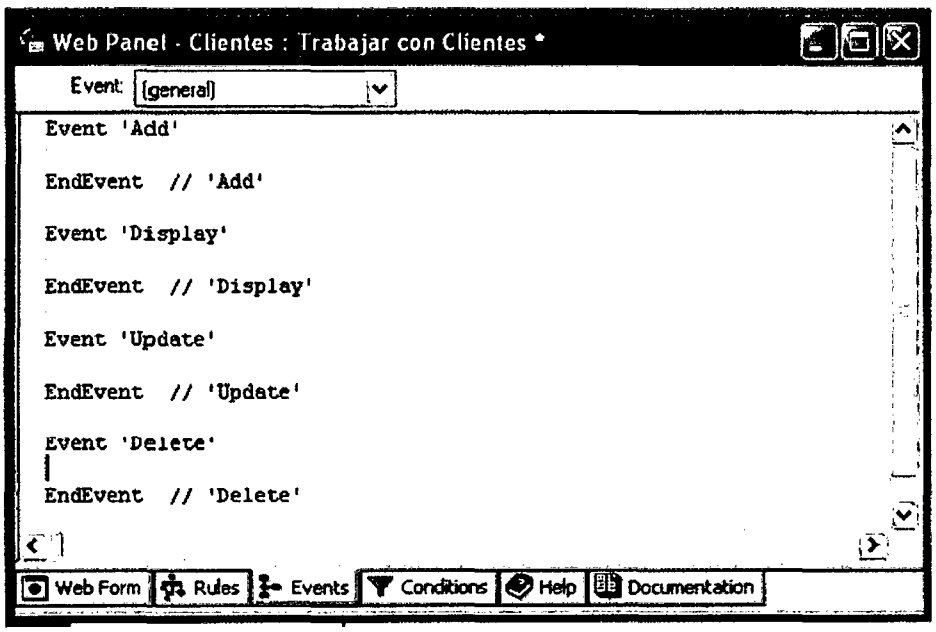


Evento **Define User**

30. Repita los pasos previos para crear los botones y eventos **Display**, **Update** y **Delete**. El Web Form del Web Panel lucirá como se muestra aquí.

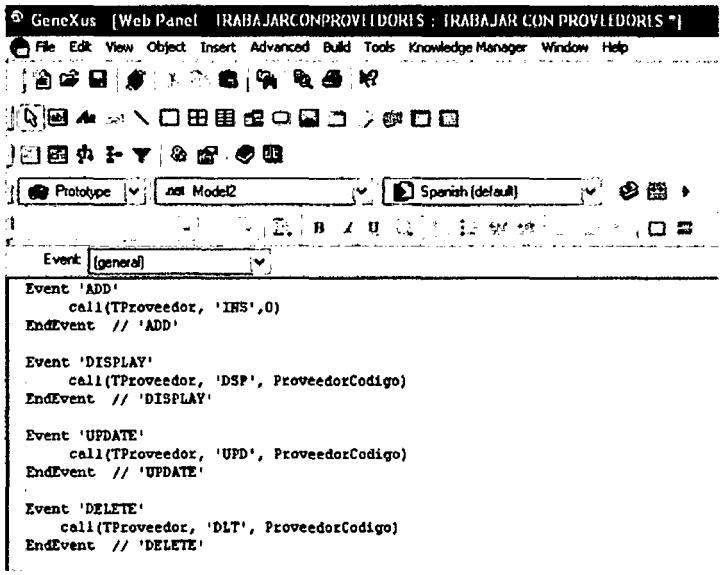


31. Haga clic en la solapa **Events** (Eventos) del Web Panel.



Formulario del Web Panel – Solapa Eventos

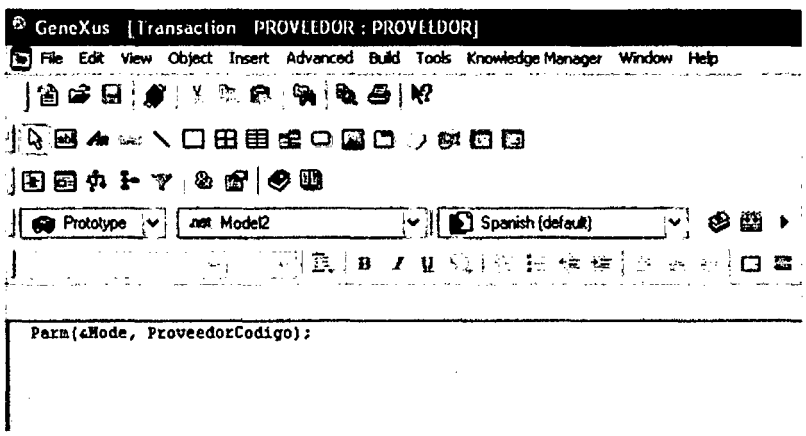
32. En el Evento Add escriba la siguiente función: **call(TProveedor, 'INS',0)**
33. En el Evento Display escriba: **call(TProveedor, 'DSP', ProveedorCodigo)**
34. En el Evento Update escriba: **call(TProveedor, 'UPD', ProveedorCodigo)**
35. En el Evento Delete escriba: **call(TProveedor, 'DLT', ProveedorCodigo)**
36. Salve el Web Panel.



Código de los Eventos del Web Panel

37. Vaya a la solapa de **Rules** de la Transacción Proveedor y escriba la siguiente regla: **Parm(&Mode, ProveedorCodigo);**

Esta regla define que el objeto **Proveedor** tomará dos parámetros de ahora en adelante. El primer parámetro será asignado a la variable local **&Mode**. El Segundo será asignado al atributo **ProveedorCodigo**, definiendo una instancia específica del objeto Cliente.



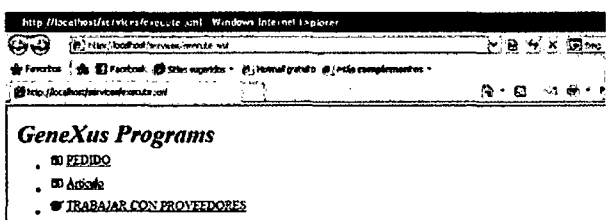
Reglas de la Transacción Proveedor

Desde las funciones de invocación del Web Panel sabemos que el primer parámetro define la operación (insertar, actualizar, eliminar o desplegar) que el usuario desea ejecutar sobre el objeto Provedor. Esto se llama **modo de la transacción** y puede tomar los valores **'INS', 'UPD', 'DLT', y 'DSP'**, respectivamente. Cuando un objeto transacción es invocado con un modo específico, el usuario solo puede ejecutar la operación especificada por el modo, es decir, insertar, actualizar o eliminar la instancia de la transacción.

Cuando se hace clic en el botón Agregar, la propiedad **ProveedorCodigo Autonumber** ignora el valor recibido en el segundo parámetro.
Cuando usted define la regla **Parm** dentro de un objeto, usted está diciendo que el objeto solo puede ser invocado por otros objetos. Por lo tanto, el objeto Proveedor no aparecerá más en el Menú del Desarrollador.

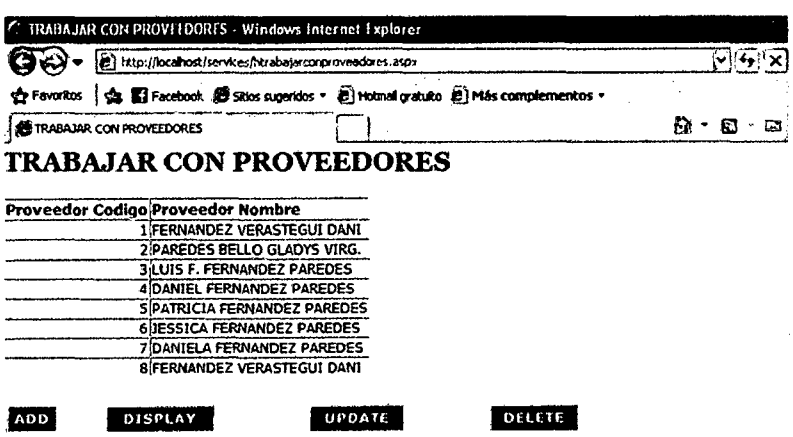
4.2.2.3.14.2. Ejecutar el Web Panel Trabajar con Proveedores.

1. Especifique genere, compile y ejecute la aplicación siguiendo los Pasos de la sección 4.1.9.3.1.: Especificación y Generación Automática de Código _ **Comando Build** hasta la sección 4.1.9.4.1.: Ejecución de la Aplicación.
2. Ejecute el Web Panel Trabajar con Proveedores.



Menú del Desarrollador con Web Panel

Ahora usted puede agregar, desplegar, eliminar o actualizar instancias del cliente desde el web panel Trabajar con Proveedores.

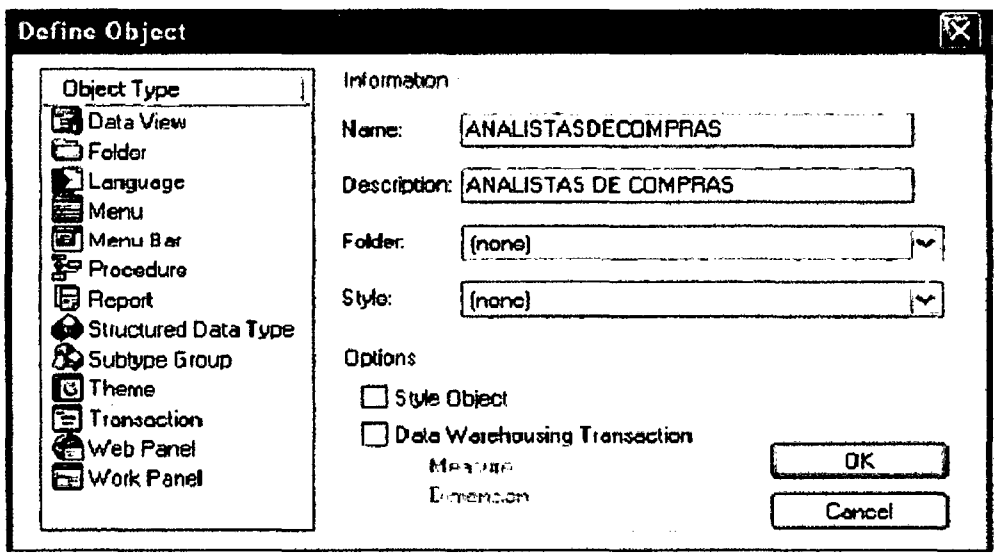


Menú Trabajar con Proveedores

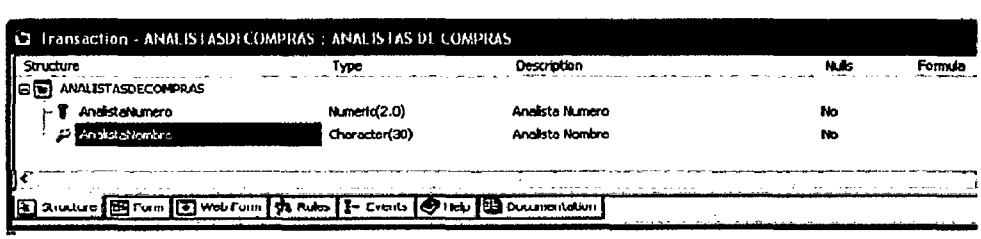
4.2.2.3.15. Inclusión de Objeto Genexus Transacción Analistas de Compras

Es posible adicionar objetos transacción a este nivel del desarrollo, para lo cual tendríamos que ubicarnos en la fase de Diseño, ingresar la

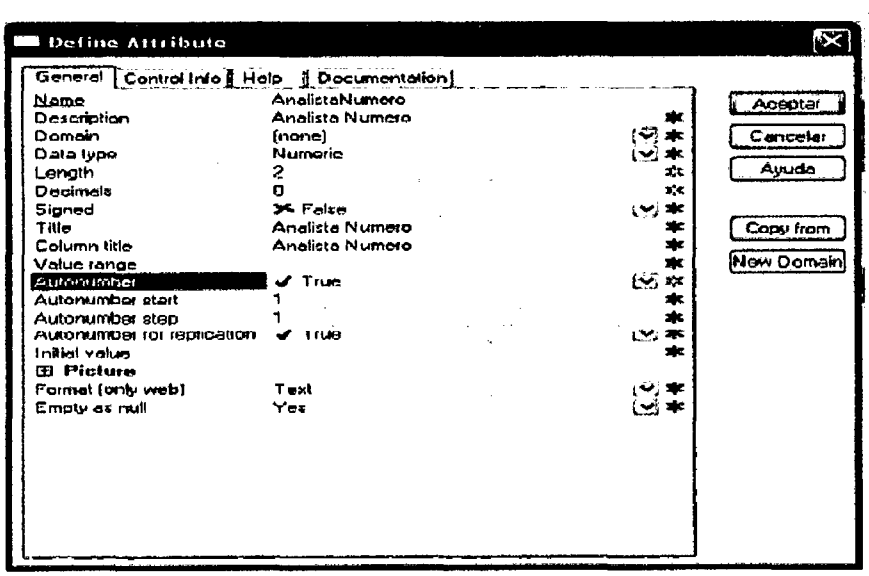
transacción Analistas de compras, luego ubicarnos en la Fase de Prototipo, Reorganizar la base de Datos, especificar, Generar y compilar los programas. (de acuerdo a secciones 1.1.1.5.1, 1.1.1.5.2, 1.1.1.5.3., 1.1.1.5.4., y 1.1.1.5.5) Como resultado obtenemos las siguientes imágenes:



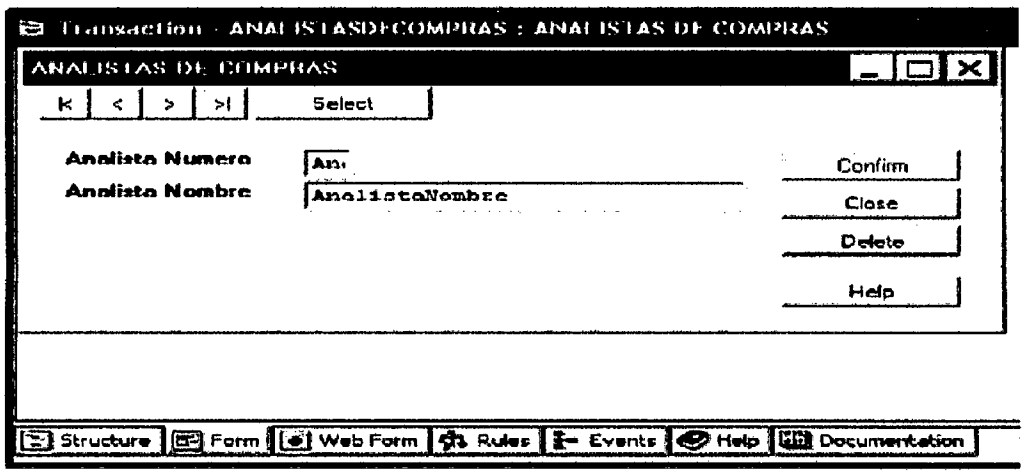
Creación de Objeto Transacción Analistas de compras



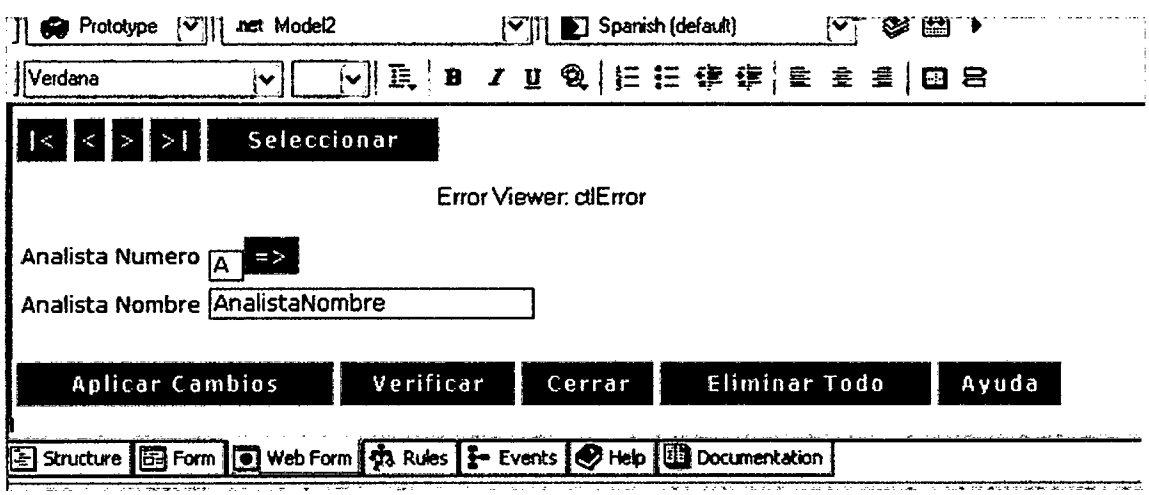
Estructura de transacción Analistas de compras



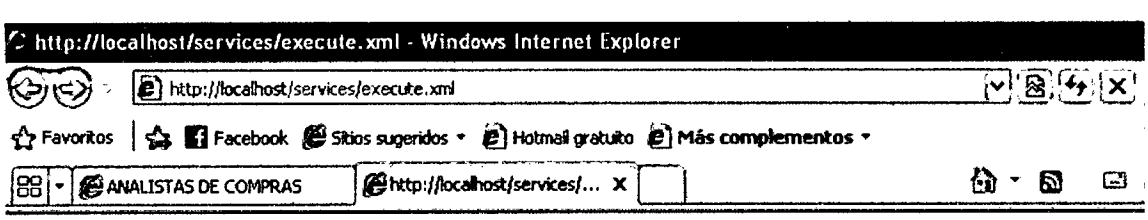
Configuración de propiedades de atributo AnalistaNumero de transacción Analistas de crédito.



Form Windows de Transacción Analistas de crédito



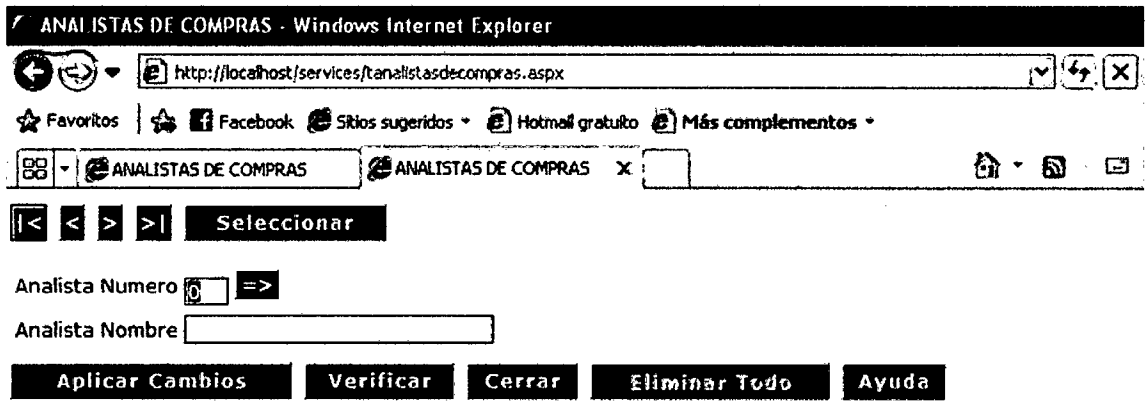
Web Form de Transacción Analistas de compras



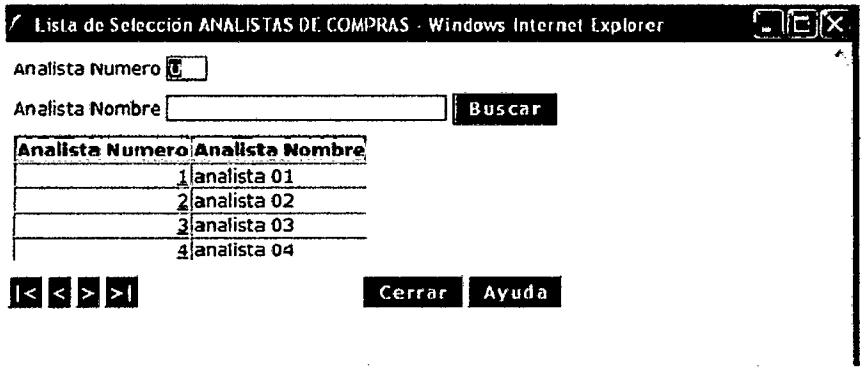
GeneXus Programs

- PEDIDO
- Articulo
- ANALISTAS DE COMPRAS
- TRABAJAR CON PROVEEDORES

Menú del Desarrollador incluyendo transacción Analistas de compras



Ejecución de Transacción Analistas de compras



Lista de selección de transacción Analistas de compras

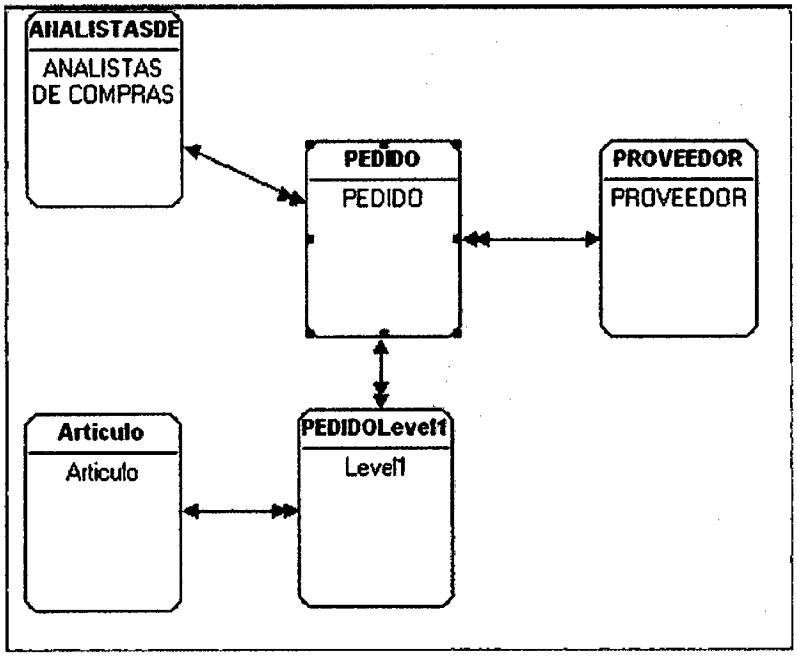
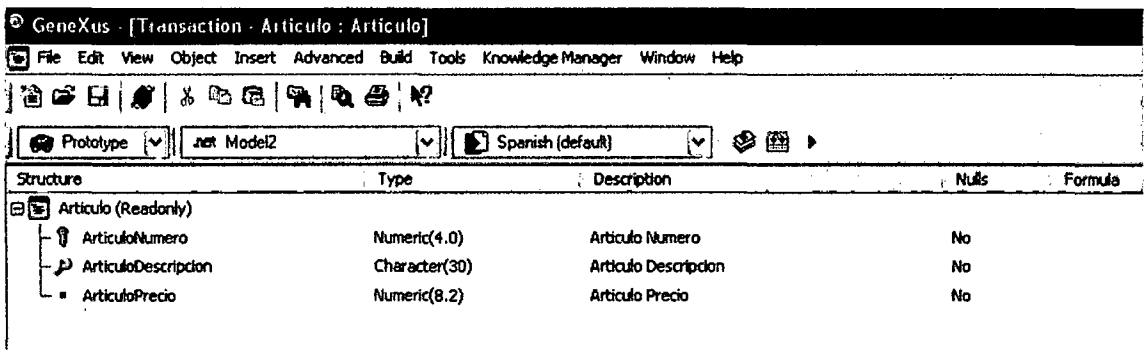


Diagrama de Bachman del Modelo de Datos.

4.2.2.3.16. Creación y consumo de web services con Genexus.

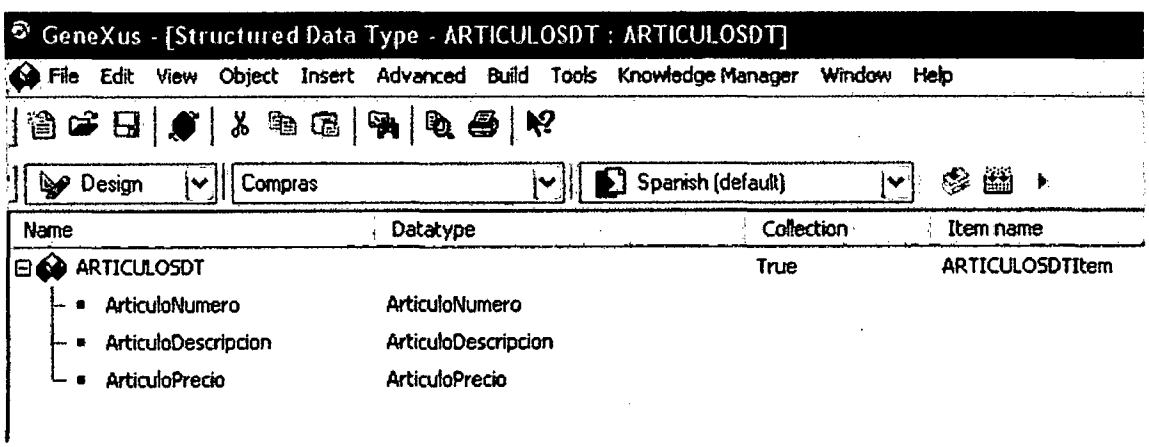
A manera de demostración detallaremos los mecanismos para proveer y consumir Web Services con Genexus, en este caso el Web Service de PRODUCTOS, el cual contiene los productos crediticios ofertados por MicreditPerú.

Usando Genexus podemos ver que la Transaction ARTICULO tiene la estructura que se muestra en la imagen de pantalla siguiente:



En base a la Transaction ARTICULO, vamos a definir un Web Services al que llamaremos ARTICULO.

Lo primero es definir un objeto Genexus denominado Tipo de Dato Estructurado (SDT) al que llamaremos ARTICULOSDT, que nos permita almacenar la información que nos interesa publicar y tendrá la siguiente estructura:



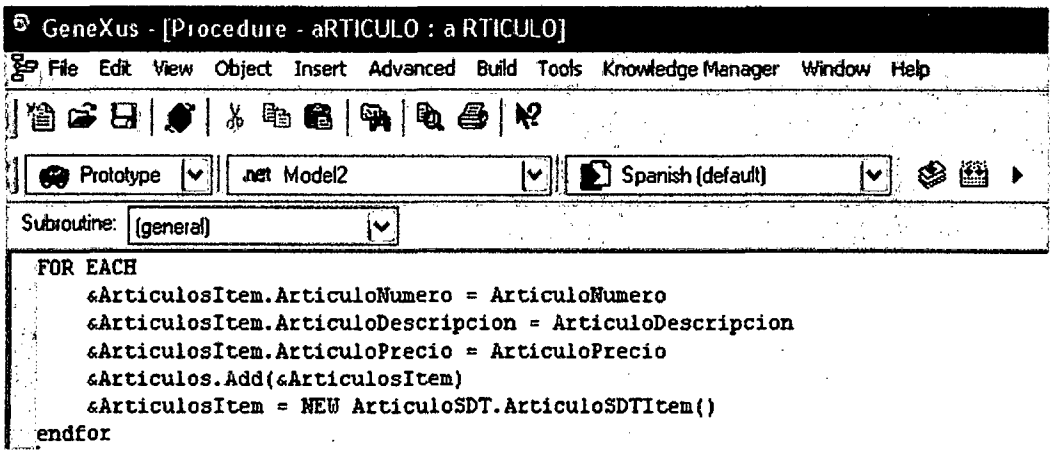
ARTICULOSDT será una colección

En la pantalla anterior, al fijar la columna **collection** con **TRUE**, automáticamente sale el nombre de **ArticuloSDTItem** en la columna denominada **Item name**

Luego crearemos el objeto Genexus procedimiento al que denominaremos ARTICULO, donde debemos definir las variables siguientes:

Una variable que permita almacenar toda la colección de articulos (&Articulo), junto con una variable que cargue cada uno de los Items de esa colección de productos (&ArticuloItem).

El procedimiento tendrá la forma siguiente:



En REGLAS colocar

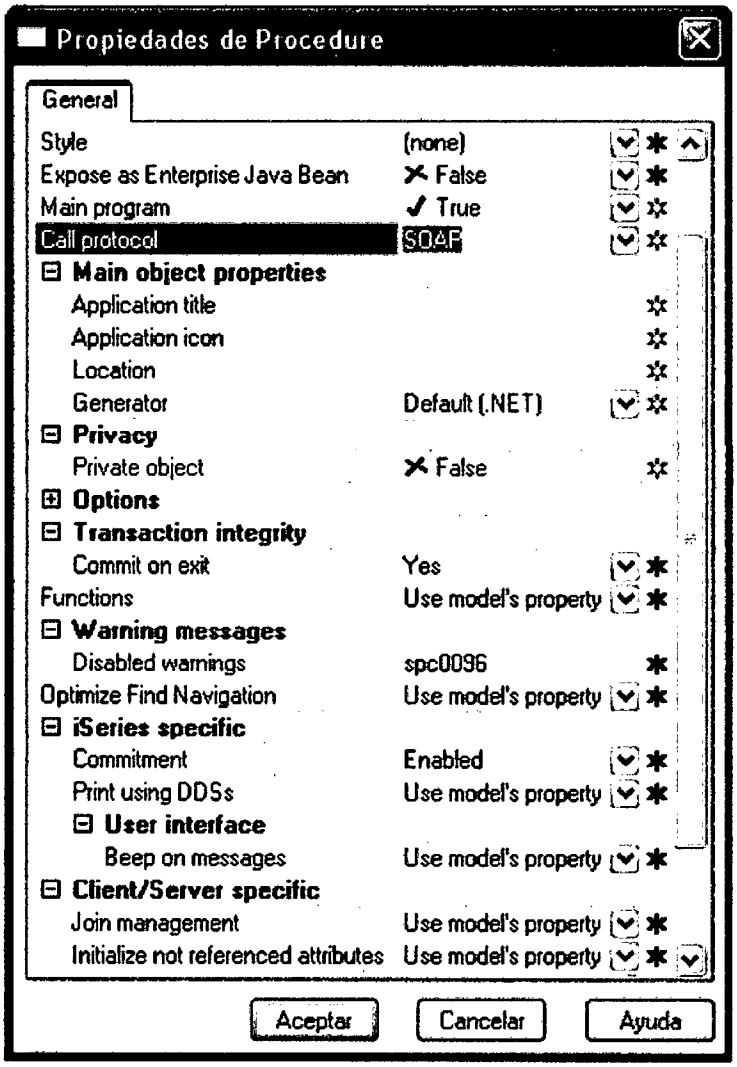
&PARM(&Articulos)

En las propiedades del objeto procedimiento Articulo configurar

MAIN PROGRAM = TRUE

CALL PROTOCOL = SOAP

Según la imagen de pantalla siguiente



Propiedades de Objeto Procedimiento Articulo

Después de generar y compilar el objeto procedimiento Genexus de nominado ARTICULO, verificaremos el Web Service generado automáticamente por Genexus, digitando la siguiente URL en el browser:

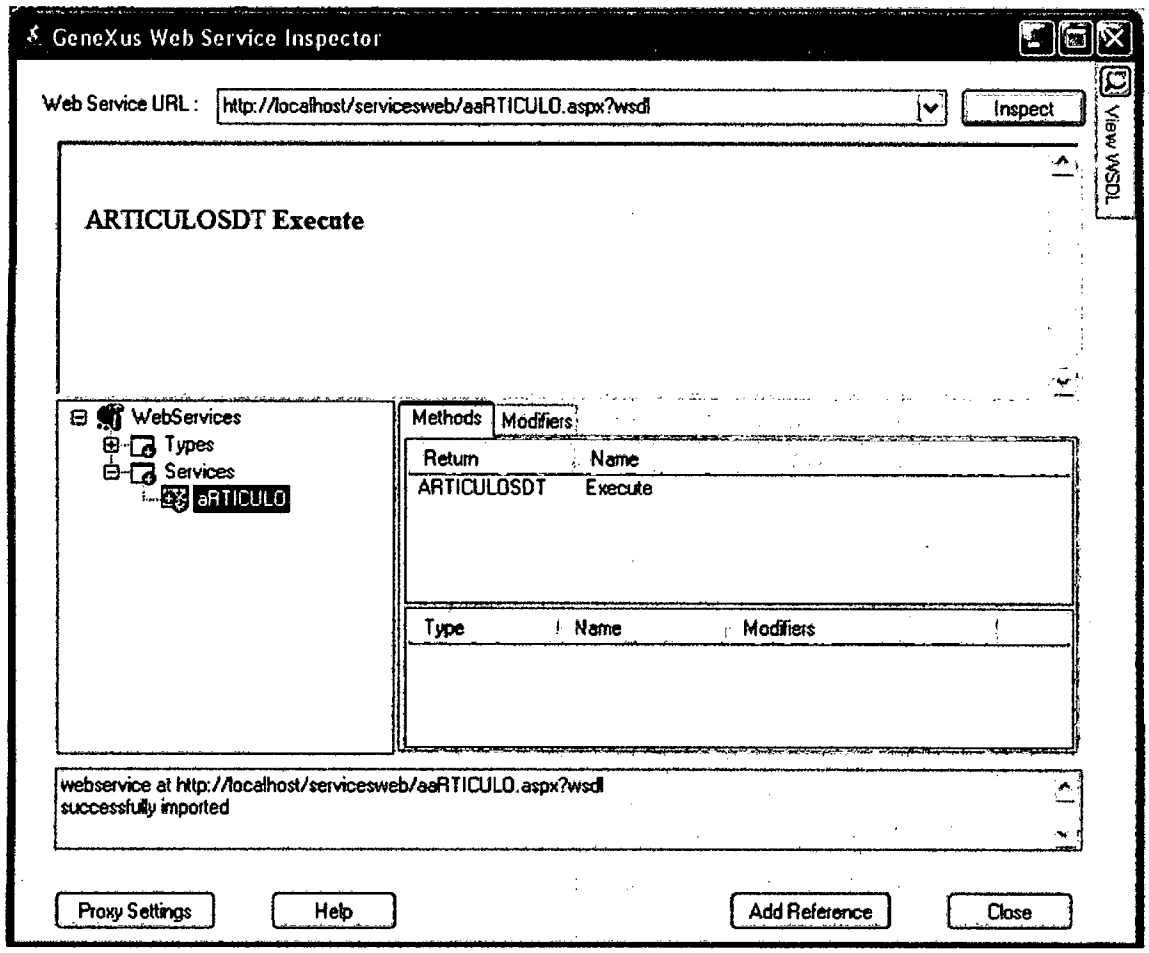
http://localhost/servicesweb/aArticulo.aspx?wsdl

CREACION DEL CONSUMIDOR DEL WEB SERVICE ARTICULO

Primero crearemos un objeto Genexus Web Panel que será el consumidor del Web Service "**Articulo**" recientemente creado.

En General un Web Service será externo a la Base de Conocimiento, por lo que vamos a crear una nueva Base de Conocimiento para crear allí ese Web Panel al que llamaremos **VerArticulos**. Tal como se describe a continuación.

1. Crearemos una nueva Base de Conocimiento de nombre "**ConsumidorArt**".
2. Desde **Diseño** (en Genexus) ejecutaremos el **WSL Inspector:** seleccionando la opción del menú **Tools/WSDL Inspector**, mostrándonos la pantalla siguiente:



En el campo "**Web Service URL**" colocaremos la siguiente URL http://localhost/servicesweb/aArticulo.aspx?wsdl

Y luego haremos click en botón **"Inspect"**, la pantalla nos muestra que el método del Web Service es "**Execute**" y el tipo de datos que devuelve es "**ArticuloSDT**".

Cuando se hace click en el botón "**Add Reference**" se definen los tipos de datos además de toda la información necesaria para poder consumir el Web Service.

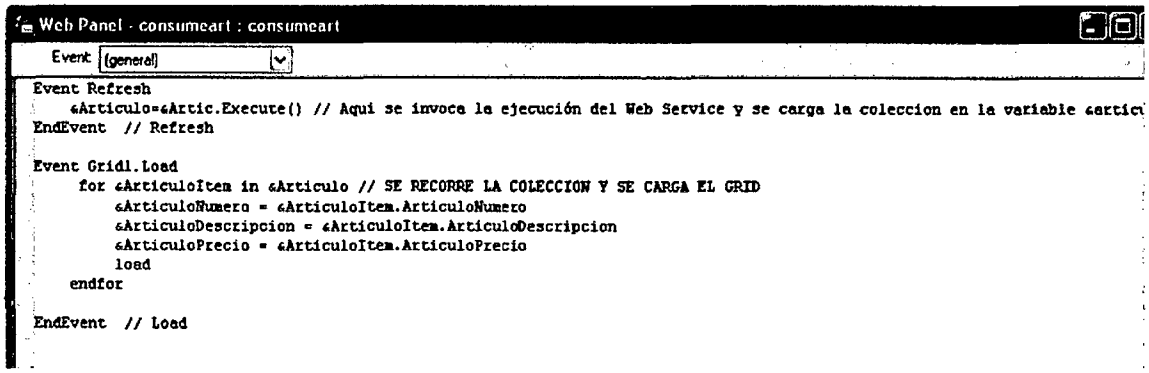
En Genexus pasamos a la etapa de prototipo forzando el impacto.

3. Crearemos el objeto Genexus Web Panel al que denominaremos **VerArticulos,** para eso definiremos las variables siguientes:

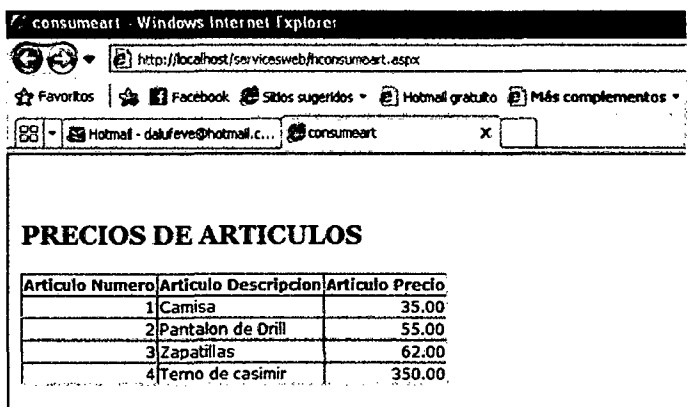
&Articulo y &ArticuloItem

Se observa que los tipos de datos **ArticuloSDT** (colección de Articulos), ArticuloSDT_ArticuloSDTItem (cada Item de la colección) y el Web Service **Articulos** se crearon automáticamente al inspeccionar el servicio con el **WSDL Inspector.**

El Form del Web Panel **VerArticulos** contendrá un **grid** donde mostrará las variables conteniendo la información de interés y se deberá de programar los eventos **Refresh** y **load** del Web Panel, como se muestra a continuación:



Finalmente salvamos, especificamos, generamos el programa, compilamos y ejecutamos el Web Service y obtenemos la siguiente pantalla mostrando la información siguiente:



Con Genexus también es posible consumir Web Service Externos (no generados por Genexus) en nuestras aplicaciones.

4.2.2.3.17. Aplicación de Patrones con Genexus

En esta sección aplicaremos el concepto de patrón o pattern a la Base de Conocimiento corporativa.

Como vimos en la sección 4.1.3.3.3. del capítulo IV, el patrón **Work With,** a partir de una transacción o para todas aquellas transacciones que deseemos, genera a todos los objetos necesarios para tener una aplicación web para cada una de las transacciones.

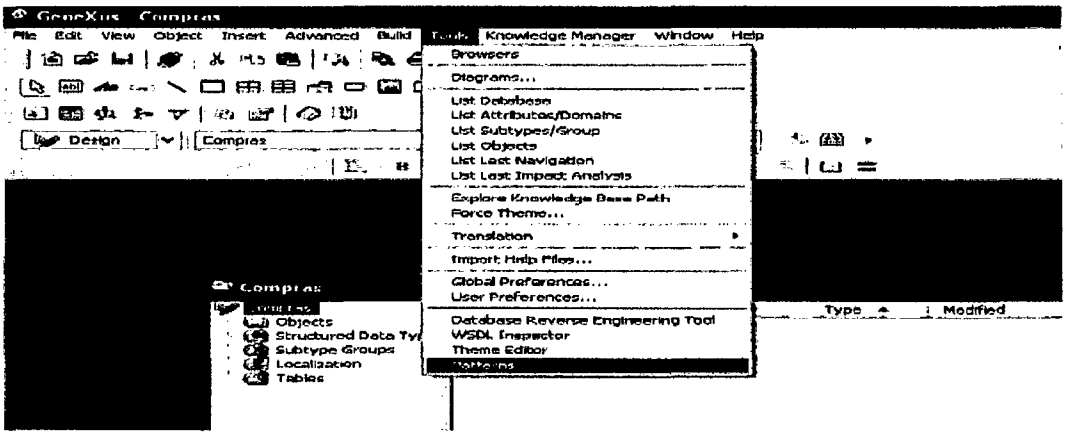
Por ejemplo, si se aplica el pattern **"Work With"** a la transacción "Analista", se generará un objeto GeneXus Web Panel, que permitirá al usuario final consultar interactivamente los Analistas de la Base de Datos (se presentará una página web conteniendo una lista con los Analistas de la tabla ANALISTA). Este Web Panel se llamará "Work With Analista" y ofrecerá entre otras cosas, un enlace asociado a cada Analista mostrado en la lista, para que mediante su selección se acceda a otro objeto Web Panel, de nombre "View Analista", que mostrará todos los datos del Analista y su información relacionada, como podrían ser Pedidos, etc. El Web Panel "Work With Analista" además, para cada Analista mostrado ofrecerá la posibilidad de modificar sus datos, agregándose automáticamente para ello una invocación a la transacción "Analista", así como la posibilidad de eliminar un registro de Analista, o de insertar un Analista nuevo, invocando para ello también a la transacción "Analista".

Se podrán configurar diversas propiedades para agregar opciones de filtrado en el Web Panel "Work With Analista", por ejemplo para que el usuario final pueda consultar solamente los Analistas de cierta oficina, o aquellos cuyos nombres se encuentren incluidos en determinado rango; y también configurando una simple propiedad, se podrá incluir en el Web Panel "Work With Analista" un combo box que ofrezca distintos ordenamientos posibles, para que el usuario final elija si desea el resultado de la consulta ordenado por nombre de Analista, por Número de identificación u otro atributo.

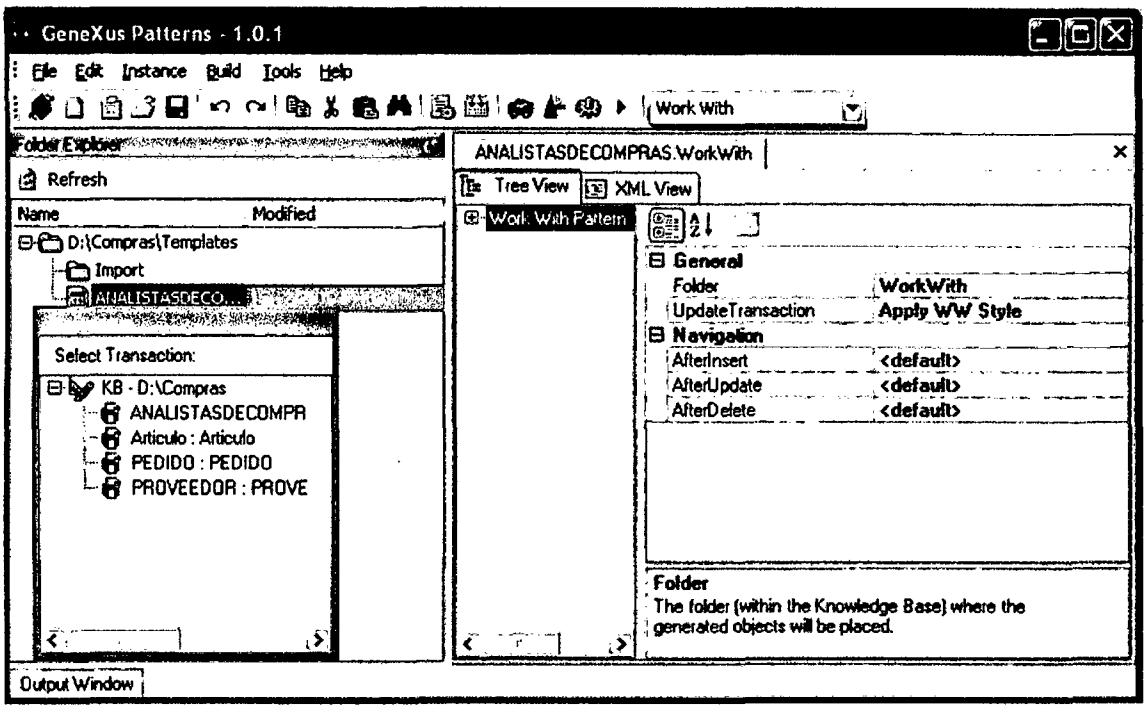
Existen 2 formas de aplicarlo, uno es desde el ambiente de Genexus y la otra forma es trabajando con la herramienta patterns, en forma independiente, ejecutando **GenexusPatterns.exe**, aquí trabajaremos con la primera forma, esto es, desde el ambiente de Genexus, para lo cual seguiremos los pasos siguientes:

1. Una vez que tengamos el prototipo web creado, nos ubicaremos en el modelo de diseño, seleccionamos y ejecutamos la opción del menú **Tools**

/ **Patterns**, ante esto, la Base de Conocimiento se cerrará automáticamente, y se reabrirá cuando se cierre la herramienta Patterns.



2. Se presenta la pantalla siguiente, mostrando las transacciones contenidas en la Base de Conocimiento en el **Tab KB Explorer**.



3. Seleccionamos en el **combo box** que se muestra en la barra de herramientas el tipo de patrón (**work with**) que queremos aplicar, en donde se ofrece por defecto los patrones predefinidos.
4. Obtenemos ún **Instance File** por cada caso, al cual queremos aplicar el patrón, un **Instance File** es un archivo **XML** conteniendo los datos propios de la instancia.

Teniendo una Transacción (o varias) seleccionada(s) presionar el **botón derecho / Generate Instance File.** Luego dar Doble clic en una Transacción

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

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

• **Instance files:** por cada situación o instancia a la cual se quiera aplicar el patrón, habrá que crear un **instance file,** con la información propia de esa instancia en particular tales como: atributos a mostrar, etc. Cada **instance file** es un archivo **XML** con los datos propios de la instancia, los patterns suelen proveer una funcionalidad para crear '**instance files**' por defecto, que posteriormente, de considerarse necesario, pueden modificarse fácilmente.

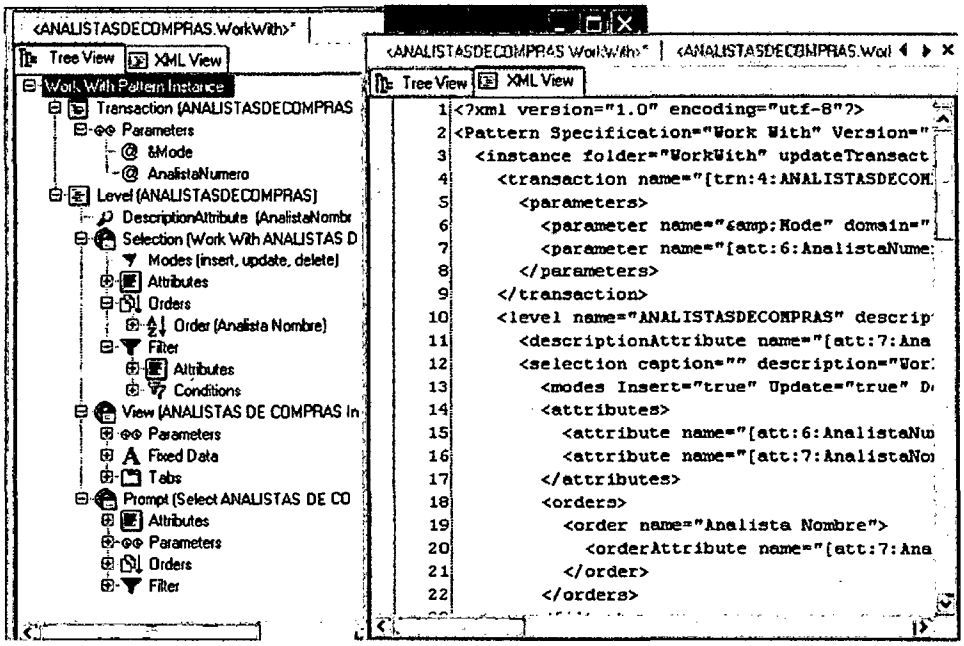
• **Template files:** contienen la implementación del patrón y de su aplicación a las instancias.

El resultado que se obtiene del proceso de instanciación de un patrón, procesando los **instance files** y **template files** es: un conjunto de objetos GeneXus que deberán ser consolidados en la Base de Conocimiento.

5. Cada '**instance file**' es un archivo **XML** con estructura jerárquica, conteniendo en cada uno de sus **nodos** un conjunto de propiedades.

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

   El editor **XML View** permite editar los **instance file** directamente en su formato **XML**. Por su parte el editor **Tree View** es mucho más amigable, sencillo de usar, y con interfaz en alto nivel que provee mayor funcionalidad para ayudar en el proceso de edición. Por todo esto el editor **Tree View** es el más usado y es el recomendado para usuarios no avanzados.



6. **Grabar los Instance Files**, para esto debemos usar:

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

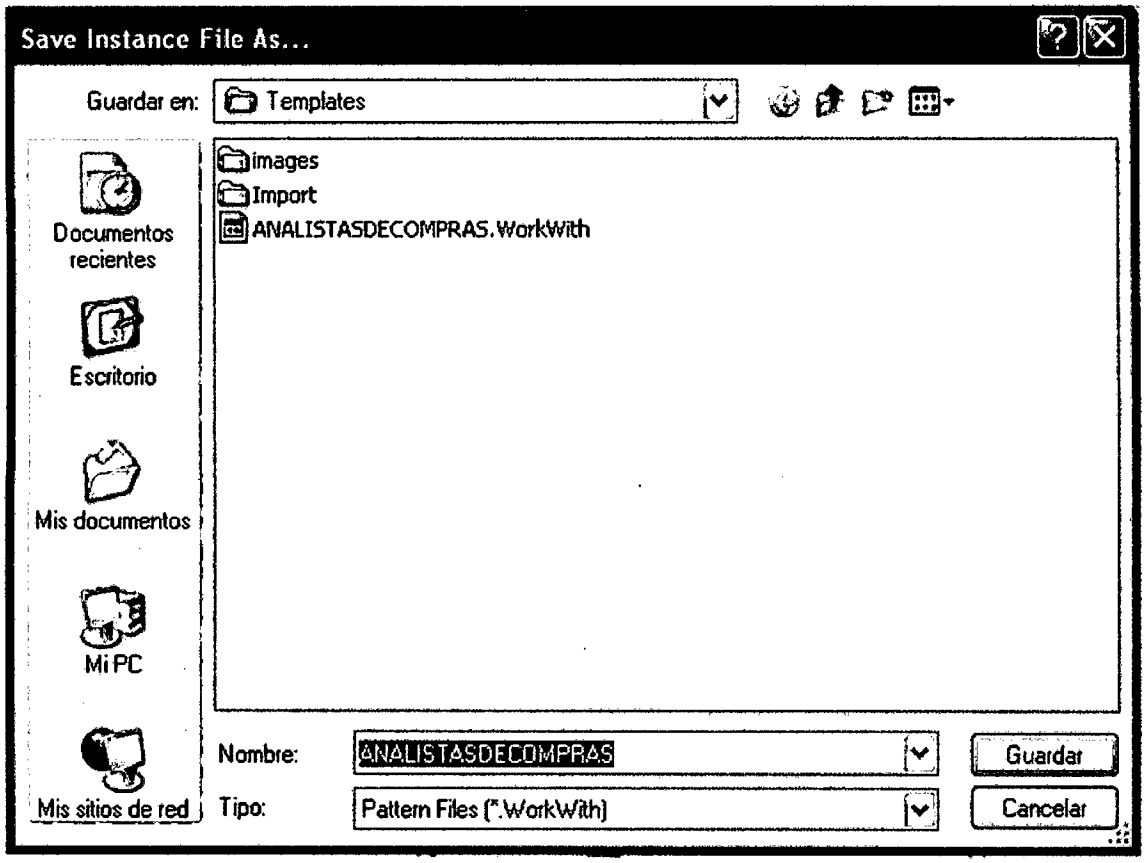
   Los 'instance files' que no se han salvado aún, se visualizan con nombre: **<Transaction Name.Pattern>***. Y una vez salvados se visualizan con el nombre: **Transaction Name.Pattern** (por ejemplo: Cliente.WorkWith).

   Los 'instance files' se almacenan en el subdirectorio **Templates** bajo el directorio de la Base de Conocimiento.

   Seleccionando el **Tab Folder Explorer** se pueden visualizar estos archivos, como se muestra en la pantalla siguiente:

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

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



7. Una vez creados y editados los **'instance files'**, el siguiente paso es **que la herramienta genere los objetos GeneXus que implementan el pattern para las instancias.**

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

Se genera en **KBPath\Templates\Import**, un archivo **<Transaction Name>.xpz.xml** por cada **'instance file'**

Existe la posibilidad de que se consoliden automáticamente a continuación o que lo haga después, desde la Base de Conocimiento, el desarrollador GeneXus.

También es posible seleccionar una transacción o grupo de transacciones estando posicionado en el **tab KB Explorer**, y mediante botón derecho ejecutar la opción **Generate, Apply** and **Consolidate**. Pero es

importante entender que seleccionando esta opción, se generarán los **instance files** por defecto

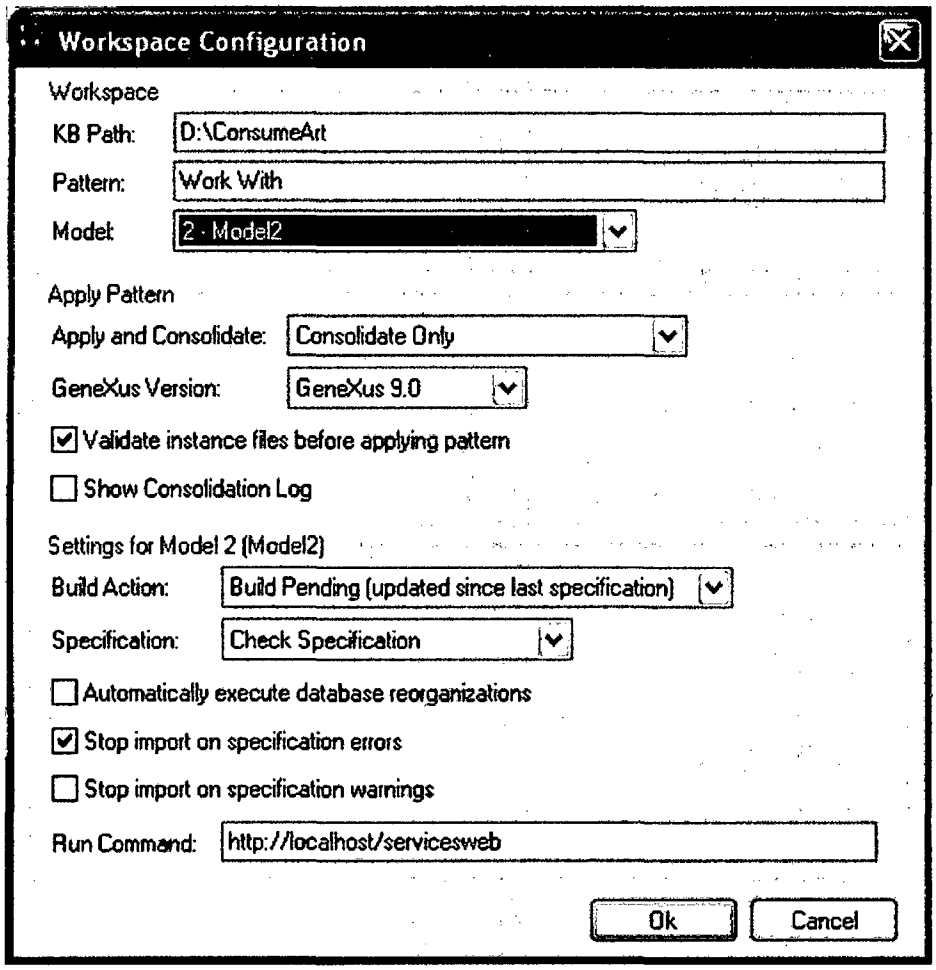
nuevamente para las transacciones seleccionadas, y luego de ello, se generarán los archivos **<Transaction Name>.xpz.xml** correspondientes y se consolidarán en la Base de Conocimiento. Si se está seguro que los **instance files** por defecto no necesitan ser editados, es posible seleccionar esta opción directamente.

A su vez es importante saber que si se selecciona la opción **Apply** and **Consolidate** se efectuarán también las siguientes acciones:

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

8. Desde la herramienta **Patterns**, es posible ejecutar las acciones que se realizan también desde GeneXus: Impactar el modelo, Especificar, Generar, Compilar y Ejecutar.

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



Las Opciones disponibles en el diálogo son: (ver pantalla siguiente)

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

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

Consolidate Only (Apply and Consolidate)

- Impact Model (Apply and Consolidate + Impact Model)
- Impact, Specify (Apply and Consolidate + Impact + Specify)
- Impact, Specify, Compile (Apply and Consolidate + Impact + Specify + Compile)
- Impact, Specify, Compile, Run (Apply and Consolidate + Impact + Specify + Compile + Run)

GeneXus Version - La versión de GeneXus correspondiente a la Base de Conocimiento se detecta automáticamente (puede ser 8.0 o 9.0). El modelo se **especificará / generará** con dicha versión.

Build Actions- Permite seleccionar qué objetos deben especificarse y generarse al seleccionar **Specify and Generate**. Las opciones disponibles son:

- Build All

- Build Pending (updated since last specification)

– Specify Consolidated Objects

Specification – Permite seleccionar el tipo de especificación / generación a ser ejecutado al seleccionar **Specify and Generate**. Las opciones disponibles son:

– Full Specification

– Check Specification

– Force Generation

Run Command – En esta opción se debe indicar la URL que se ejecutará si se seleccionó la opción Impact, Specify, Compile, Run.

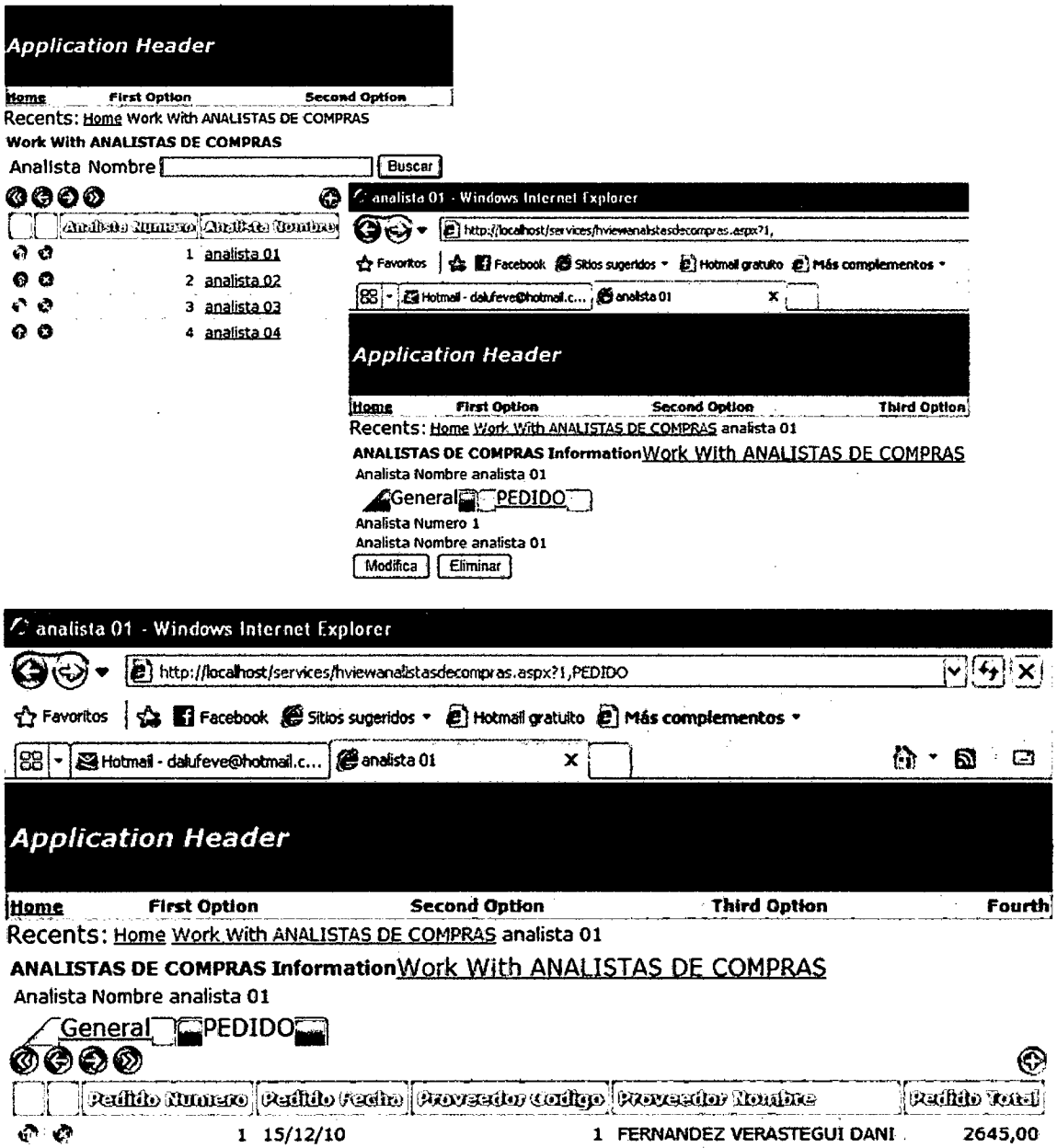
Se debe configurar asi: para aplicaciones .NET ... Run Command = **http://localhost/services/hhome.aspx**

para aplicaciones Java ... o **http://localhost:8080/servlet/hhome**

Nota: Es conveniente aclarar que **"home"** es el nombre de un web panel generado por el patrón WorkWith; el mismo ofrece enlaces a todos los web panels WorkWith generados. La letra **h** que antecede a su nombre en la invocación, corresponde al prefijo que se agrega a todo objeto web panel main o ejecutable que se invoca.

En las pantallas que se muestran a continuación, se muestra el resultado de la ejecución.

RESULTADO DE LA EJECUCION DE PATTERN O PATRON WORKWITH DE GENEXUS.



4.2.2.3.18. Desarrollo Multiplataforma.

Ahora que ya ha generado y ejecutado una aplicación en un ambiente o plataforma (.NET), para generar y ejecutar la misma aplicación en otros ambientes solo hay que definir un nuevo modelo de Prototipo o Producción.

La herramienta Genexus Trial Versión ofrece los generadores Genexus .NET y Genexus Java. No obstante, la versión completa de Genexus suporta las plataformas líderes del mercado. Visite

http://www.genexus.com/technologies para obtener una lista completa de las tecnologías o plataformas soportadas por Genexus.

4.2.2.4. FASE DE PRODUCCION

El objetivo principal de esta fase de la MDDS-GX es implementar la Base de datos y los programas de aplicación definidos en la fase de diseño en una plataforma igual a la que será puesta en producción en un sistema de software que satisfaga las necesidades del usuario. Asimismo se toma en cuenta a los requisitos no funcionales que se han establecido para el sistema de software.

-Un modelo de Producción, se utiliza en la etapa de puesta en producción, cuando el prototipo fue aprobado y la aplicación o los cambios efectuados ya pueden ser llevados al cliente.

Una vez que el modelo de prototipo ha sido aprobado, es momento de "pasar a producción", creando un modelo de Producción. Este modelo tendrá como plataforma la del cliente, dado que los programas que se obtengan de aquí (generados automáticamente por Genexus) son los que se llevarán al cliente. Al crear este modelo, como con todo modelo de Prototipo o de Producción, se crearán las tablas en la Base de Datos asociada. Genexus genera un programa de creación de tablas en el lenguaje de programación asociado al modelo, éste programa se compila, creándose un ejecutable que luego es corrido obteniéndose la creación de las tablas.

Cuando una aplicación desarrollada en Genexus ha sido puesta en producción, necesariamente deberá haber al menos 3 modelos definidos en la base de conocimiento:

- El modelo de Diseño, el cual se crea automáticamente al crear la base de conocimiento,

- El modelo de Prototipo, utilizado para ir desarrollando y probando la aplicación,

- El modelo de Producción, pues es necesario tener una imagen fiel de la aplicación que se lleva al cliente, en la plataforma de producción.

Otra vez las pruebas son una actividad importante en curso en esta fase. La cual se divide en cuatro sub- fases:

- Identificar el modelo de producción a Prototipar: Identificar los requerimientos funcionales y no funcionales que deben ser considerados en el sistema de software a prototipar.

- Concordar el cronograma: De acuerdo en cómo y cuándo realizar estos requisitos.

- Crear prototipo del modelo de producción: Crear un sistema de software que con seguridad puede ser entregado a los usuarios finales para el uso diario. Investigan, perfeccionar y consolidar el prototipo de la iteración actual en proceso de creación de prototipos son también importantes en esta sub-fase.

- Revisión del prototipo del modelo de producción: Comprobar la exactitud del sistema diseñado. Una vez más las pruebas y el examen son las principales técnicas utilizadas, los registros de prueba y la retroalimentación del usuario.

Los resultados de esta etapa son un prototipo del modelo de producción, durante esta sub-fase los usuarios finales ponen a prueba y al final de la iteración y luego de construir el sistema de software probado se entrega a la siguiente fase. En esta etapa, el sistema de software está construido y las diferentes iteraciones están consolidadas en la Base de conocimiento corporativa y listo para pasar a la siguiente fase, la implementación.

4.2.2.5. FASE DE IMPLEMENTACION

En la etapa de implementación, se prueba el sistema de software, se entrega a los usuarios y se lleva a cabo el entrenamiento de los futuros usuarios. El sistema de software que se entrega ha sido revisado para incorporar los requerimientos establecidos en las etapas iniciales del proyecto. La etapa de implementación se puede subdividir en cuatro sub- etapas:

- Aprobación y guías del usuario: Los usuarios finales aprueban el sistema de software para su implementación y se crean las guías de uso del sistema de software.

- Formar a los usuarios: Se entrena a los usuarios en el uso del sistema de software.

- Poner en práctica: Implementar el sistema de software ya probado en las instalaciones de los usuarios finales.

- Revisión del negocio: Examinar el impacto del sistema implementado en el negocio, una cuestión central será si el sistema cumple los objetivos establecidos al comienzo del proyecto. En función de este, se pasa a la fase siguiente, el post-proyecto o bucles de vuelta a una de las fases anteriores de desarrollo futuro.

Eventualmente la fase puede llegar a iterarse. A partir de aquí hay cuatro cursos de acción posibles: (1) Si el software satisface todos los requerimientos, el desarrollo ha terminado. (2) Si aún existen requerimientos por sin atender, se debe correr el proceso nuevamente desde el inicio. (3) Si se ha dejado de atender algún aspecto relacionado con el diseño de la Bases de Datos, el proceso se debe correr desde la fase de modelo de diseño en adelante. (4) Si se identificó algunas cuestiones relacionadas con la plataforma, el proceso puede correrse desde la fase de modelo de prototipo o producción.

Los resultados de esta etapa son un sistema entregado e instalado, listo para su uso por los usuarios finales entrenados.

4.2.3. ROLES REPRESENTADOS EN MDDS-GX

Existen roles incorporados a la metodología MDDS-GX. Es importante que los miembros del proyecto deban designarse para cumplir diferentes roles antes de que comience a ejecutarse el proyecto. Cada rol tiene su propia responsabilidad. Los roles son:

- **Promotor Ejecutivo:** Es un rol muy importante que debe cumplir un usuario de la organización que tenga la capacidad y la responsabilidad de comprometer fondos y los recursos adecuados. Este rol tiene un gran poder para la toma de decisiones.

- **Visionario:** Es el que tiene la responsabilidad de iniciar el proyecto, asegurando que los principales recursos se encuentren desde el inicio. El Visionario tiene la percepción más precisa de los objetivos del negocio del sistema y del proyecto. Otra tarea es supervisar y mantener el proceso de desarrollo en el buen camino.

- **Usuario experto:** Trae el conocimiento de la comunidad de usuarios en el proyecto, asegura que los desarrolladores reciban suficiente cantidad de

retroalimentación del usuario durante el proceso de desarrollo, participa en la evaluación y aprobación de los prototipos.

- **Asesor del usuario:** Puede ser cualquier usuario que representa un punto de vista importante y aporta el conocimiento diario del proyecto.
- **Administrador del Proyecto:** Puede ser cualquier persona de la comunidad de usuarios o del personal de Tecnologías de Información que gestione el proyecto en general.
- **Coordinador Técnico** Responsable del diseño de la arquitectura del sistema y el control de la calidad técnica del proyecto.
- **Jefe de equipo** Lidera a su equipo y asegura que el equipo funciona de manera efectiva en su conjunto.
- **Analista Genexus:** Interpretar los requisitos funcionales y no funcionales del sistema y los modela haciendo uso de la herramienta CASE Genexus, construye y participa conjuntamente con el usuario en las pruebas de los prototipos.
- **Probador** Usuario que es responsable de evaluar y aprobar los prototipos Comprueba la corrección de extensiones técnicas mediante la realización de pruebas. El Probador tendrá que proporcionar comentarios y documentación.
- **Documentador:** Es responsable de recopilación y registro de los requisitos, acuerdos y decisiones tomadas en cada taller.
- **Facilitador**: Es responsable en la gestión del progreso de los talleres, actúa como un motor para su preparación y comunicación.
- **Roles de Especialistas:** Arquitecto de Negocios, Responsable de Calidad, Integradores de Sistemas, etc.

4.2.4. NATURALEZA ITERATIVA E INCREMENTAL DE MDDS-GX.

Además de llevar a cabo la priorización de los requerimientos, la MDDS-GX también proporciona un enfoque de desarrollo iterativo e incremental.

En el Modelo de diseño, de prototipo, de producción y la fase de implementación sus sub fases pueden iterarse varias veces antes de entrar en la siguiente fase. Cada iteración aborda una serie de nuevas funcionalidades,

incorporando nuevos objetos, y cada iteración se basa en el trabajo anterior. Cada iteración se puede deshacer, si es necesario.

En función a los requerimientos identificados al momento de estar elaborando los prototipos el desarrollador puede retornar a cualquier fase, inclusive a la fase de estudio del negocio. El proyecto debe proceder a la fase posterior sólo cuando cumpla todos los requisitos definidos por el proyecto y los objetivos de negocio.

Debido a la naturaleza iterativa de MDDS-GX, es esencial para mantener una buena gestión de requisitos y de la configuración durante todo el proyecto. Esto asegura que se incorporen al proyecto todos los requerimientos que se identificaron a inicios del proyecto e inclusive aquellos identificados durante el desarrollo del mismo.

4.2.5. FACTORES CRITICOS DE ÉXITO DE LA MDDS-GX.

En la metodología MDDS-GX existe una serie de factores que son de gran importancia para garantizar el éxito de los proyectos, tales como:

1. En primer lugar está la aceptación de MDDS-GX por los altos directivos y empleados de la empresa. Esto asegura que los diferentes actores del proyecto estén motivados desde el principio y seguir participando en el proyecto.

2. El segundo factor se deriva directamente del anterior y es el compromiso de la dirección para garantizar la participación del usuario final. El enfoque de prototipos requiere una fuerte involucración y dedicación por parte del usuario final para evaluar y dar su aprobación a los prototipos.

3. Luego está el equipo del proyecto. Este equipo tiene que estar compuesto por usuarios expertos y desarrolladores Genexus de experiencia, que conformen un grupo estable, donde no haya mucha rotación de usuarios. Una cuestión importante es la potenciación del equipo del proyecto. Esto significa que uno o más de los miembros del equipo ha de poseer el poder y la posibilidad de tomar decisiones importantes en relación con el proyecto sin tener que escribir solicitudes formales a la alta dirección, que pueden consumir mucho tiempo. Para que el equipo del proyecto para sea capaz de ejecutar un proyecto de éxito, también necesitan la tecnología adecuada para llevar a cabo el proyecto.

CAPITULO V

ESTUDIO DE CASOS

INTRODUCCION

En el presente capítulo se presenta la validación del trabajo de investigación realizado en esta Tesis, que se ha abordado aplicando el método de investigación en acción descrito anteriormente, se describe el sistema de software que se ha desarrollado como caso de estudio; se ofrece una visión global del método de desarrollo presentado en esta Tesis, resaltando temas que están más estrechamente relacionados con el Desarrollo del Software.

Luego de aplicar la Metodología Propuesta MDDS-GX, la Fase Estudio del Sistema de Software a Desarrollar y las Sub Fases Estudio de Factibilidad y Análisis del negocio, la primera de ellas tienen por objeto, obtener un Informe de factibilidad, un Esquema de Plan global del proyecto de desarrollo de Software y un registro de los riesgos más importantes identificados, en la Sub Fase de Estudio del negocio, el objeto es obtener una lista de los requerimientos, priorizados en orden de importancia, una definición de la Arquitectura global inicial del sistema de software en fase de desarrollo, se determinan los objetos Genexus transacciones, las cuales provienen de los sustantivos identificados en las visiones de la realidad proporcionados por los usuarios expertos en el negocio, estos objetos conformaran la base de Conocimiento.

En lo referente al desarrollo de software haciendo uso de la herramienta CASE Genexus, hemos usado los manuales de Genexus [Artech Consultores S.R.L., Marzo 2006], [Artech Consultores S.R.L., Marzo 2007],[Artech Consultores S.R.L., Diciembre 2006] y [Márquez Lisboa Daniel, 2006].

5.1. APLICACION INFORMATICA DE FACTURACION DE UNA TIENDA DE VENTA DE EQUIPOS DE COMPUTO.

Con la finalidad de presentar los conceptos anteriores, a efectos de no realizar una presentación demasiado abstracta de la Metodología MDDS-GX y de la Herramienta Genexus y a manera de ejemplificar su uso, desarrollaremos una pequeña Aplicación Web la cual estará en Plataforma .NET con SQL Server 2005 Express. Se hará uso de la versión 9.0 de Genexus.

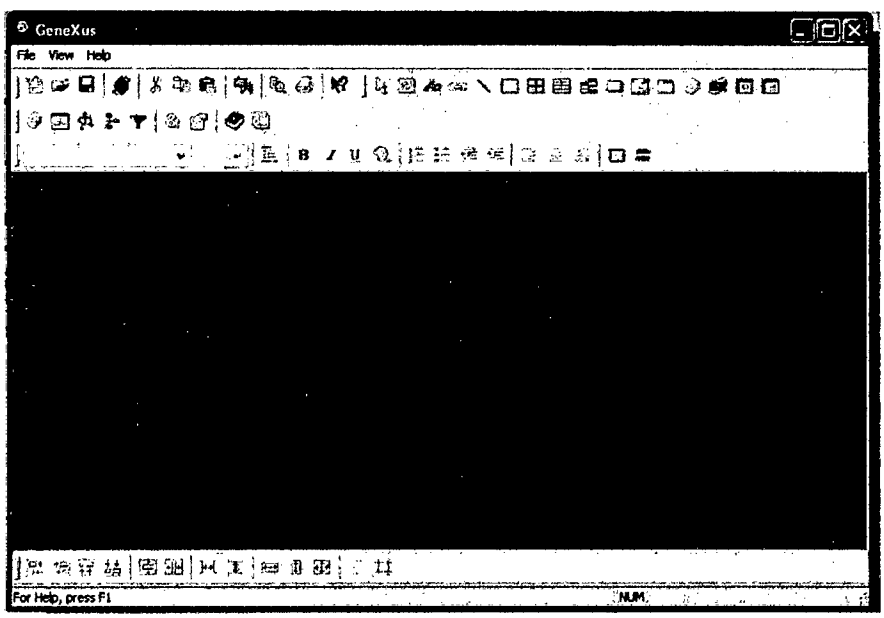
Los Detalles de su Instalación de Genexus versión 9.0, se puede hacer desde:

http://ftpusa.artech.com.uy/files/1288959459/manual%20de%20instalacion%20genexus%2090.pdf

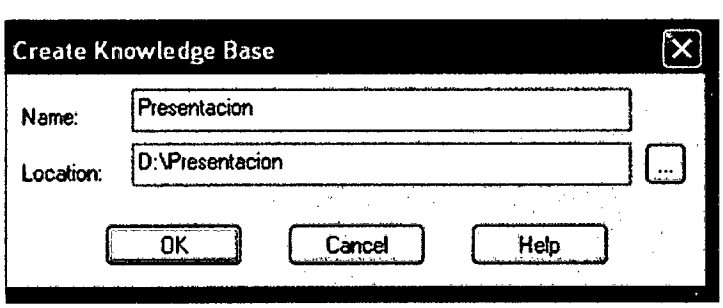
5.1.1. FASE DE DISEÑO: DISEÑO DE LA APLICACIÓN BASADO EN EL CONOCIMIENTO.

5.1.1.1. Creación de una Base de Conocimiento.

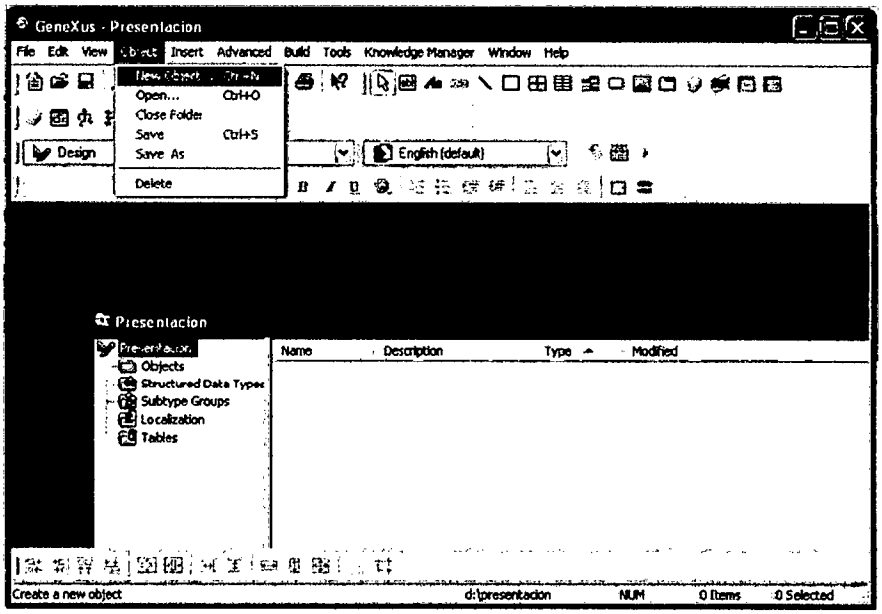
1. Ejecute Genexus. Se presentará la pantalla siguiente:

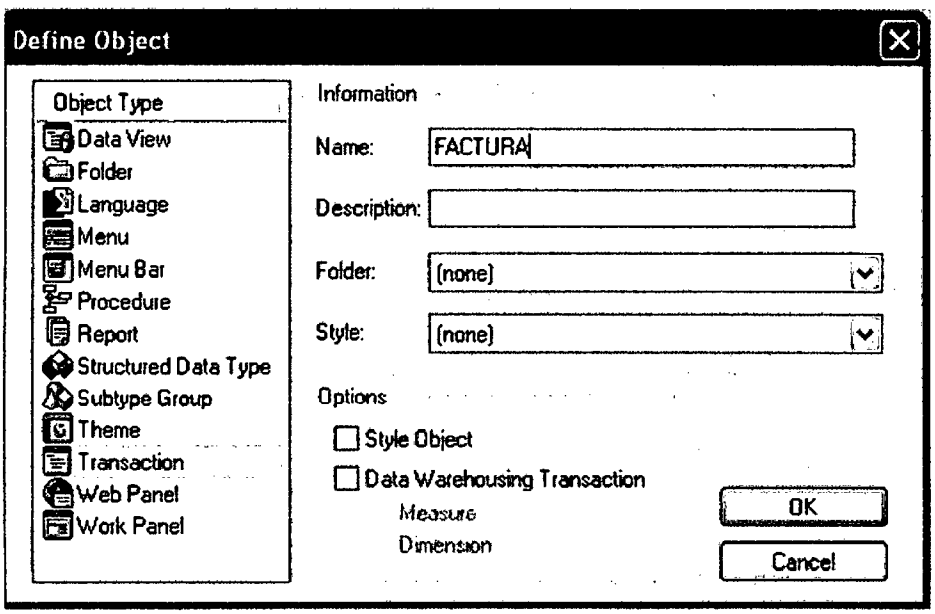


2. En el menú **file,** seleccione y haga clic en **New Knowledge Base**
3. Digite un Nombre a la base de Conocimiento: Por ejemplo "Presentación", luego presione clic en **OK** para continuar.

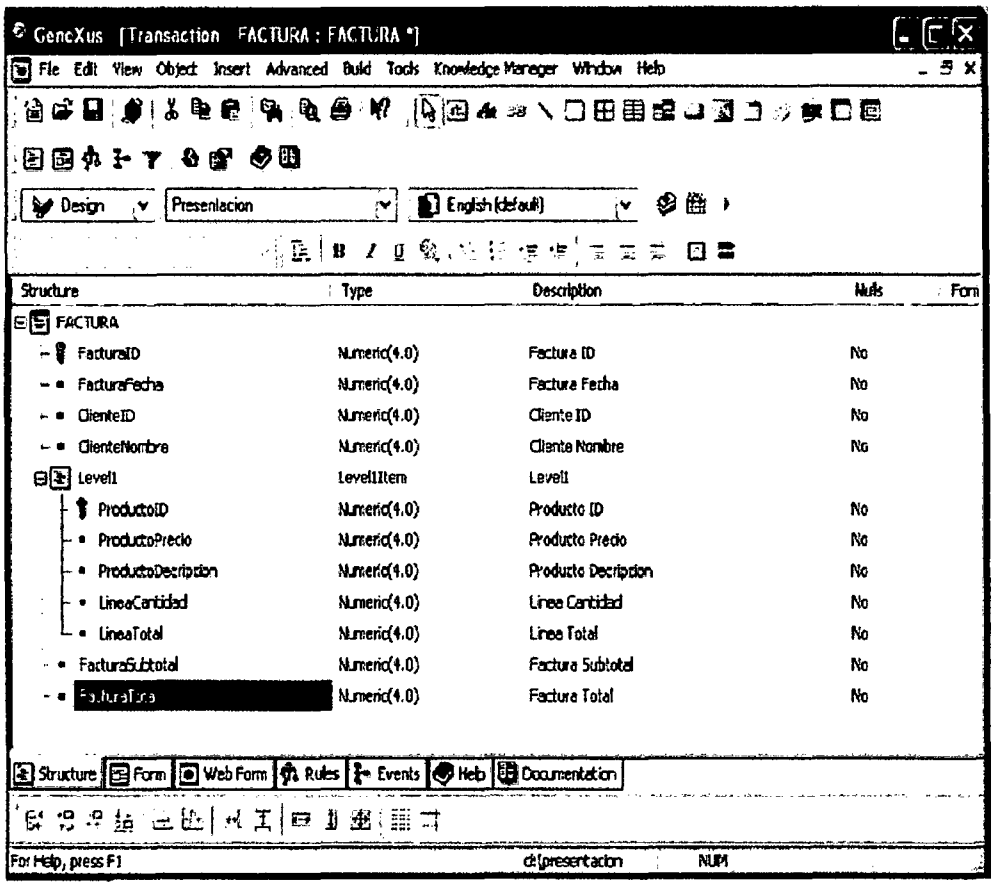


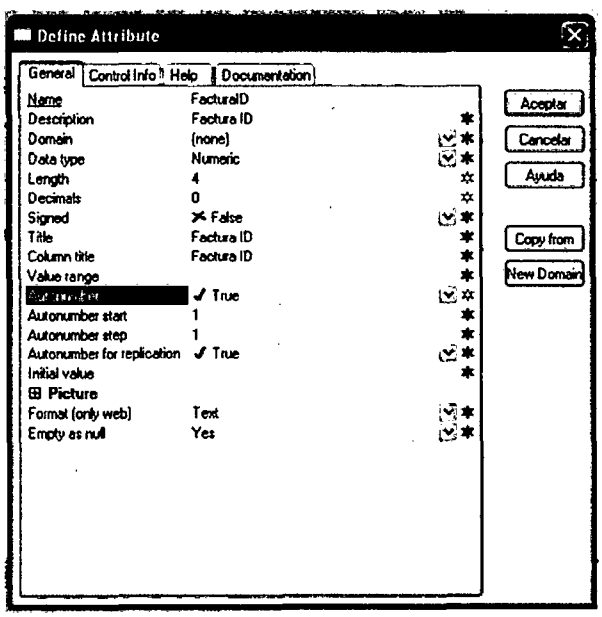
5.1.1.1.1. Creación de un Objeto Transacción: FACTURA





5.1.1.1.2. Descripción de la Estructura de la Transacción (structure):





5.1.1.1.3. Definición de campos calculados: Fórmulas.

Ahora definiremos los siguientes atributos fórmula:

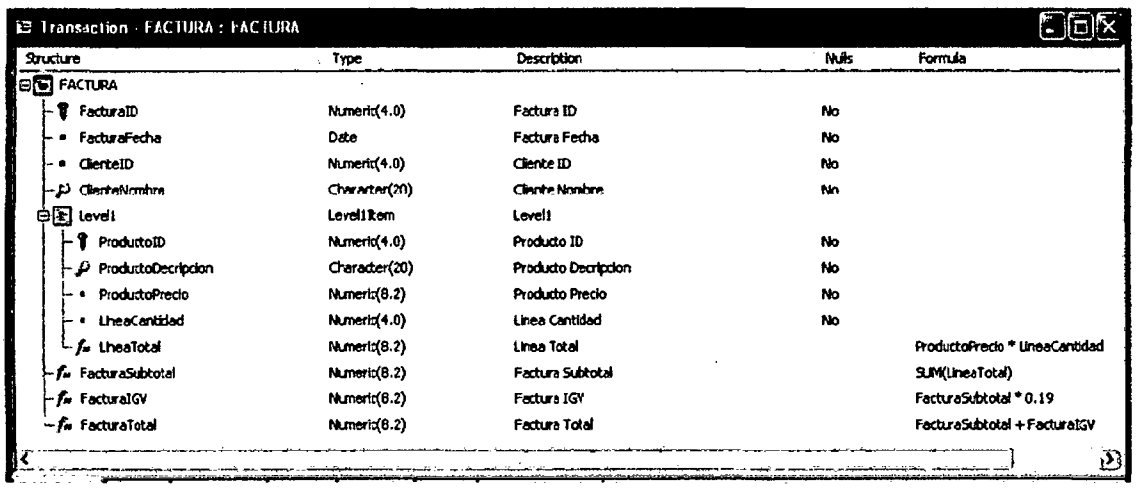
LineaTotal = ProductoPrecio * LineaCantidad

FacturaSubtotal = SUM(LineaTotal)

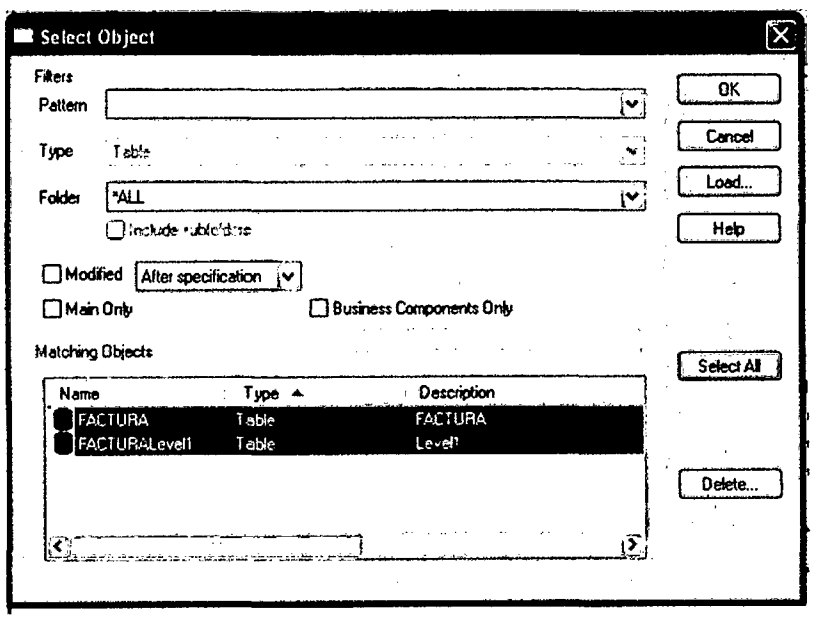
FacturaIGV = FacturaSubtotal * .19

FacturaTotal = FacturaSubtotal + FacturaIGV

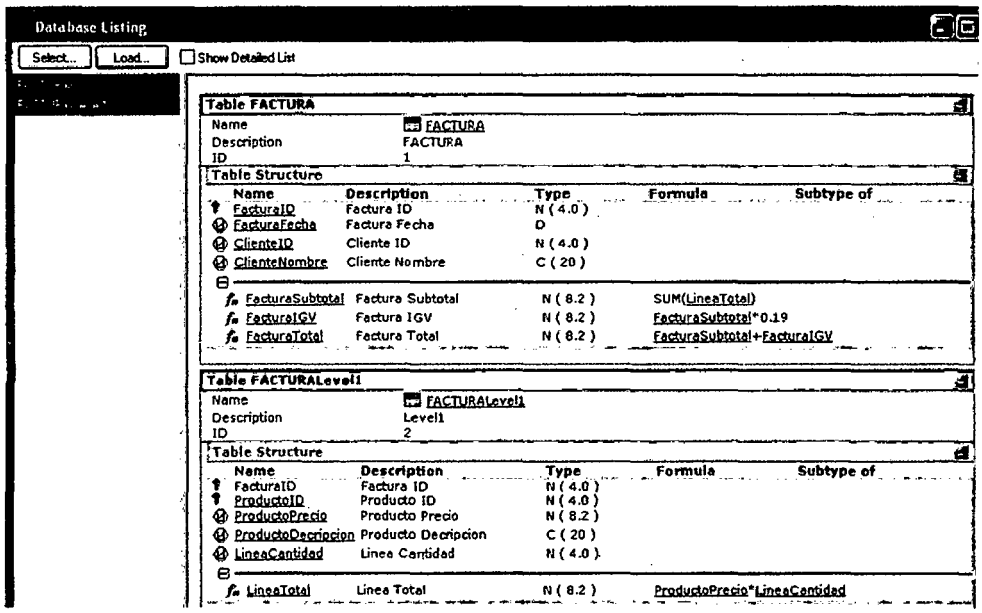
Luego tendremos una pantalla como la siguiente:



5.1.1.1.4. Visualización del Modelo de datos Inferido por Genexus

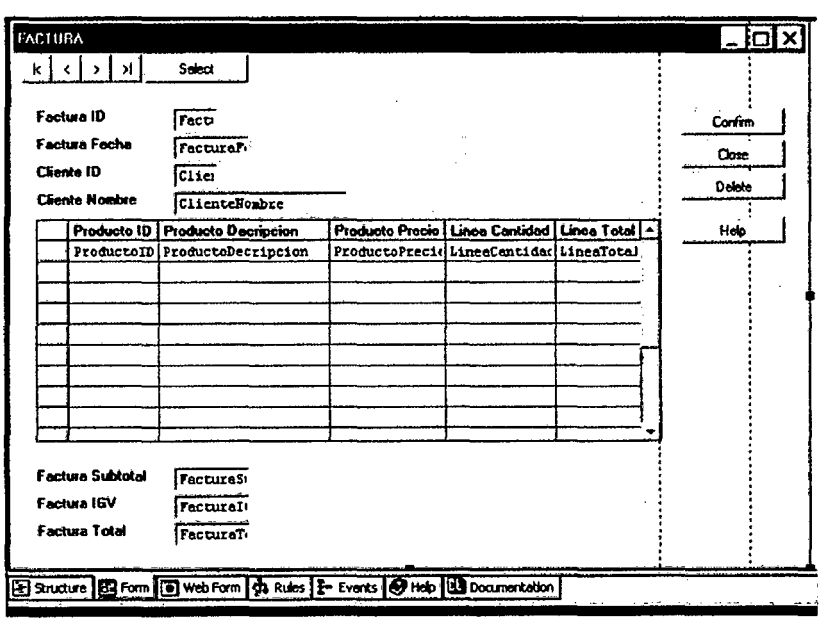


Al dar clic en Ok, se presentara la pantalla siguiente, mostrando el listado de la Base de Datos (del Modelo de Datos) generada por Genexus.

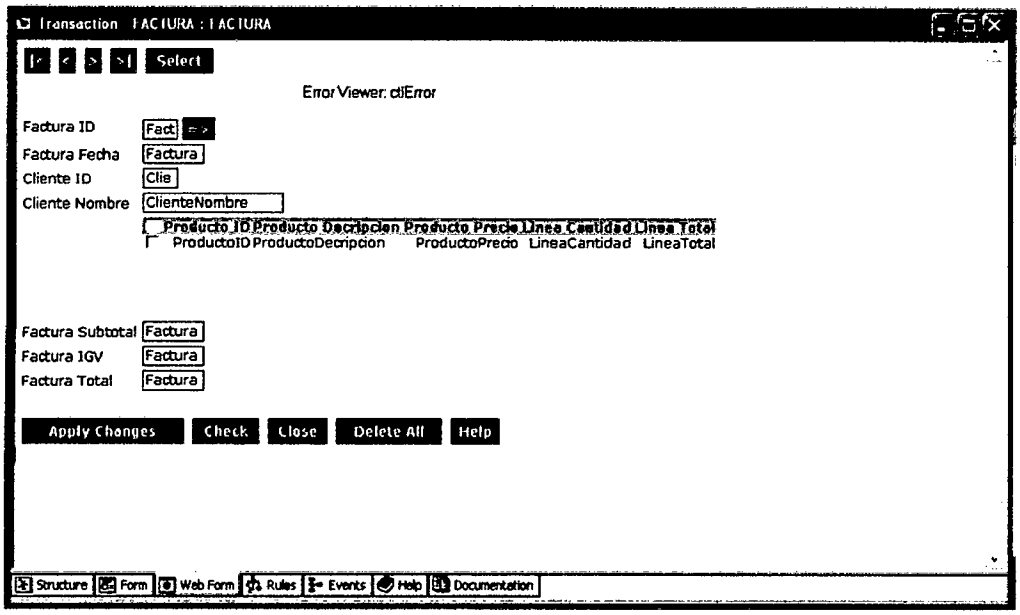


5.1.1.1.5. Visualización de los Formularios (Forms) del Objeto Transacción

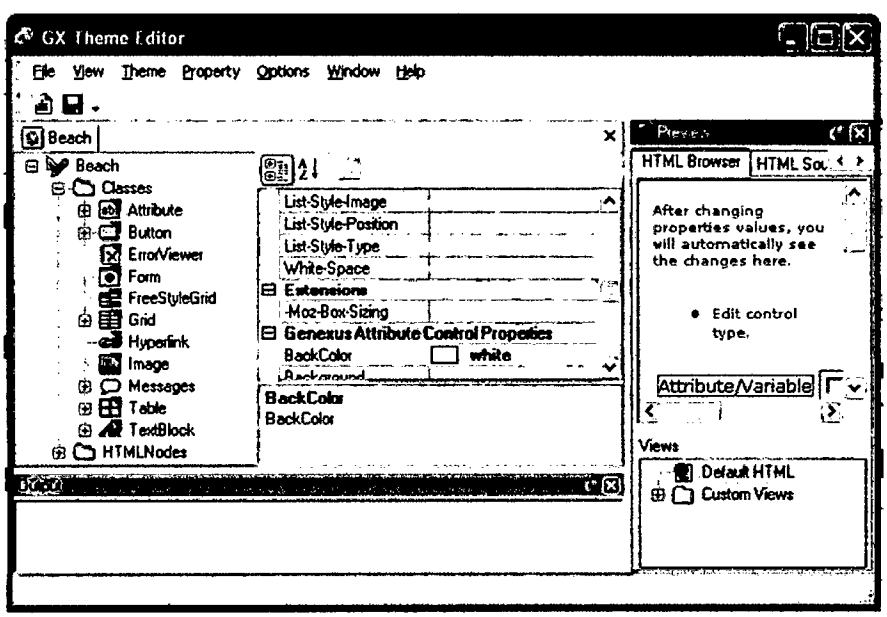
Fomulario Windows generado automaticamnet

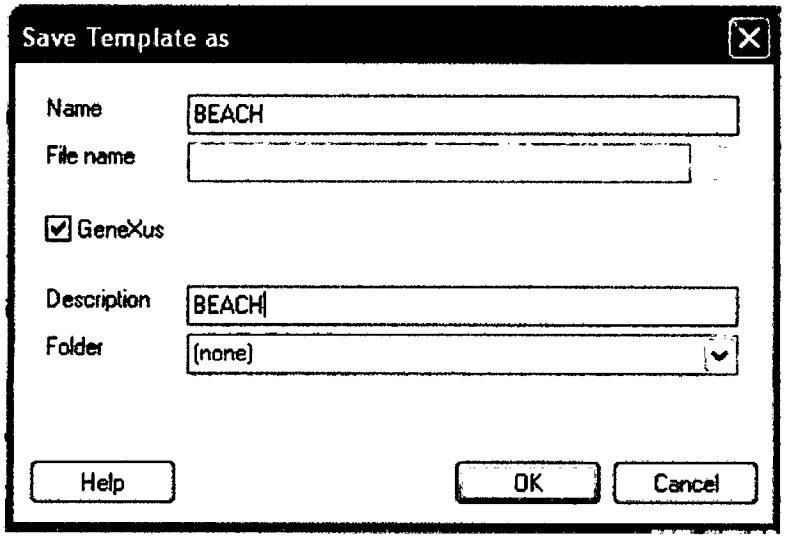


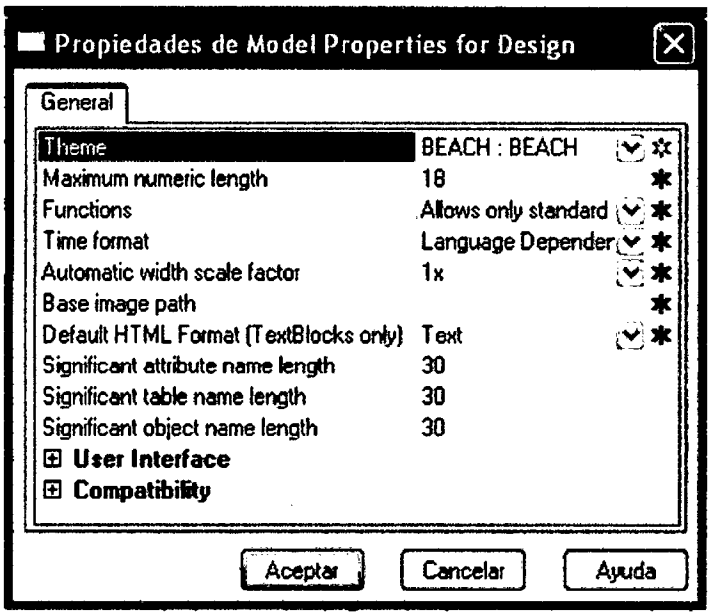
Formulario web generado automaticamente

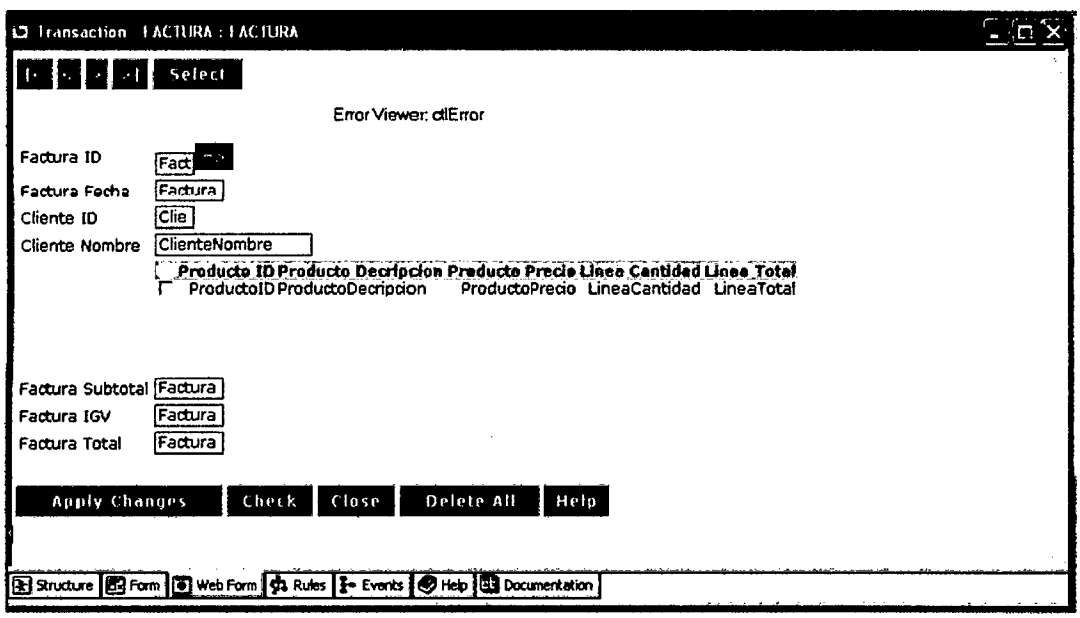


5.1.1.1.6. Creación de Formularios atrayentes: Temas

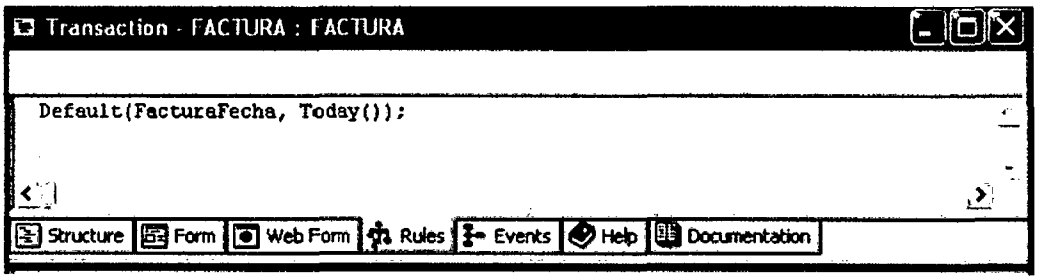



Save Template as
Name
BEACH
File name
GeneXus
Description
BEACH
Folder
(none)
Help
OK
Cancel


Propiedades de Model Properties for Design
General
Theme
BEACH : BEACH
Maximum numeric length
18
Functions
Allows only standard
Time format
Language Depender
Automatic width scale factor
1x
Base image path
Default HTML Format (TextBlocks only)
Text
Significant attribute name length
30
Significant table name length
30
Significant object name length
30
User Interface
Compatibility
Aceptar
Cancelar
Ayuda

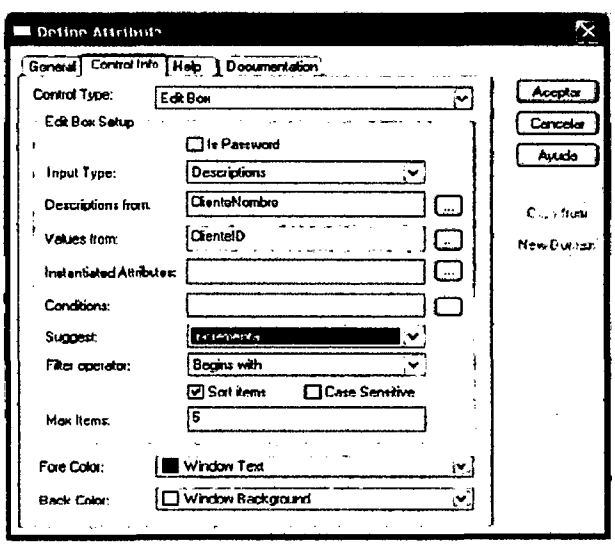

Transaction FACTURA : FACTURA
Select
Error Viewer: dtlError
Factura ID
Factura Fecha
Cliente ID
Cliente Nombre
ClienteNombre
Producto ID Producto Decripcion Producto Precio Linea Cantidad Linea Total
ProductoID ProductoDecripcion ProductoPrecio LineaCantidad LineaTotal
Factura Subtotal
Factura IGV
Factura Total
Apply Changes
Check
Close
Delete All
Help
Structure
Form
Web Form
Rules
Events
Help
Documentation

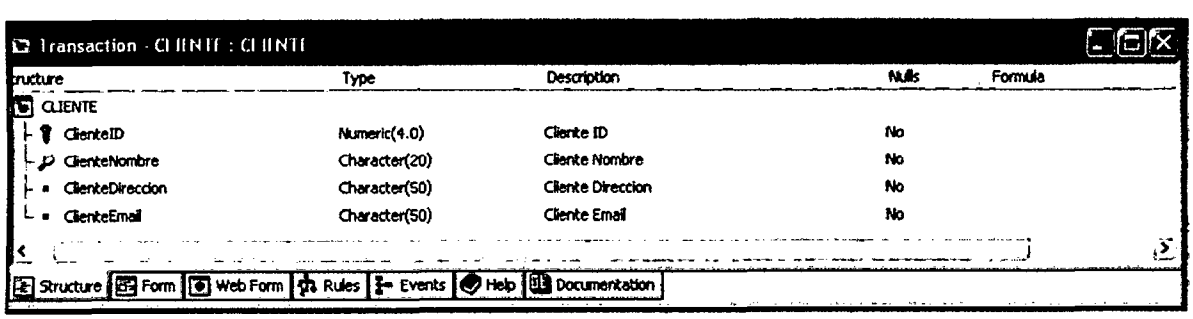
5.1.1.1.7. Agregar Reglas del Negocio: Reglas

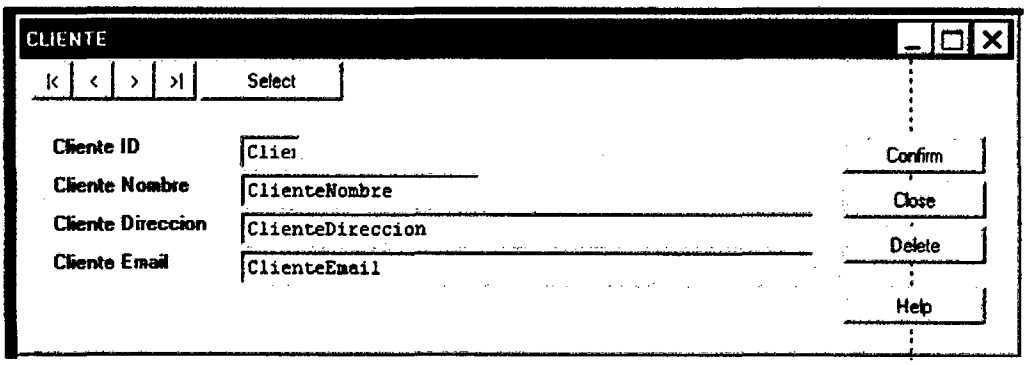


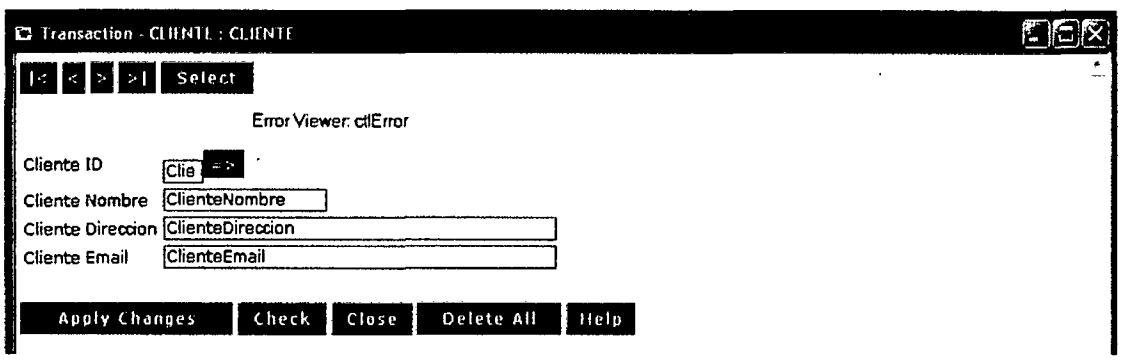
5.1.1.1.8. Creación del Objeto Transacción Cliente

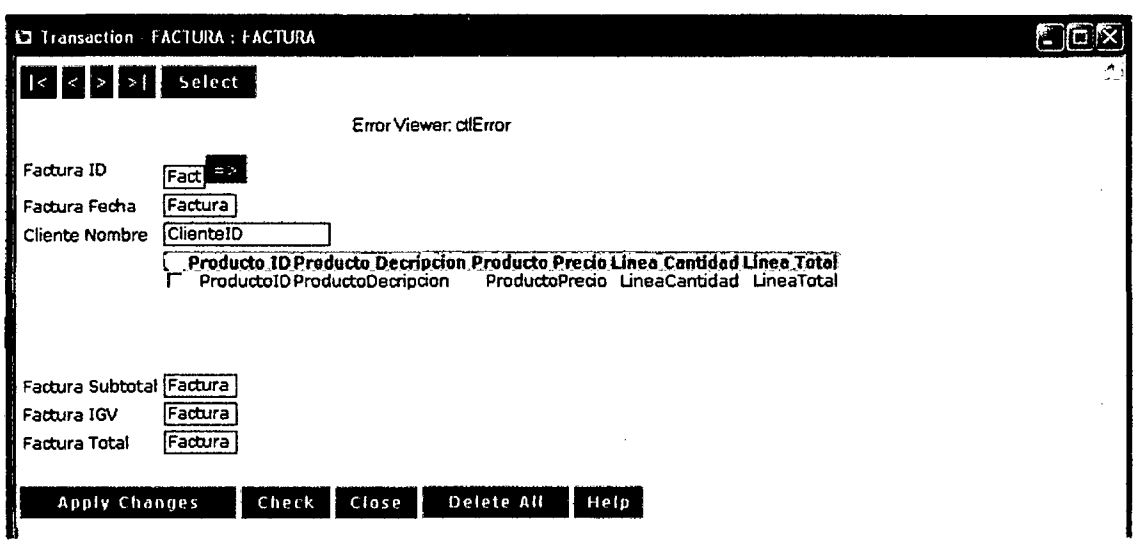
ATRIBUTO	TIPO	DESCRIPCION
ClienteID	----------	----------
ClienteNombre	----------	----------
ClienteDireccion	Carácter (50)	Cliente Dirección
ClienteEmail	Carácter (50)	Cliente Email






CLIENTE
Select
Cliente ID
Clie
Cliente Nombre
ClienteNombre
Cliente Direccion
ClienteDireccion
Cliente Email
ClienteEmail
Confirm
Close
Delete
Help


Transaction - CLIENTE : CLIENTE
Select
Error Viewer: ctlError
Cliente ID
Clie
Cliente Nombre
ClienteNombre
Cliente Direccion
ClienteDireccion
Cliente Email
ClienteEmail
Apply Changes
Check
Close
Delete All
Help


Transaction - FACTURA : FACTURA
Select
Error Viewer: ctlError
Factura ID
Fact
Factura Fecha
Factura
Cliente Nombre
ClienteID
Producto ID Producto Decripcion Producto Precio Linea Cantidad Linea Total
ProductoID ProductoDecripcion ProductoPrecio LineaCantidad LineaTotal
Factura Subtotal
Factura
Factura IGV
Factura
Factura Total
Factura
Apply Changes
Check
Close
Delete All
Help

5.1.1.1.9. Revisión de los Cambios efectuados al Modelo de Datos

Table CLIENTE

Name	CLIENTE
Description	CLIENTE
ID	3

Table Structure

Name	Description	Type	Formula	Subtype of
ClienteID	Cliente ID	N (4.0)		
ClienteNombre	Cliente Nombre	C (20)		
ClienteDireccion	Cliente Direccion	C (50)		
ClienteEmail	Cliente Email	C (50)		

Table FACTURA

Name	FACTURA
Description	FACTURA
ID	1

Table Structure

Name	Description	Type	Formula	Subtype of
FacturaID	Factura ID	N (4.0)		
FacturaFecha	Factura Fecha	D		
ClienteID	Cliente ID	N (4.0)		
FacturaSubtotal	Factura Subtotal	N (8.2)	SUM(LineaTotal)	
FacturaIGV	Factura IGV	N (8.2)	FacturaSubtotal*0.19	
FacturaTotal	Factura Total	N (8.2)	FacturaSubtotal+FacturaIGV	

Table FACTURALevel1

Name	FACTURALevel1
Description	Level1
ID	2

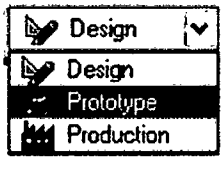
Table Structure

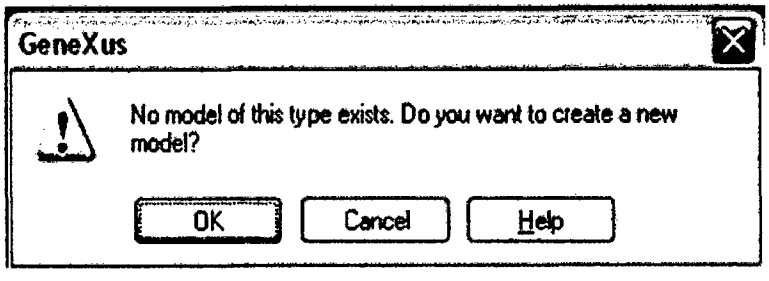
Name	Description	Type	Formula	Subtype of
FacturaID	Factura ID	N (4.0)		
ProductoID	Producto ID	N (4.0)		
ProductoPrecio	Producto Precio	N (8.2)		
ProductoDecripcion	Producto Decripcion	C (20)		
LineaCantidad	Linea Cantidad	N (4.0)		
LineaTotal	Linea Total	N (8.2)	ProductoPrecio*LineaCantidad	

5.1.1.2. FASE DE PROTOTIPO: Generación Automática de la Base de Datos.

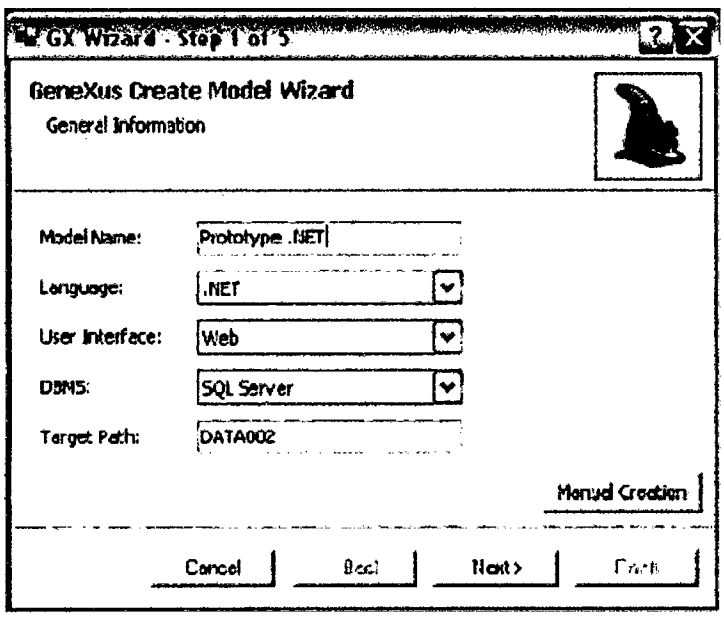
5.1.1.2.1. Prototipando de la aplicación.

5.1.1.2.2. Prototipando de la Aplicación en .NET con SQL Server 2005 Express Edition

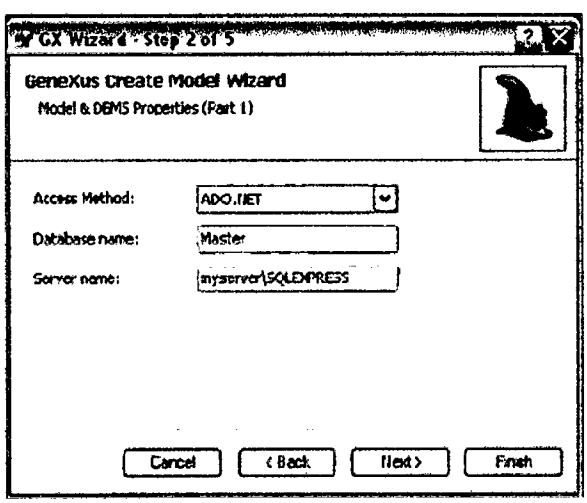




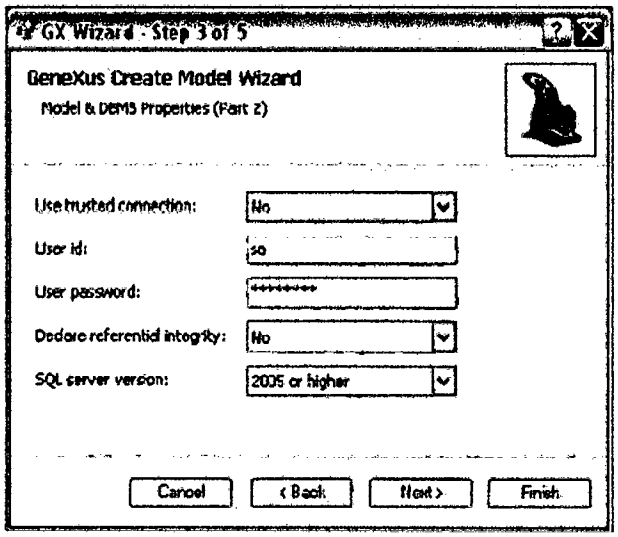
14. El Ayudante Para la Creación de Modelos Genexus lo guiará en la configuración de los parámetros del nuevo modelo. Configure lo siguiente:
    - Nombre del Modelo : Prototype .NET
    - Lenguaje : .NET
    - Interfase de Usuario : Web
    - DBMS : SQLServer
    - Ruta Objetivo : Deje el valor predeterminado

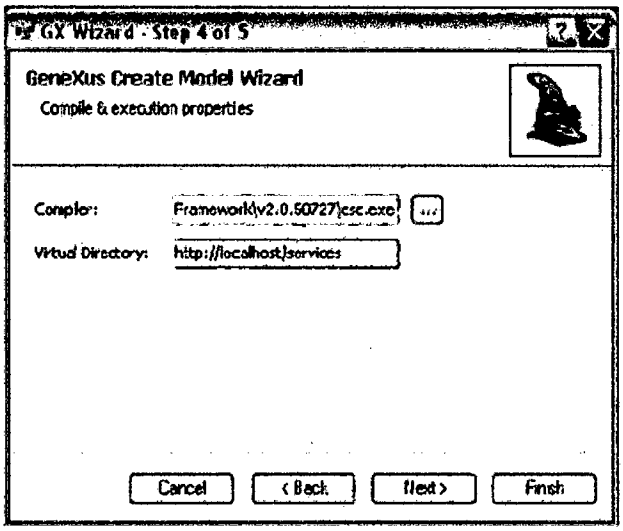


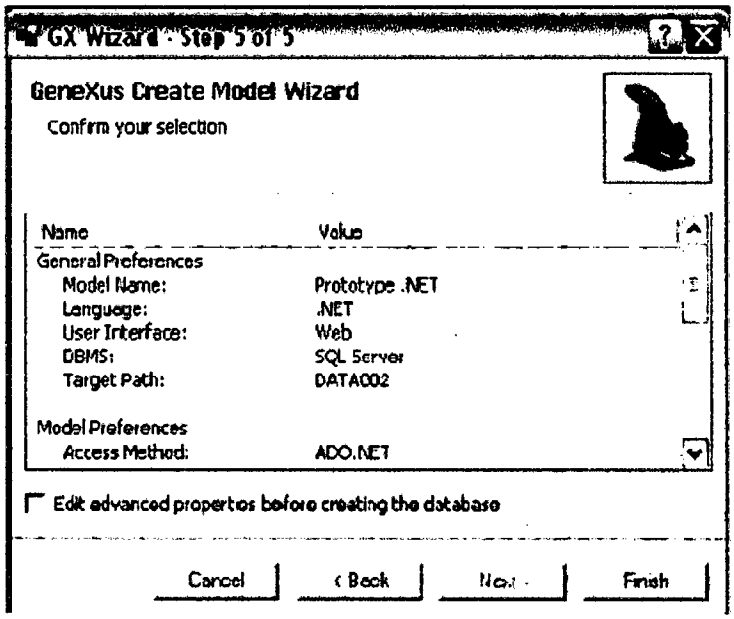
- Método de Acceso : ADO.NET
- Nombre de la Base de Datos : Master 10
- Nombre del Servidor : <Machine Name>\SQLEXPRESS

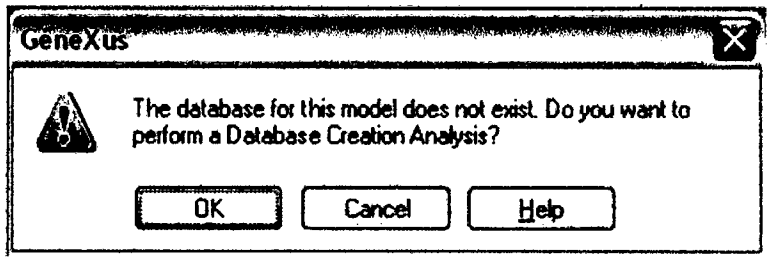


- Usar conexión segura : No
- ID de Usuario : sa
- Contraseña de Usuario : 18854554
- Declarar integridad referencial : No
- Versión de SQL server : 2005 o superior









5.1.1.2.3. Visualización del informe de creación de la base de Datos.

Name CLIENTE
Description CLIENTE
ID 3

Table Structure

Name	Description	Type	Formula	Subtype of
ClienteID	Cliente ID	N (4.0)		
ClienteNombre	Cliente Nombre	C (20)		
ClienteDireccion	Cliente Direccion	C (50)		
ClienteEmail	Cliente Email	C (50)		

Indices

Name	Type	Composition
ICLIENTE	Primary Key	ClienteID

Superordinated To

Id	Table	By
1	FACTURA	ClienteID

Name FACTURA
Description FACTURA
ID 1

Table Structure

Name	Description	Type	Formula	Subtype of
FacturaID	Factura ID	N (4.0)		
FacturaFecha	Factura Fecha	D		
ClienteID	Cliente ID	N (4.0)		
FacturaSubtotal	Factura Subtotal	N (8.2)	SUM(LineaTotal)	
FacturaIGV	Factura IGV	N (8.2)	FacturaSubtotal*0.19	
FacturaTotal	Factura Total	N (8.2)	FacturaSubtotal+FacturaIGV	

Indices

Name	Type	Composition
IFACTURA	Primary Key	FacturaID
IFACTURA1	Foreign Key	ClienteID

Subordinated To

Id	Table	By
3	CLIENTE	ClienteID

Superordinated To

Id	Table	By
2	FACTURALevel1	FacturaID

Table FACTURALevel1

Name	FACTURALevel1
Description	Level1
ID	2

Table Structure

Name	Description	Type	Formula	Subtype of
FacturaID	Factura ID	N (4.0)		
ProductoID	Producto ID	N (4.0)		
ProductoPrecio	Producto Precio	N (8.2)		
ProductoDecripcion	Producto Decripcion	C (20)		
LineaCantidad	Linea Cantidad	N (4.0)		
LineaTotal	Linea Total	N (8.2)	ProductoPrecio*LineaCantidad	

Indices

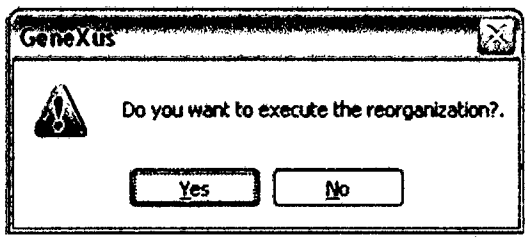
Name	Type	Composition
IFACTURALevel1	Primary Key	FacturaID, ProductoID
IFACTURALevel11	Foreign Key	FacturaID

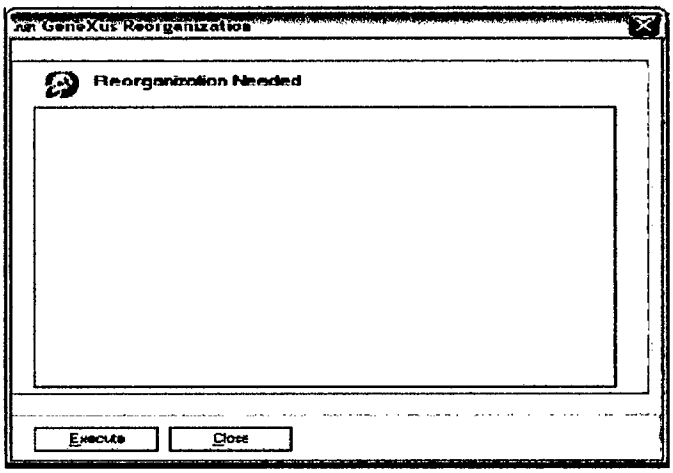
Subordinated To

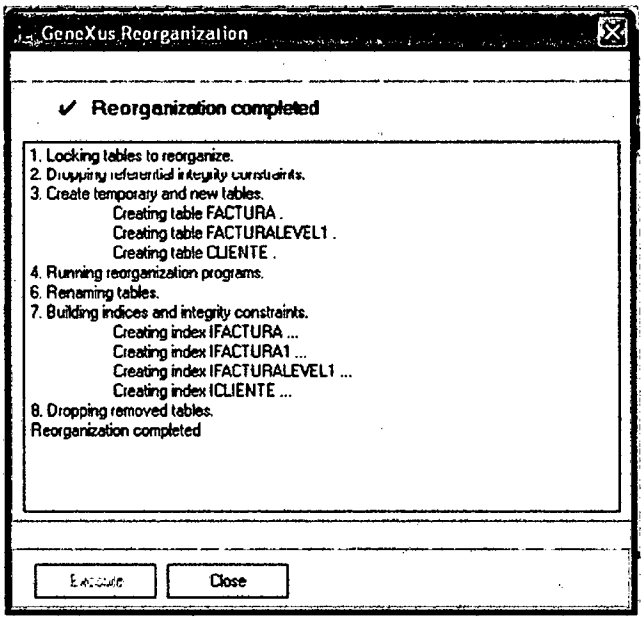
Id	Table	By
1	FACTURA	FacturaID

5.1.1.2.4. Creación de la Base de datos del modelo de prototipo.

Reorganización de Genexus (Base de Datos):



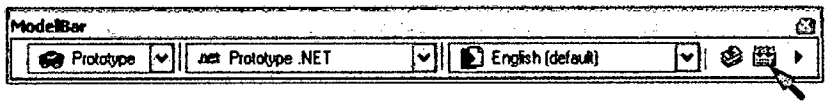


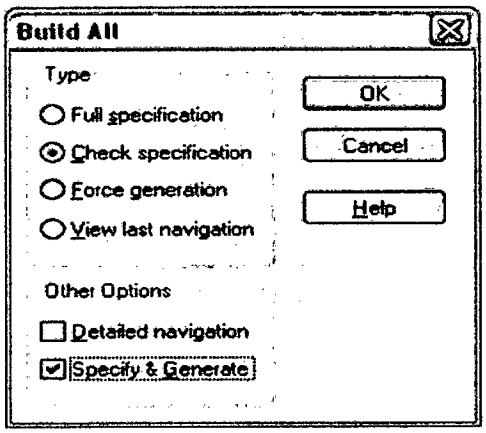


5.1.1.3. FASE DE PROTOTIPO: Generación Automática de Código.

Hasta el momento, hemos creado un nuevo esquema de base de datos que soporta el modelo de datos inferido por Genexus para su modelo de Prototipo. A continuación generaremos el código fuente para su aplicación en el entrono .NET.

5.1.1.3.1. Especificación y Generación Automatica de Código Fuente: Comando Build.





5.1.1.3.2. Visualización del Reporte de Especificación:

Transaction CLIENTE Navigation Report

Name	CLIENTE	Environment	.NET
Description	CLIENTE	Spec. Version	9_0_6-009
Status	**No Generation Required**	Form Class	HTML
		Program Name	TCLIENTE
		Parameters	

Levels

Level CLIENTE

=CLIENTE(*ClienteID*)

Insert into CLIENTE (ClienteNombre, ClienteDireccion, ClienteEmail)

Update CLIENTE (ClienteNombre, ClienteDireccion, ClienteEmail)

Delete from CLIENTE

Referential integrity controls on delete:
- FACTURA(*ClienteID*)

Prompts

Table	Program	In Parameters	Out Parameters
CLIENTE	Gx0030		ClienteID

Reporte de Especificación de la Transación Factura.

Transaction FACTURA Navigation Report

Name	FACTURA	Environment	.NET
Description	FACTURA	Spec. Version	9_0_6-009
Status	**No Generation Required**	Form Class	HTML
		Program Name	TFACTURA
		Parameters	

Warnings

spc0107 Candidate key ClienteNombre for ClienteID may have duplicated values.

Levels

Level FACTURA

=FACTURA(*FacturaID*)
=CLIENTE(*ClienteID*)

Insert into FACTURA (FacturaFecha, ClienteID)

Update FACTURA (FacturaFecha, ClienteID)

Delete from FACTURA

Formulas:

Navigation to evaluate: FacturaSubtotal

=FACTURA(*FacturaID*)
=FACTURALEVEL1(*FacturaID*)

Level FACTURALEVEL1

=FACTURALEVEL1(*FacturaID, ProductoID*)

Insert into FACTURALEVEL1 (FacturaID, ProductoID, ProductoDescripcion, ProductoPrecio, LineaCantidad)

Update FACTURALEVEL1 (ProductoDescripcion, ProductoPrecio, LineaCantidad)

Delete from FACTURALEVEL1

Prompts

Table	Program	In Parameters	Out Parameters
FACTURALEVEL1	Gx0021	FacturaID	ProductoID
FACTURA	Gx0010		FacturaID

Suggestions

Control	Type	Navigation
ClienteID	1	=CLIENTE(*ClienteID*) INTO ClienteNombre Where ClienteNombre. toupper() like @ClienteNombre. toupper() Order ClienteNombre

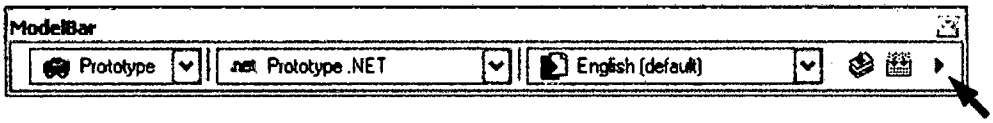
Lista para seleccionar objetos Genexus

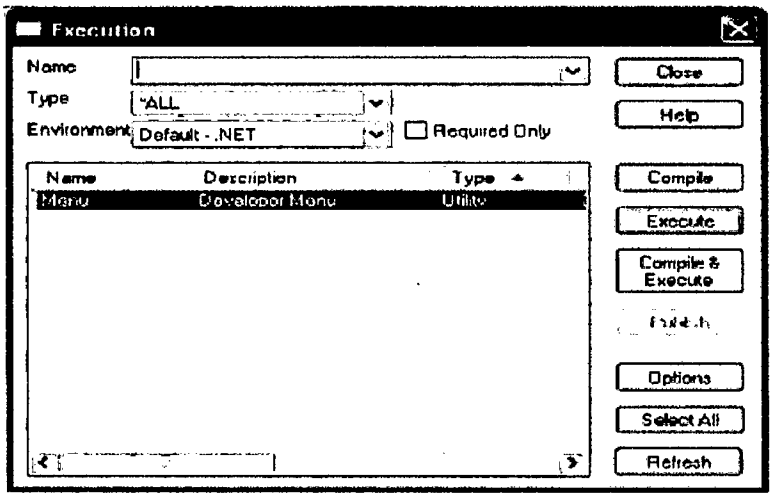
Name	Type	Description	Modified
CLIENTE	Transaction	CLIENTE	2010-10-02 22:17:46
FACTURA	Transaction	FACTURA	2010-10-04 23:33:24
Gx0010	Web Prompt	Selection List FACTURA	2010-10-04 23:33:00
Gx0021	Web Prompt	Selection List FACTURA1	2010-10-04 23:33:00
Gx0030	Web Prompt	Selection List CLIENTE	2010-10-04 23:33:00

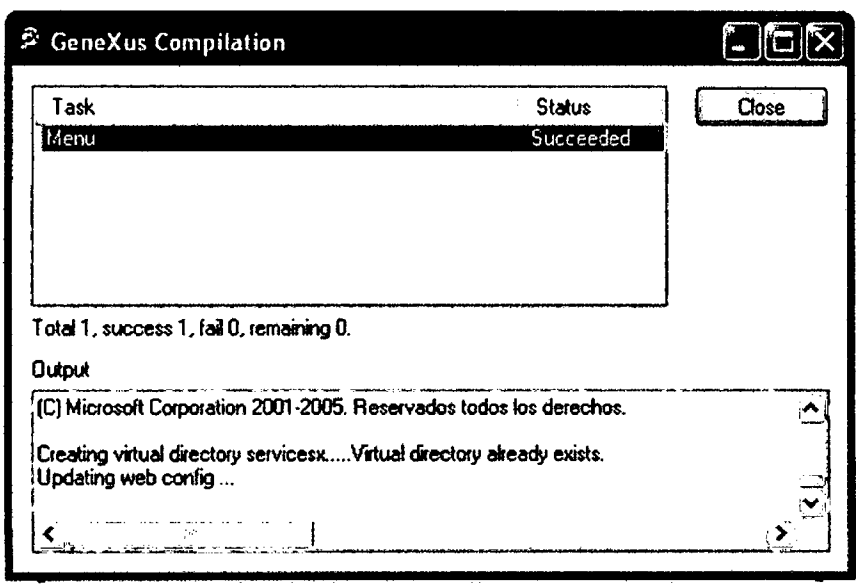
5.1.1.4. Generación de Prototipo Funcional Definido con Genexus

5.1.1.4.1. Ejecución de la Aplicación

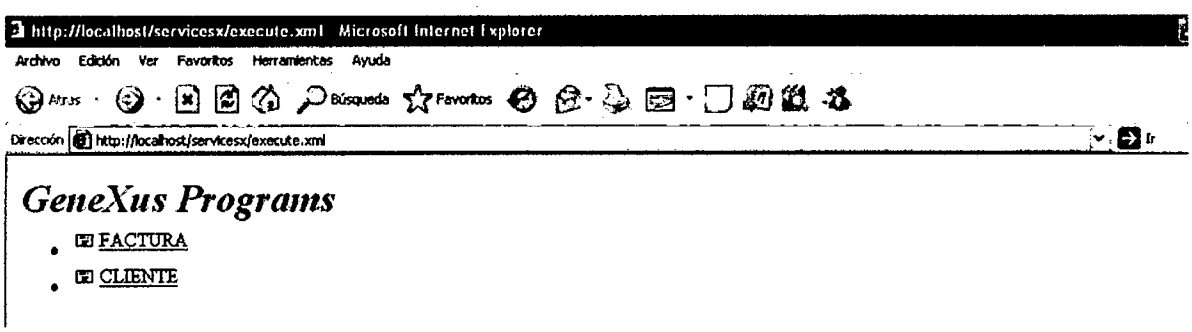
Barra de Herramientas del Modelo





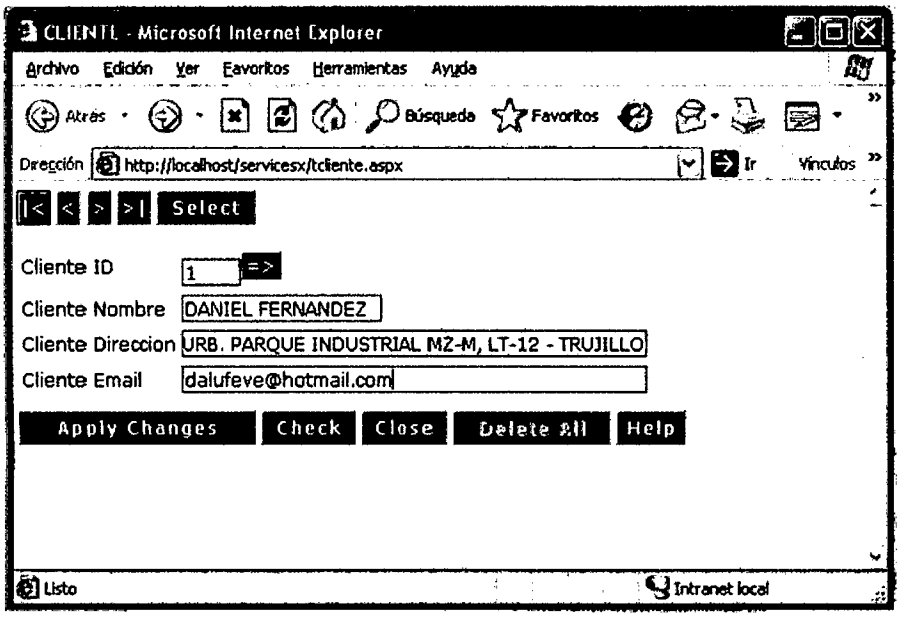


5.1.1.4.2. Prueba de la Aplicación.

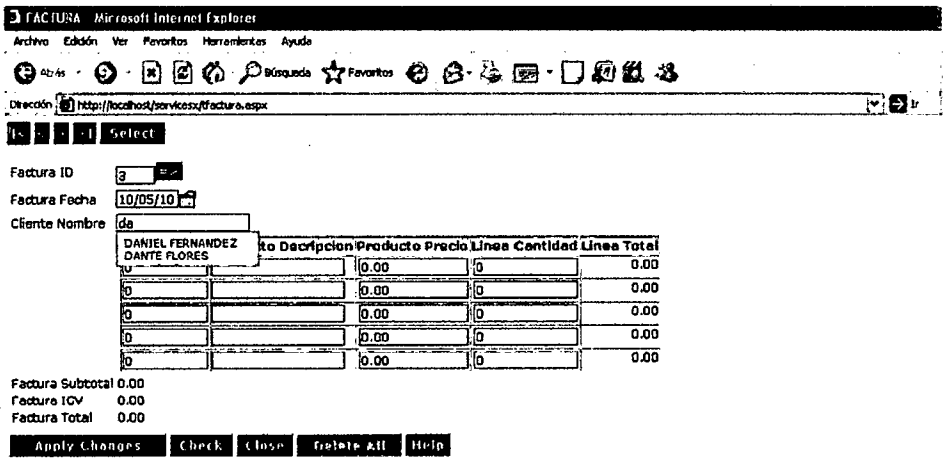


Menú del Desarrollador

Ingreso de algunas instancias del cliente para usarlas en futuras facturas.



Instancia de la Transacción Cliente.



Instancia de la transacción Factura

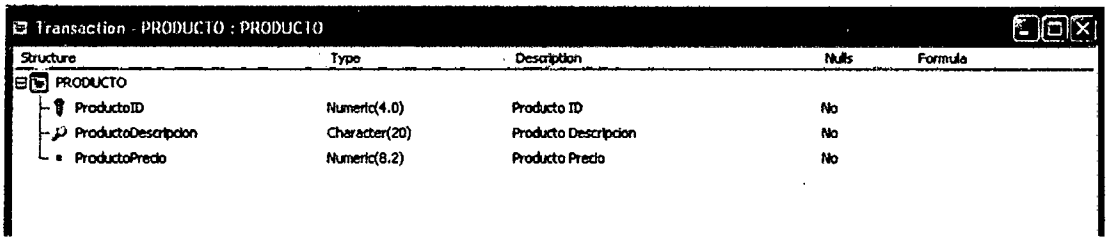
5.1.1.5. Desarrollo Incremental y mantenimiento de la Aplicación.

5.1.1.5.1. Inclusión de Nuevos objetos en el Modelo: Objeto Transacción Producto.

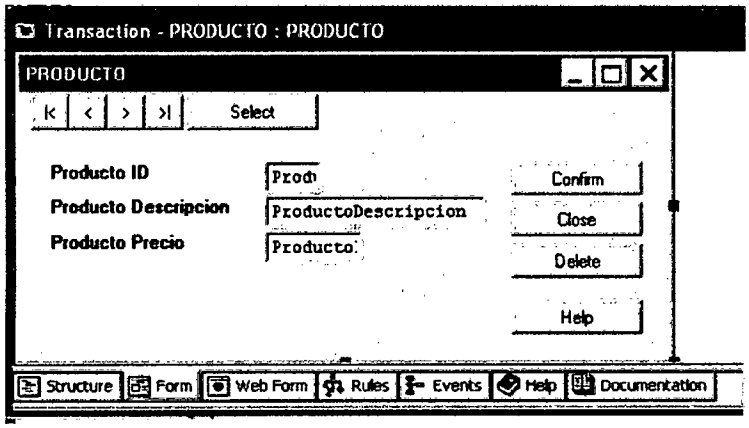
Cree la transacción Producto siguiendo lo descrito en la sección 4.1.9.1.2.: Creación de un Objeto Transacción y en la sección 4.1.9.1.3.: Descripción de la Estructura de la Transacción . Inserte los siguientes atributos en la Estructura de la Transacción Producto:

ATRIBUTO	TIPO	DESCRIPCION
Producto	.....	.....
ProductoDescripcion	.....	.....
ProductoPrecio	.....	.....

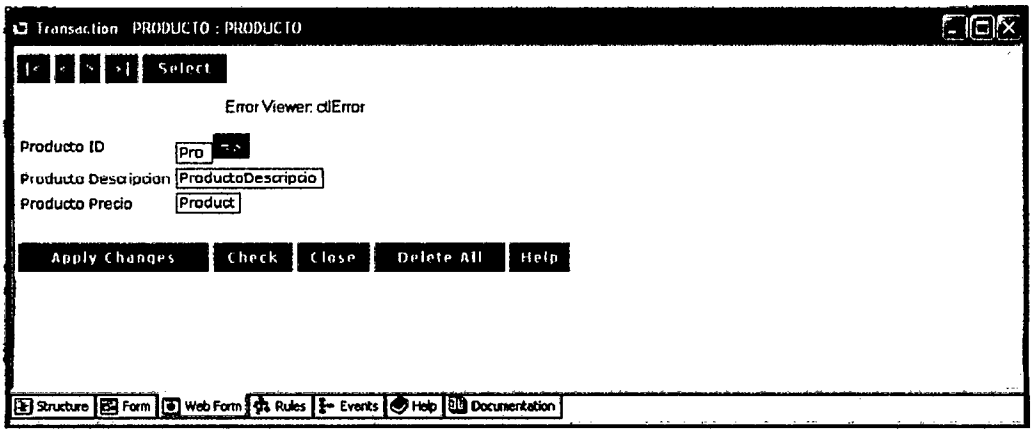
La estructura (Structure) de la transacción Cliente, su Formulario (Windows), y su Formulario Web se verán como sigue.



Estructura de la transacción Producto



Formulario Windows de la transacción Producto.



Formulario Web de la transacción Producto

5.1.1.5.2. Revisión de los cambios efectuados en el Modelo de Datos.

Table FACTURALevel1

Name	FACTURALevel1
Description	Level1
ID	2

Table Structure

Name	Description	Type	Formula	Subtype of
FacturaID	Factura ID	N (4.0)		
ProductoID	Producto ID	N (4.0)		
LineaCantidad	Linea Cantidad	N (4.0)		
LineaTotal	Linea Total	N (8.2)	ProductoPrecio*LineaCantidad	

Table PRODUCTO

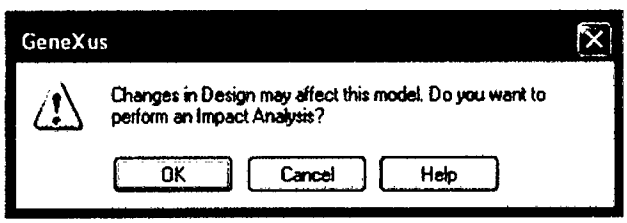
Name	PRODUCTO
Description	PRODUCTO
ID	4

Table Structure

Name	Description	Type	Formula	Subtype of
ProductoID	Producto ID	N (4.0)		
ProductoDescripcion	Producto Descripcion	C (20)		
ProductoPrecio	Producto Precio	N (8.2)		

Listado de Base de Datos (Modelo de Datos) y Transacción Producto

5.1.1.5.3. Análisis de Impacto y Reorganización de la Base de Datos.



Caja de Diálogo del Análisis de Impacto

Table name: FACTURALEVEL1

FACTURALEVEL1 needs conversion

Rename FACTURAFACTURA1 to FACTURALEVEL1

Table Structure

	Attribute	Definition	Previous values	Takes value from
	FacturaID	Numeric (4)Not null		FACTURAFACTURA1.FacturaID
	ProductoID	Numeric (4)Not null		FACTURAFACTURA1.ProductoID
	LineaCantidad	Numeric (4)Not null		FACTURAFACTURA1.LineaCantidad
Del	ProductoDecripcion	Character (20)Not null		
Del	ProductoPrecio	Numeric (8.2)Not null		

Indexes

	Name	Definition	Composition
New	IFACTURALEVEL1	primary key Clustered	FacturaID ProductoID
New	IFACTURALEVEL11	duplicate	ProductoID
Del	IFACTURAFACTURA1	primary key Clustered	FacturaID ProductoID

Foreign key constraints

	Referenced table	Attributes
	FACTURA	FacturaID
New	PRODUCTO	ProductoID

Table name: PRODUCTO

PRODUCTO is new

Warnings

⚠ raz0007 Attribute ProductoDescripcion does not allow nulls and has not a Initial Value. An empty default value will be used.

⚠ raz0005 For each value of ProductoID there may be several values of ProductoPrecio.

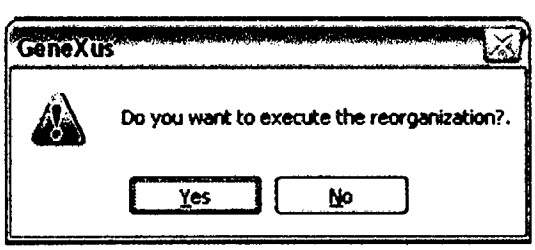
Table Structure

	Attribute	Definition	Previous values	Takes value from
	ProductoID	Numeric (4)Not null		FACTURAFACTURA1.ProductoID
	ProductoDescripcion	Character (20)Not null		
	ProductoPrecio	Numeric (8.2)Not null		FACTURAFACTURA1.ProductoPrecio

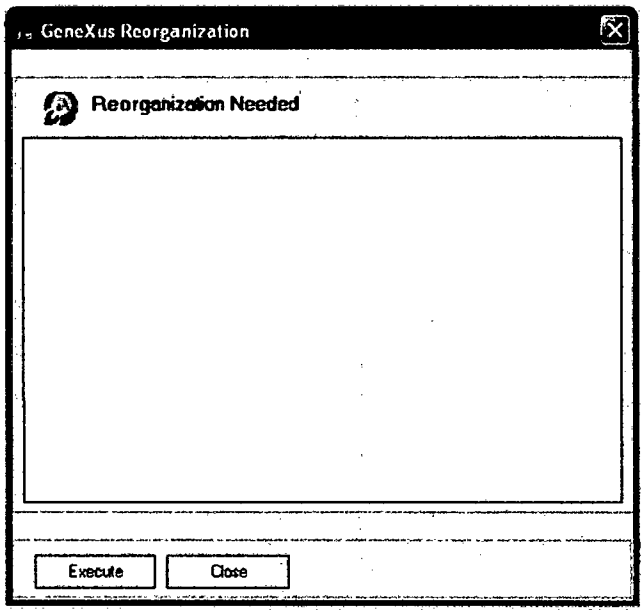
Indexes

Name	Definition	Composition
IPRODUCTO	primary key Clustered	ProductoID

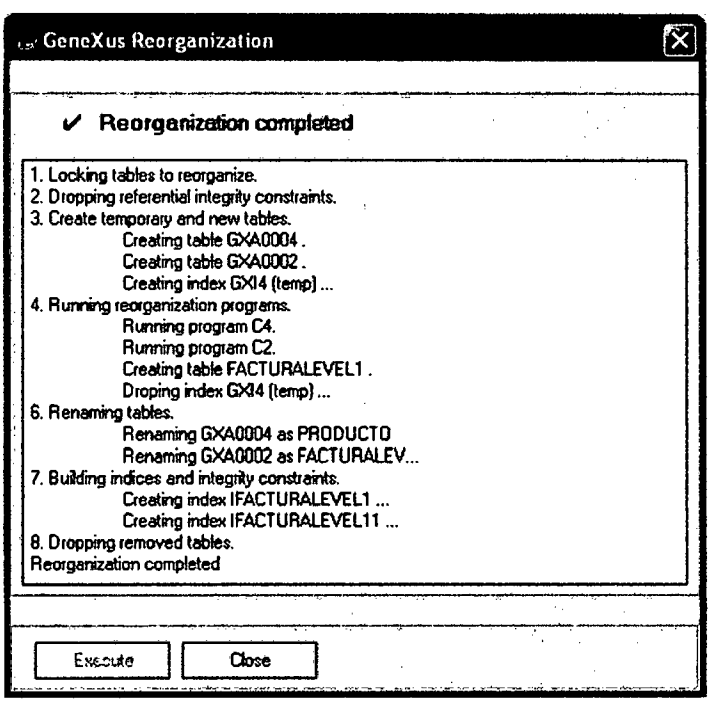
Reporte del Análisis de Impacto



Caja de Diálogo para la Ejecución de la Reorganización

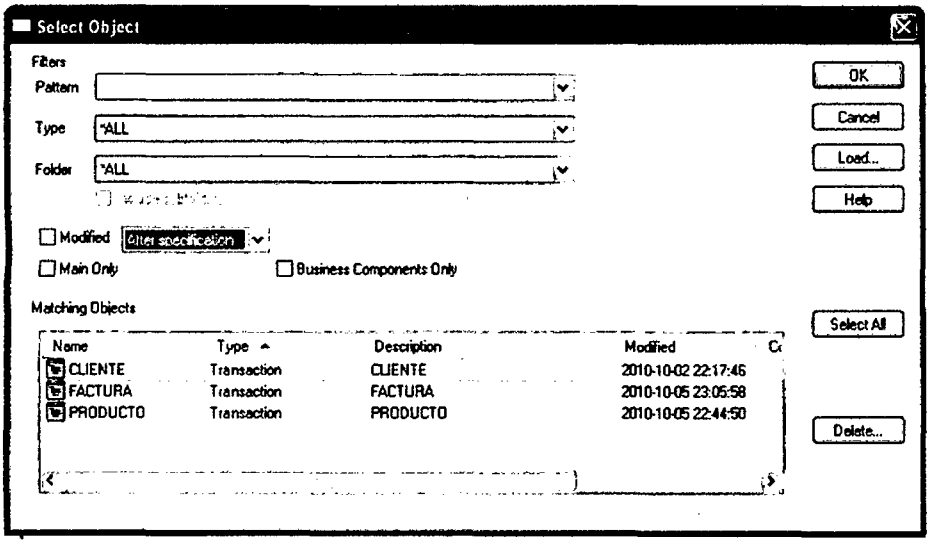


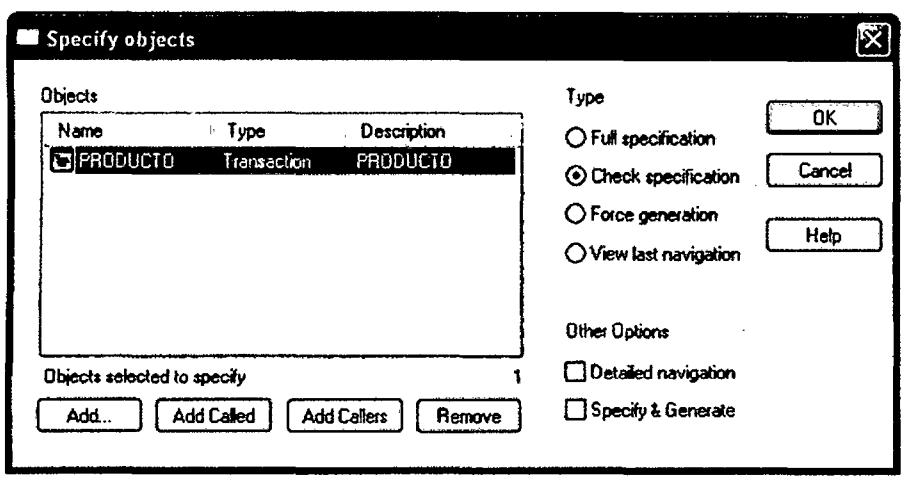
Caja de Diálogo de Reorganización de Genexus



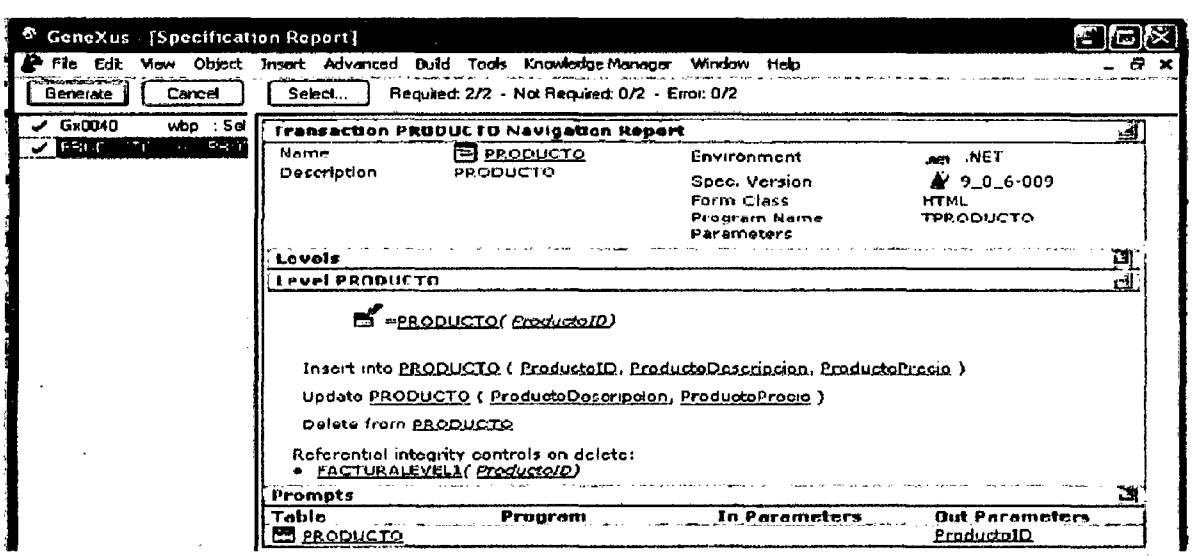
Caja de Diálogo de Reorganización de Genexus

5.1.1.5.4. Regeneración de los programas de la Aplicación.



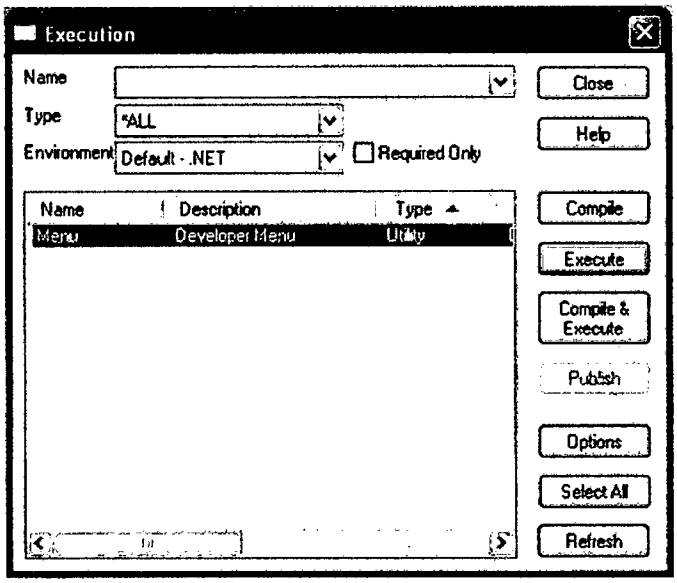


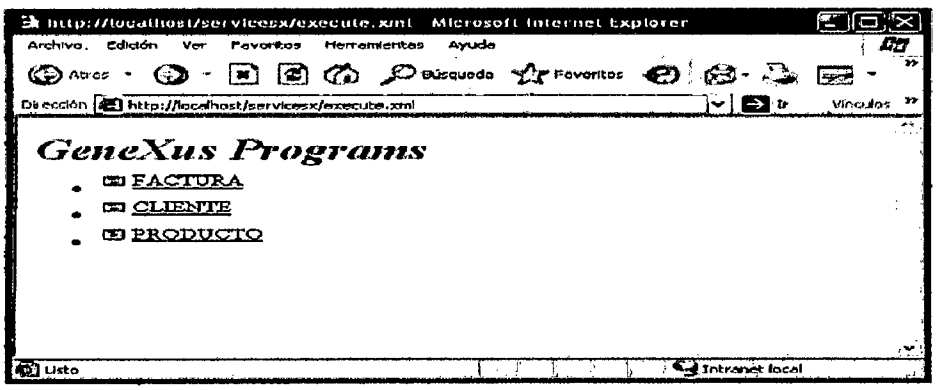
Caja de Diálogo para Especificar Objeto



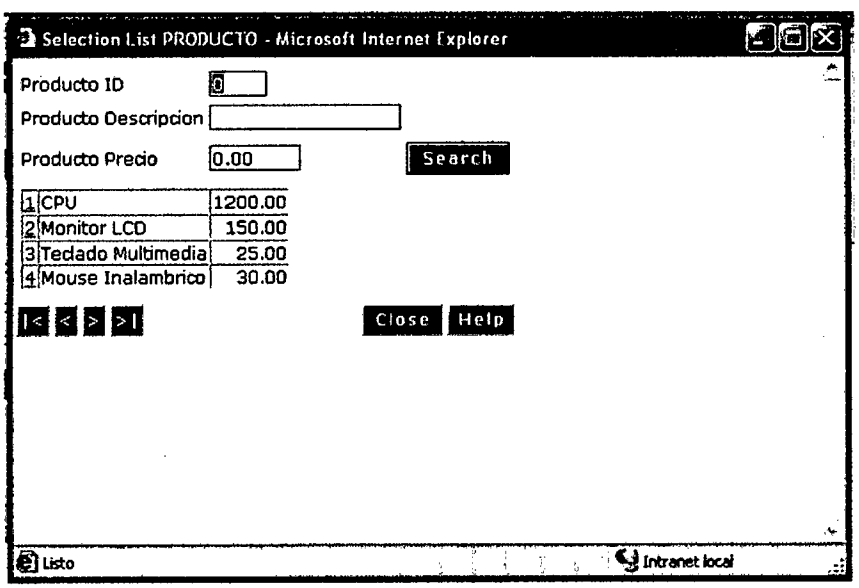
Caja de Diálogo del Reporte de Especificación

5.1.1.5.5. Compilación y Ejecución de la Aplicación.



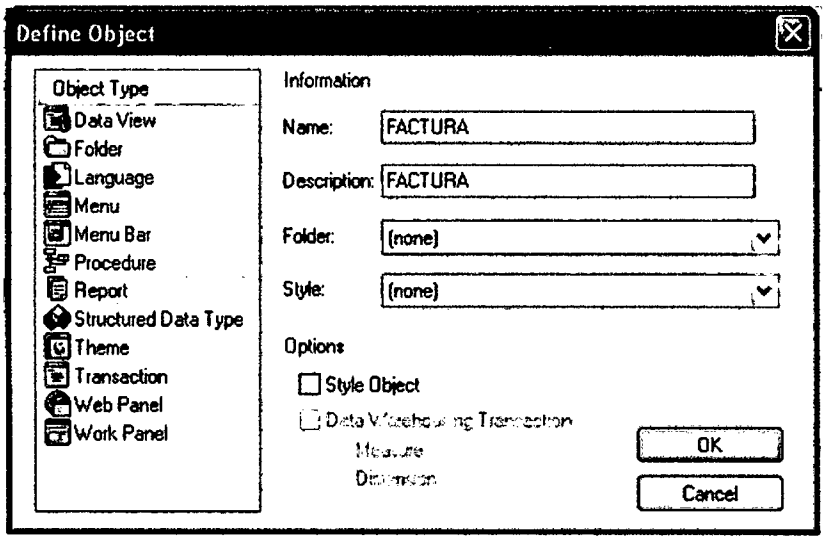


Menú del Desarrollador.

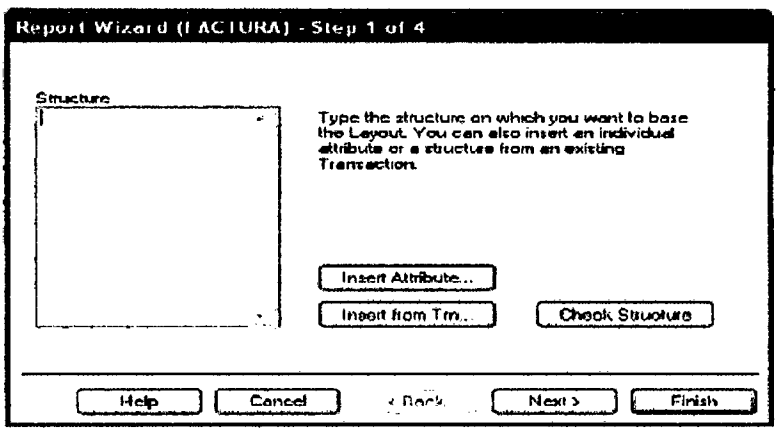


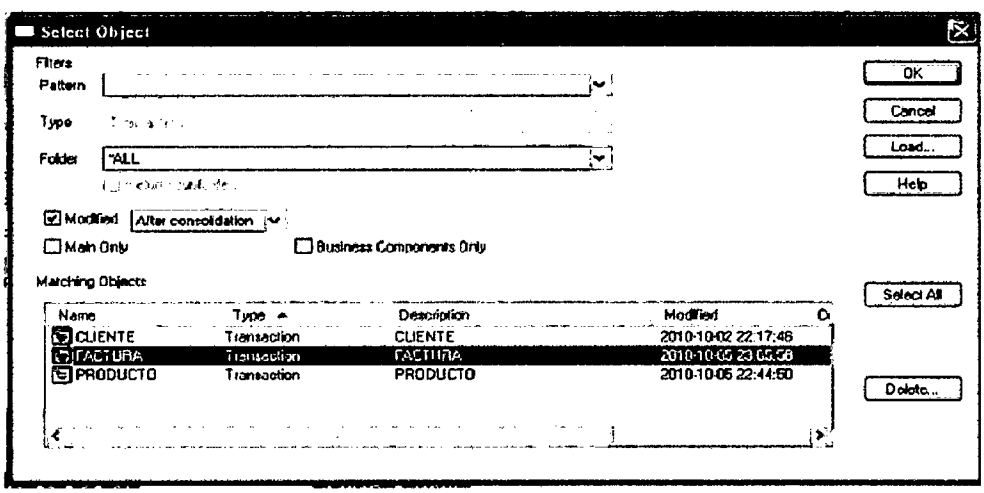
5.1.1.6. Construcción de Procesos no Interactivos (Reportes y Procedimientos)

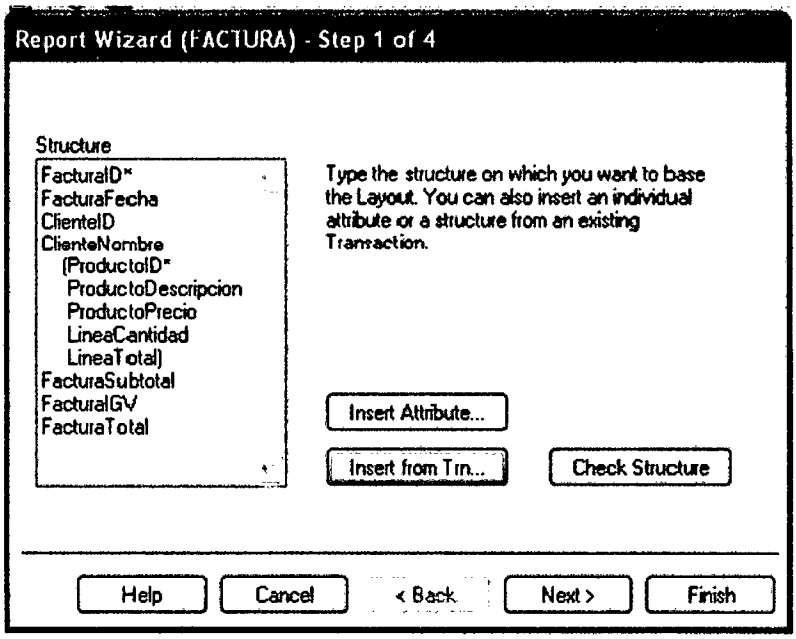
5.1.1.6.1. Creación e Invocación de Un Reporte.



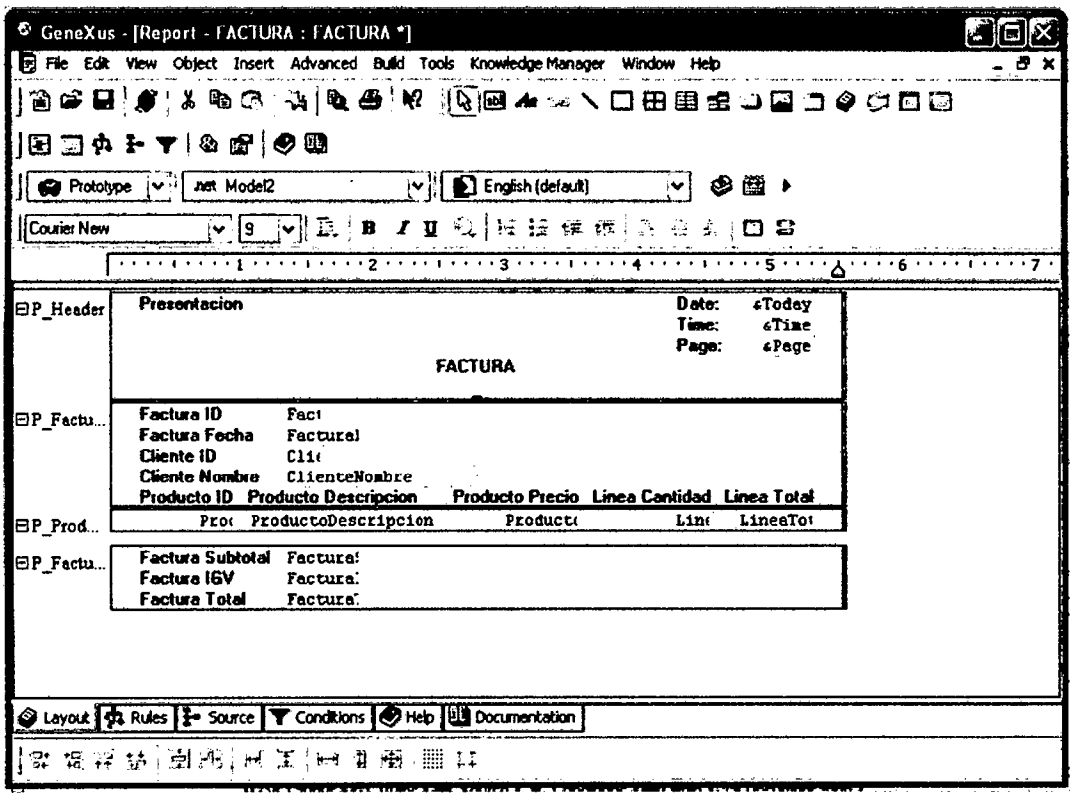
Caja de Diálogo para la Definición del Objeto



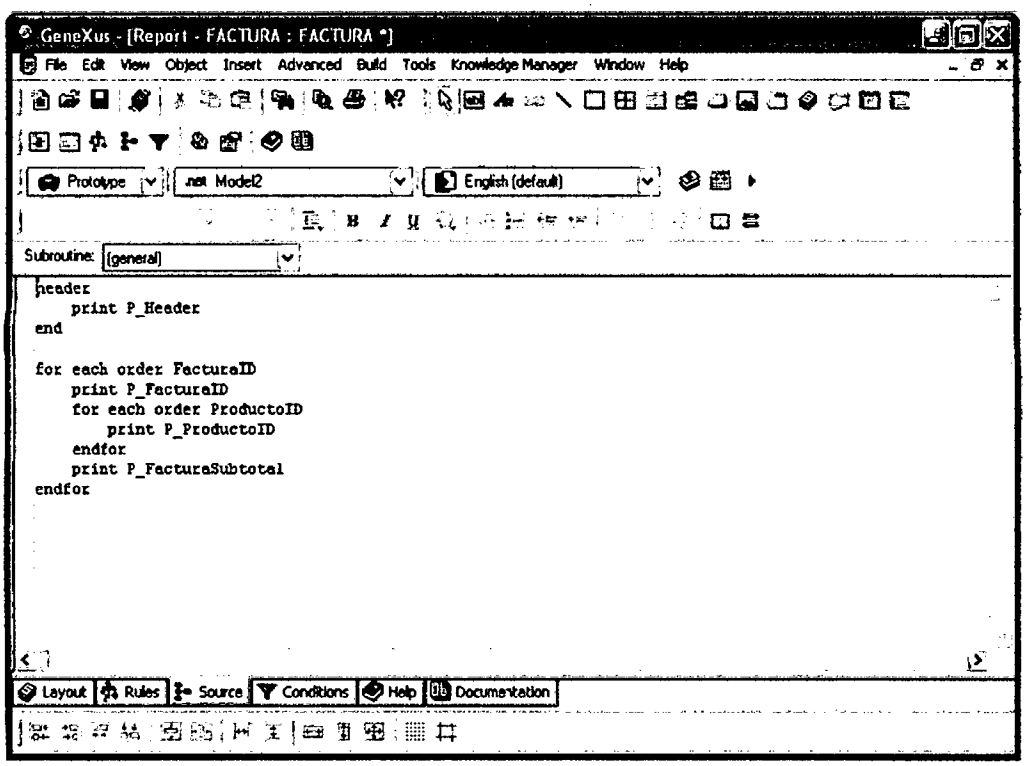




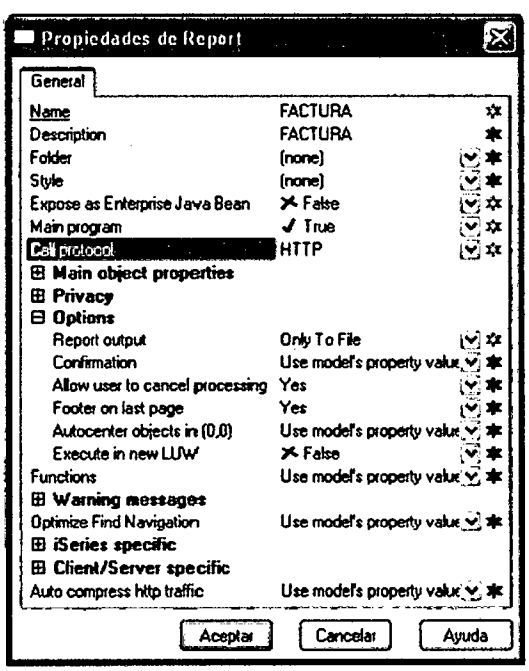
Caja de Diálogo para Seleccionar Objeto



Solapa de Composición del Reporte de Factura (Layout)



Solapa Source del Reporte de Factura

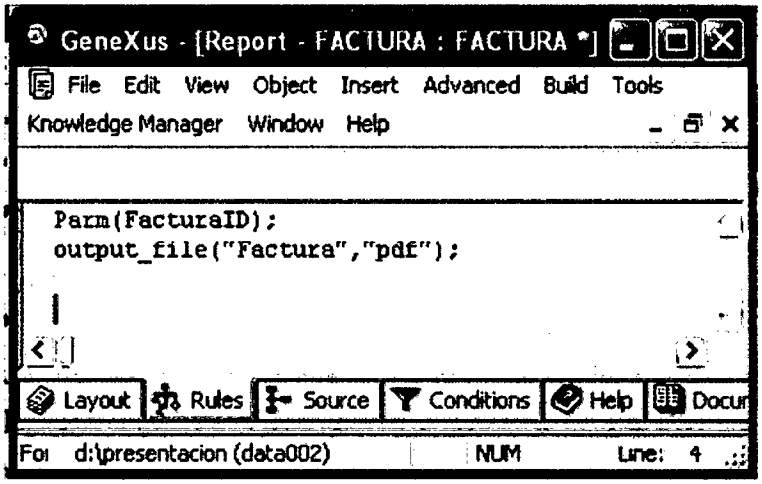


Ventana de las Propiedades del Reporte

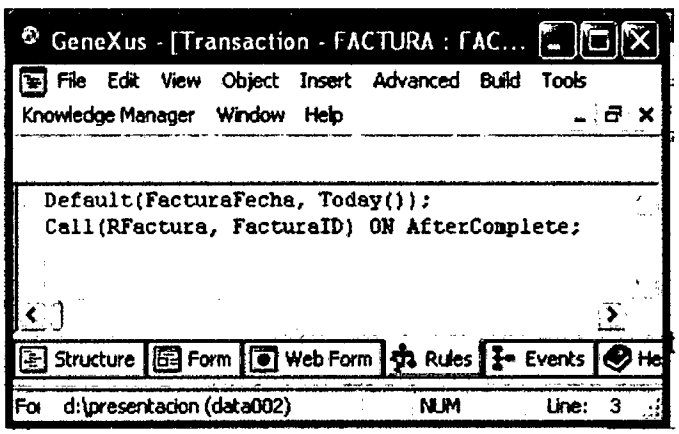
Reglas a incluir

Parm(FacturaD);

Output_file('Factura',"pdf");



Escriba: **Call(RFactura, FacturaID) ON AfterComplete;**



Solapa Reglas de la Transacción Facturas

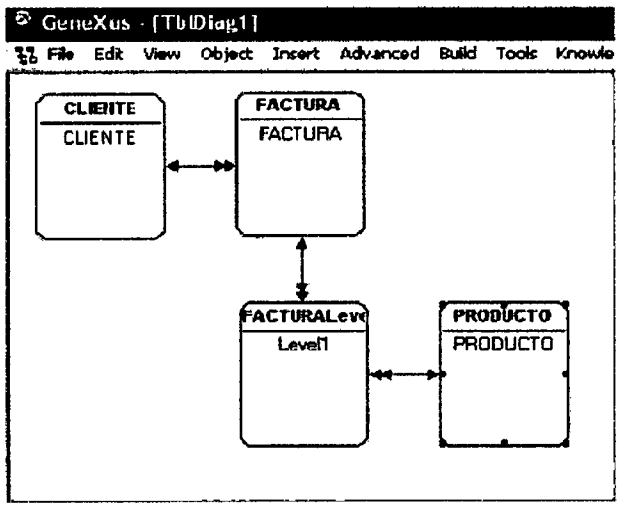
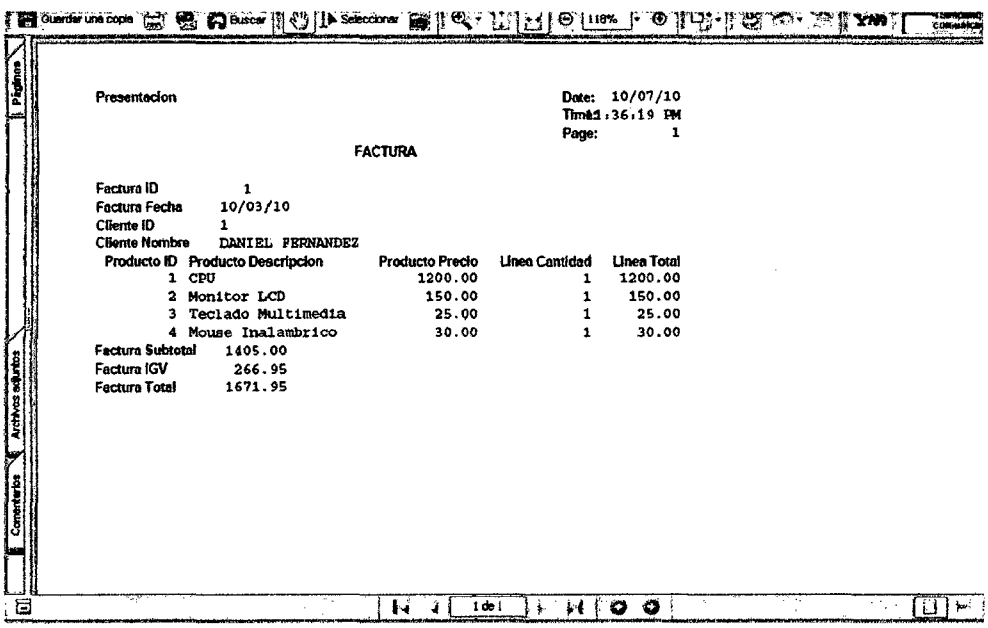


Diagrama de Bachmann del Modelo de Datos

En base a este diagrama podemos identificar la tabla extendida de cada una de las tablas de la aplicación.

Tabla Base	Tabla Extendida
Cliente	Cliente
Producto	Producto
Factura	Factura, Cliente
Factura1	Factura1, Factura, Cliente, Producto

5.1.1.6.2. Especificación, Generación y Ejecución de la Aplicación.



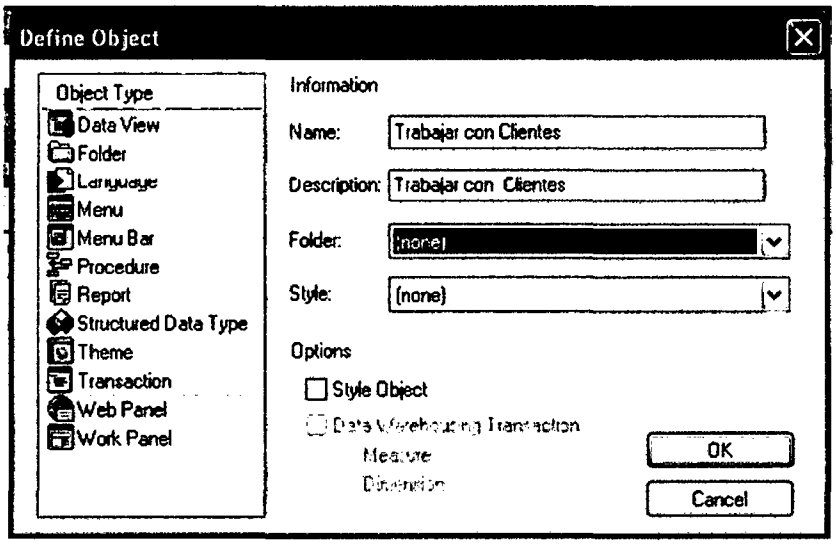
Reporte de Factura

5.1.1.7. Consultas y Diálogos Interactivos (Work Panels y Work Panels)

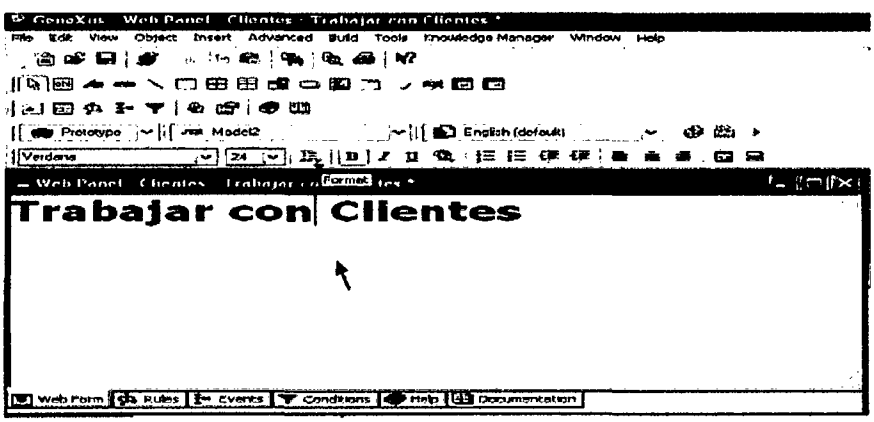
5.1.1.7.1. Creación de un Web Panel: Trabajar con Clientes.

Work Panels y Web Panels

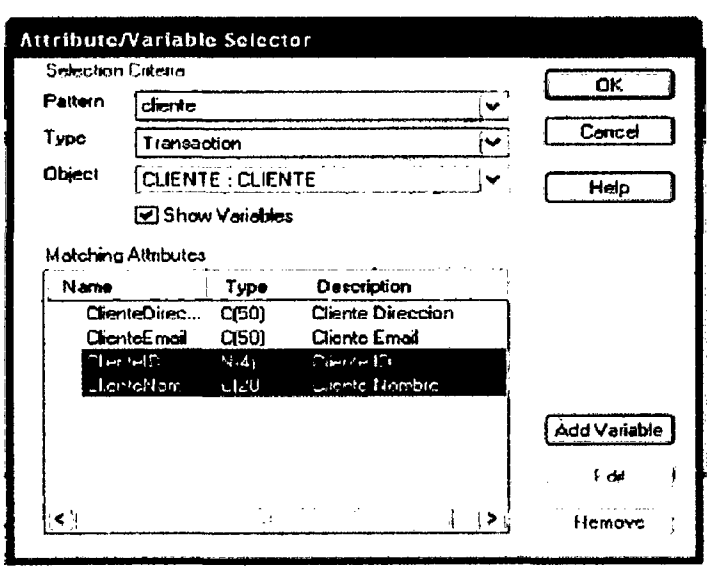
Manejo de Eventos en Web Panels:



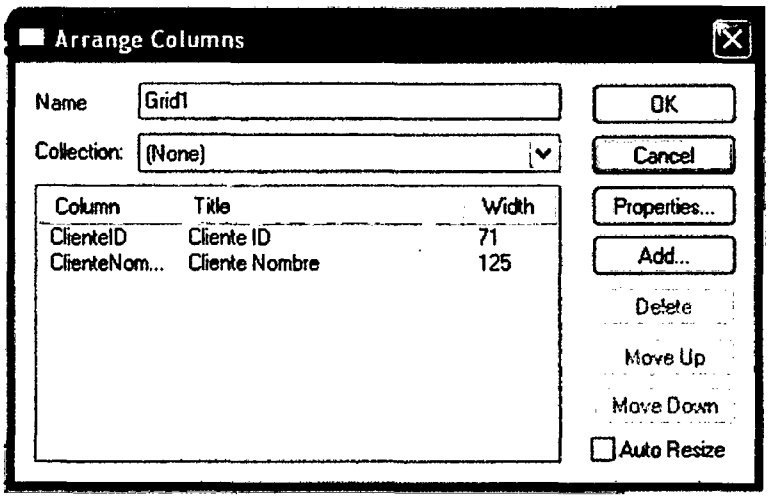
Caja de Diálogo para Definir Objeto



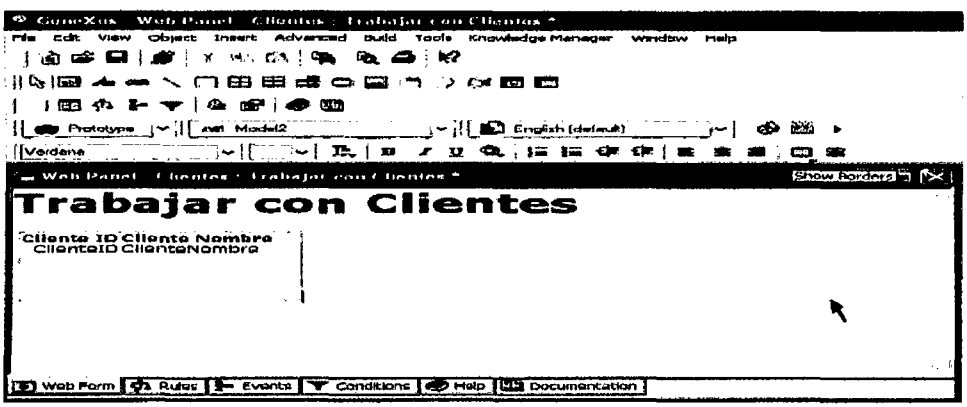
Formulario de Web Panel con Barra de Herramientas de Formato



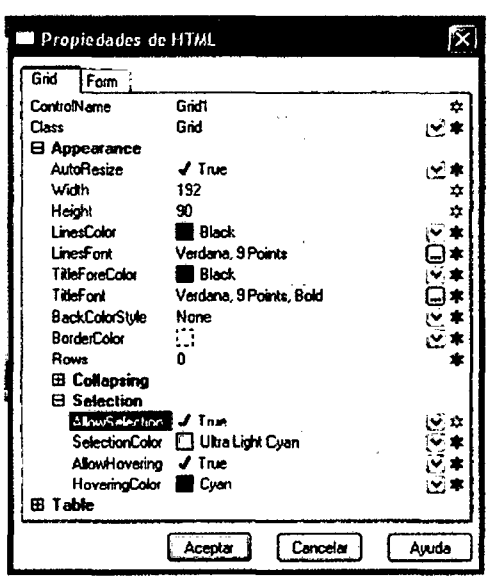
Ventana del Selector de Atributo /Variable



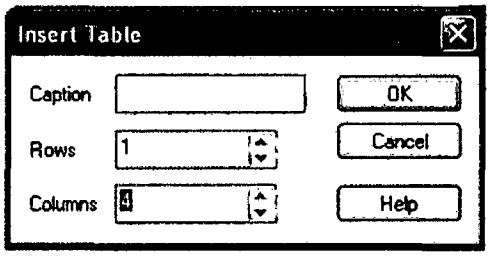
Ventana Arreglar Columnas



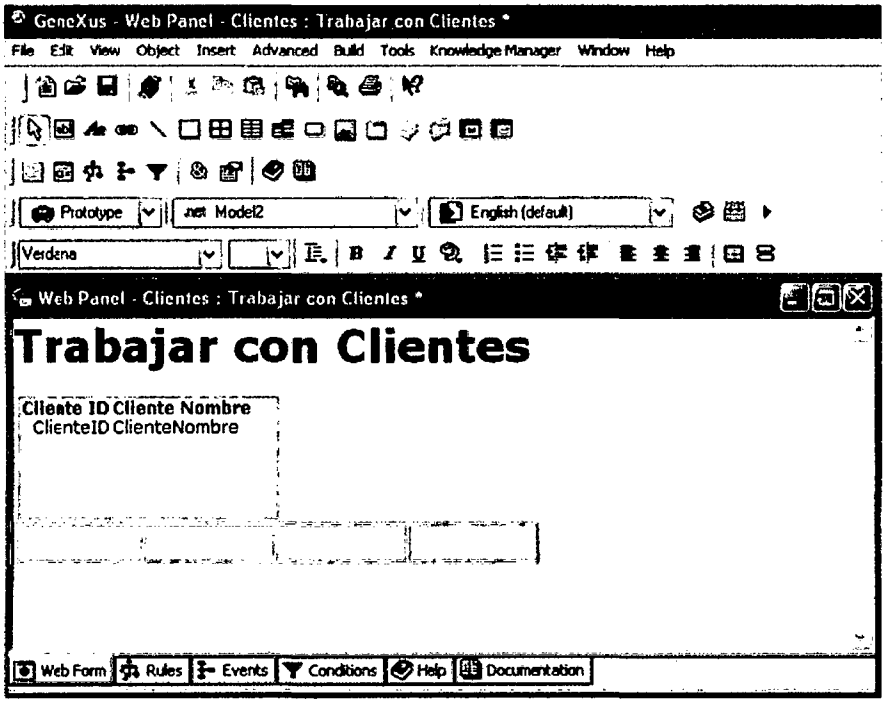
Formulario de Web Panel con grilla



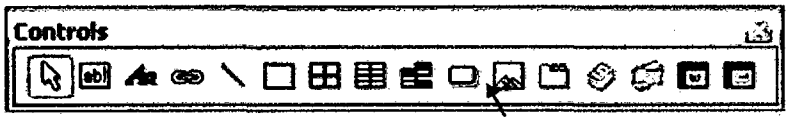
Ventana de Propiedades de la Grilla



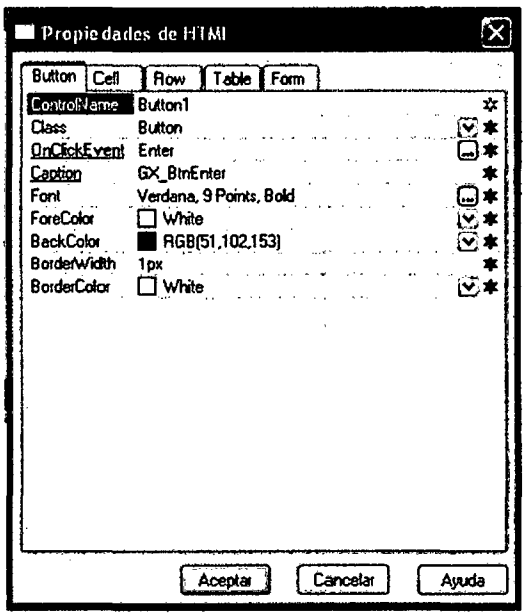
Ventana para Insertar Tabla



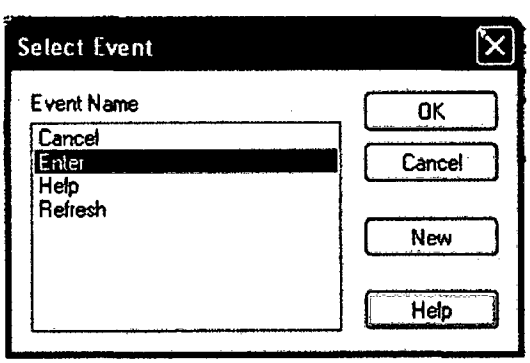
Formulario de Web Panel con grilla y tabla



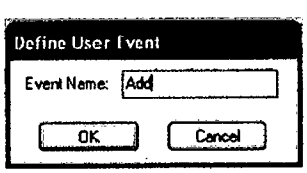
Barra de Herramientas de la Paleta (Controles) – Icono del **Botón**



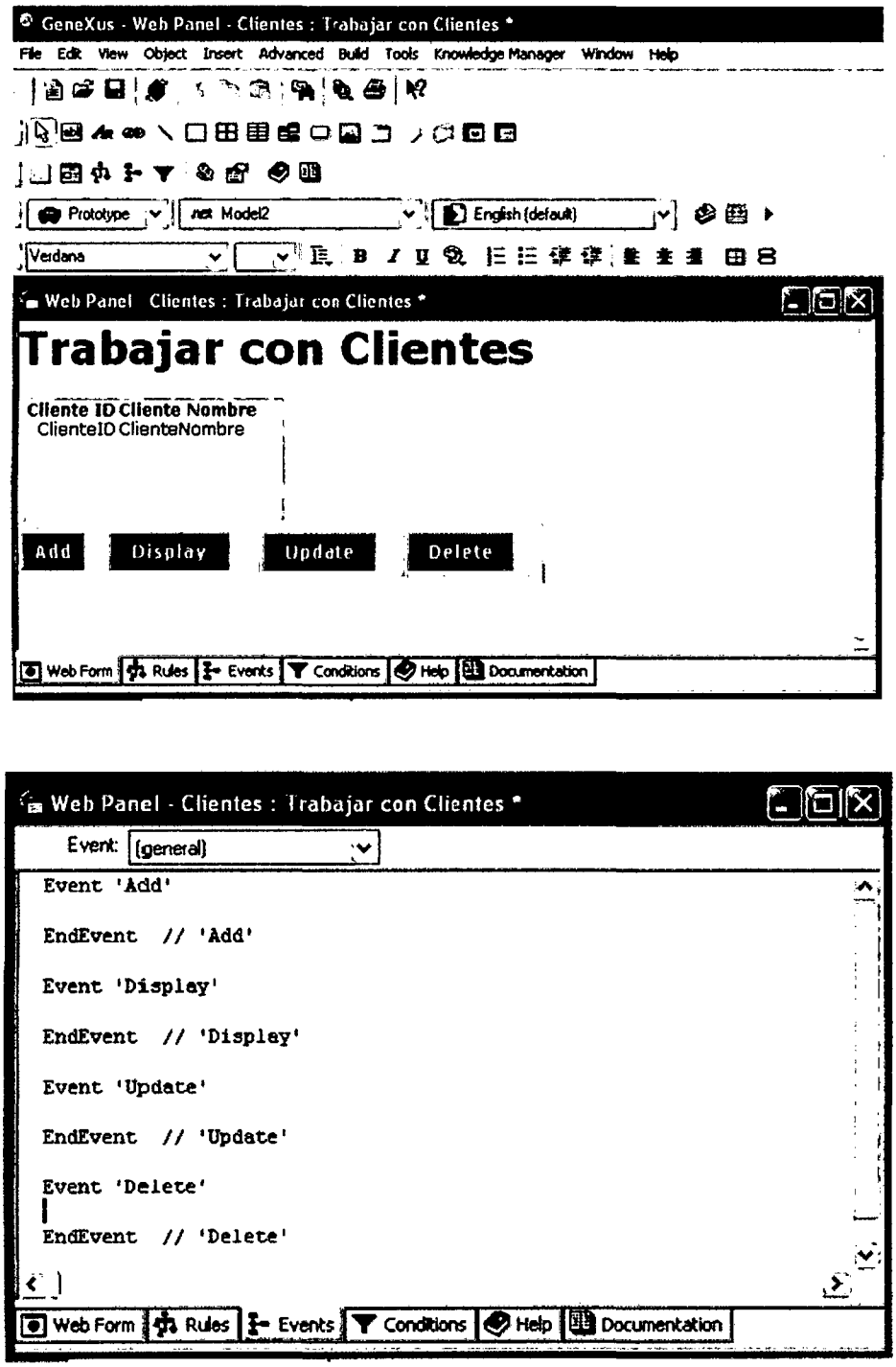
Propiedades HTML del Botón



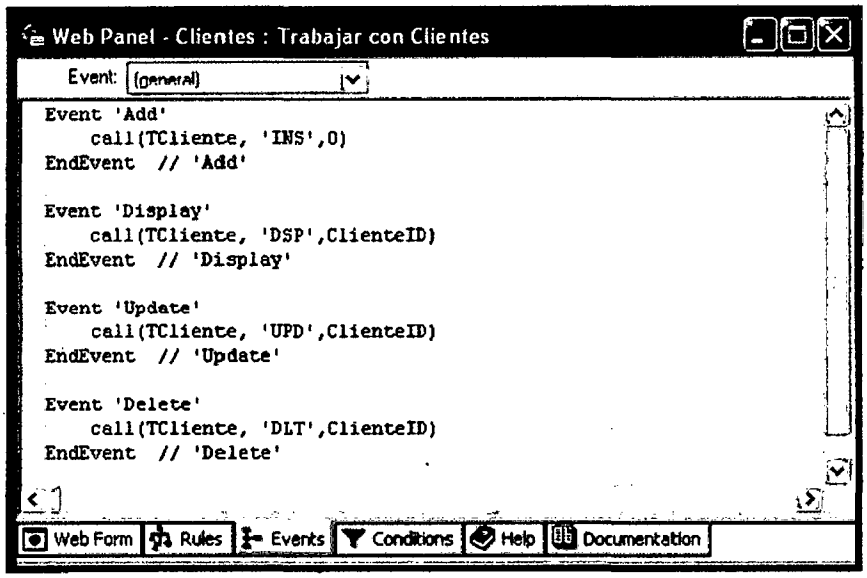
Evento Select



Evento **Define User**

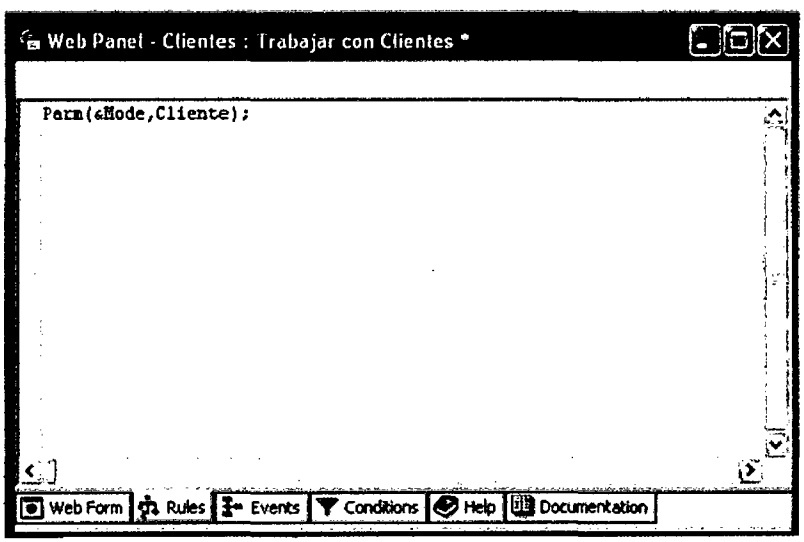


Formulario del Web Panel – Solapa Eventos



Código de los Eventos del Web Panel

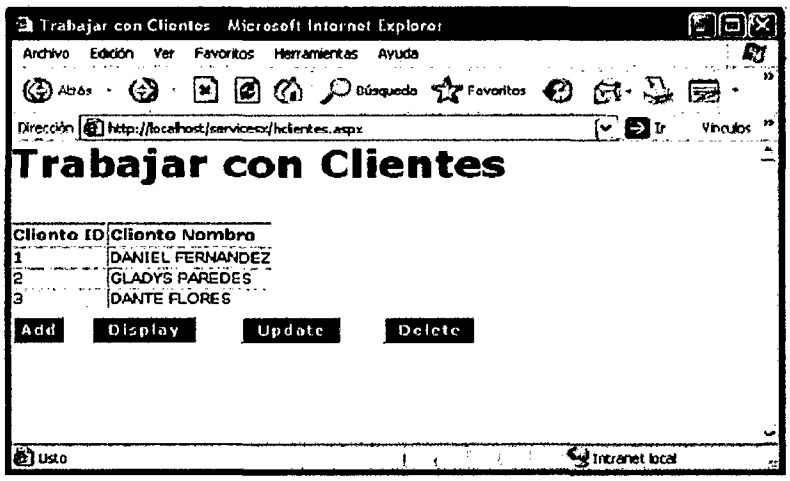
Reglas de la Transacción Cliente: **Parm(&Mode, CustomerID);**



Reglas de la Transacción Cliente

5.1.1.7.2. Ejecutar el Web Panel Trabajar con Clientes.

Menú del Desarrollador con Web Panel



Menú Trabajar con Clientes

5.2. ESTUDIO DE CASO: Aplicación en Empresa de Microfinanzas.

5.2.1. FASE DE ESTUDIO DEL NEGOCIO: Subfases de Estudio de Factibilidad y Estudio Técnico (Resumen):

Una síntesis de los requerimientos, llamémosle comunes y de mayor prioridad a todas las empresas de microfinanzas entrevistadas de la Localidad de Trujillo, fue que el sistema de software debería permitir obtener información de las operaciones de créditos, contabilidad y de la producción y administración de los analistas de créditos de las diferentes agencias distribuidas a nivel nacional, en forma simultánea. Que cada vez que se realice una operación de créditos se actualicen los datos relativos al crédito (solicitudes de créditos, nuevos ingresos, pagos de cuotas, etc.), a su contabilización y lo referente a los records de ventas y pagos de los analistas de créditos. Ver figura Nº 24. Esquema de Topología a Implementar.

Descripción de Módulos identificados

- **Módulo de Créditos**: Permite el ingreso y/o actualización de los datos generales del cliente y garantes, así como de sus fuentes de ingreso (ya sea dependientes o independientes) y la administración de garantías. Permitirá además la administración de los créditos desde la solicitud hasta su cancelación, este módulo permitirá obtener reportes de resumen para comité de créditos e historial de los créditos que el cliente haya tenido con la institución, moratoria, perdón de recargos, manejo de crédito preferencial o

automático, calificación de los créditos de acuerdo a los atrasos acumulados por cuota, libreta de pagos, manejo de línea de créditos y comisiones, pagares y contratos etc. Deberá contar con una opción para parametrizar los diferentes criterios de calificación (días de mora, garantías, etc.).

Permitirá además obtener reportes Crediticios, incluyendo reportes de cartera (reportes de saldos, reportes para analistas, reportes de recuperación, etc.), reportes estadísticos (porcentajes de cartera por actividad económica, por sector económico, por destino, por plazo del crédito, por montos otorgados, etc.).

- **Módulo de Contabilidad**: Este módulo permitirá el ingreso y/o actualización del Plan de Cuentas, de la tabla de interfaces contables (relación entre las operaciones y sus correspondientes contabilizaciones), generación de asientos contables, reportes contables, reportes de administración de cartera.

- **Módulo de Comisiones**: Permitirá el ingreso y/o actualización de datos de los analistas de créditos, pagos de comisiones por las ventas de créditos, por cobranzas, records de producción, asignación y transferencia de cartera de créditos entre analistas, etc.

A efectos de satisfacer el requerimiento de poder proporcionar información en línea de las diferentes agencias distribuidas a nivel nacional, de definió que era necesario aprovechar de las ventajas que nos brinda la Internet y sus técnicas y arquitecturas subyacentes, como son: AJAX y Web Services. Elaborando para este propósito un esquema simplificado de la Topología a implementar.(ver Fig. Nº 24)

OBJETOS GENEXUS TRANSACCIONES IDENTIFICADOS: De las entrevistas sostenidas con los usuarios expertos en cada uno de los procesos se lograron identificar lo objetos Genexus Transacciones siguientes: Clientes, Solicitudes (de Crédito), Productos (diferentes tipos de créditos), Líneas de crédito, Intereses (diferentes valores), Analistas (encargados de la venta de los créditos, su evaluación, aprobación, seguimiento), Moneda, Garantes, Garantías, Oficinas (agencias ubicadas a nivel nacional), Departamentos (donde pertenecen las agencia y clientes), Plan de cuentas, Asientos

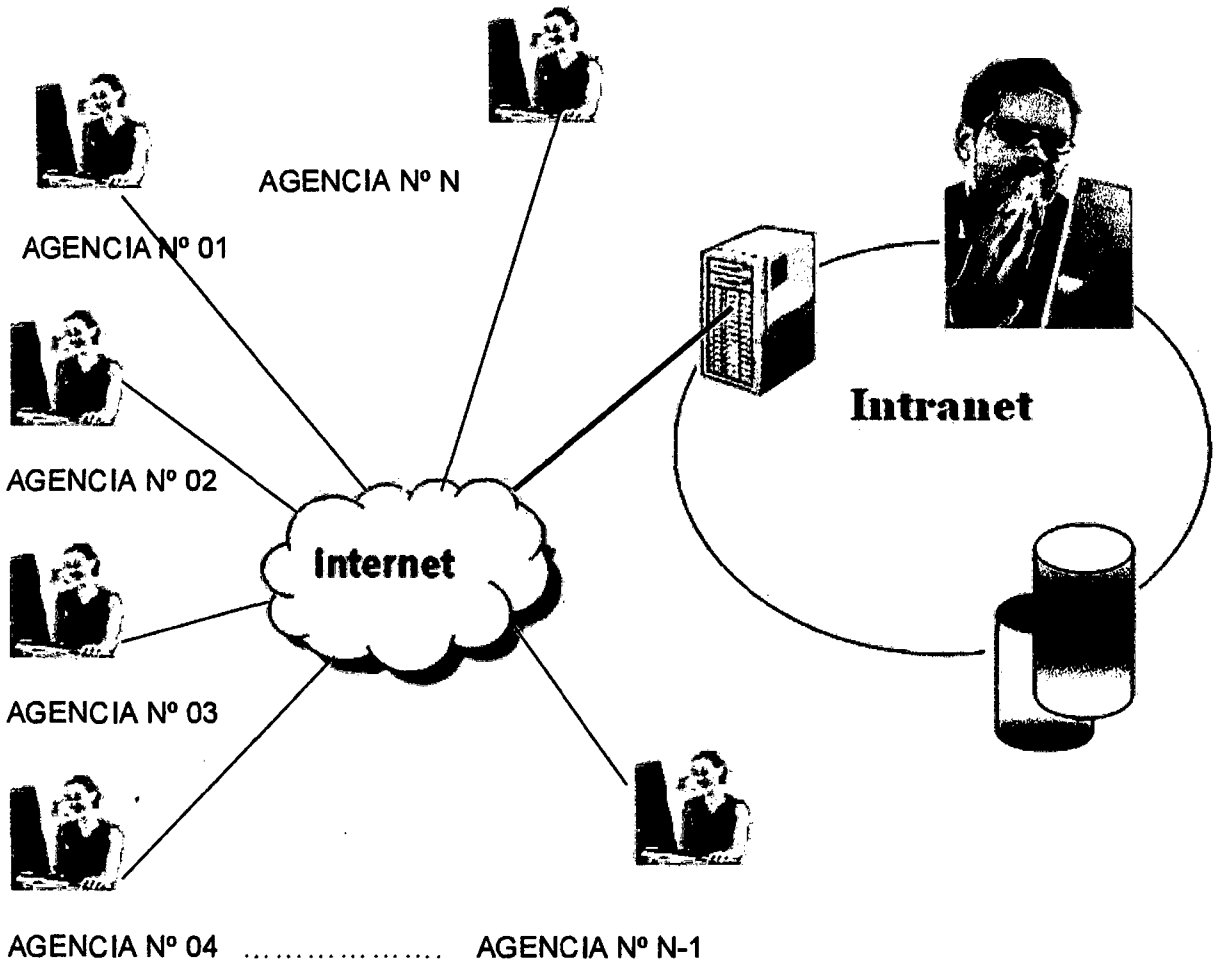


Figura Nº 24
Esquema simplificado de Topología a implementar

Fuente: Elaboración propia

contables, Interfases Contables (Relación entre operaciones y sus contabilizaciones correspondientes), Pagos (de cuotas de crédito, etc.), desembolsos (de créditos), Seguros, Usuarios, Niveles (de usuarios), Requerimientos (por cada tipo de producto crediticio), comisiones (por evaluación de créditos), operaciones,...

INTEGRACIÓN DE MODULOS

Teniendo en cuenta el requerimiento de contar con un sistema de software que apoye a los procesos de Créditos, Contabilidad y la Administración de los Analistas de Créditos, en forma integrada; es necesario implementar un

sistema de software integrado que integre los módulos antes mencionados, con una Base de Datos llamémosle corporativa, de la cual se pueda obtener información de gestión y gerencial que apoyen en la toma de decisiones.

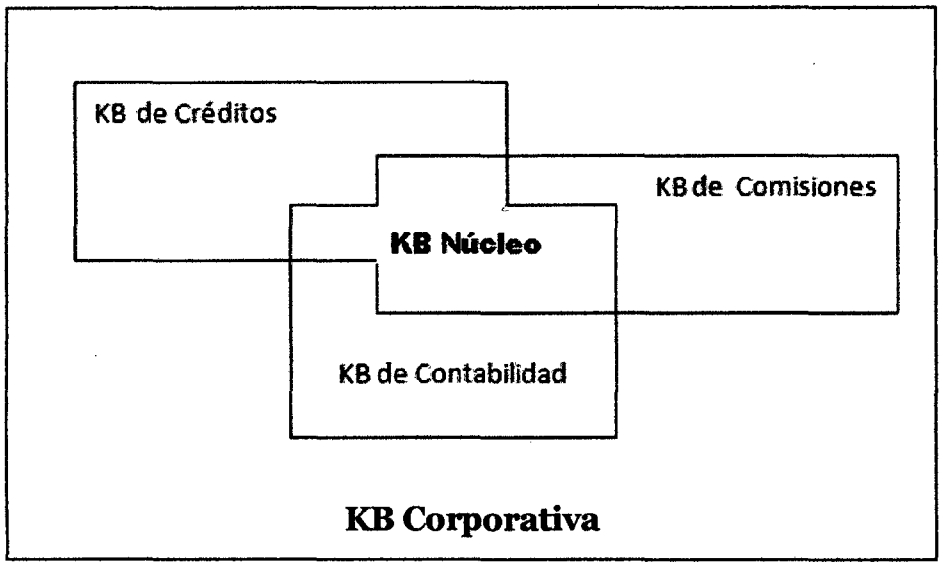
De acuerdo a la forma en que trabaja Genexus se ha considerado definir 5 Bases de conocimiento (KB): ver figura Nº 25

a. Base de Conocimiento **NÚCLEO**: Contiene los objetos de la empresa, que son compartidos por los diferentes módulos (aplicaciones).
b. Base de Conocimiento **CREDITOS**: Contiene el conocimiento del Módulo de créditos
c. Base de Conocimiento de **CONTABILIDAD**: Contiene el Conocimiento del Módulo de Contabilidad.
d. Base de Conocimiento de **COMISIONES**: Contiene el conocimiento de la administración del Módulo de comisiones a pagar a los Analistas de Créditos.
e. Base de Conocimiento **CORPORATIVA**: Contiene la consolidación de las bases de conocimiento los Módulos o aplicaciones de Créditos, Contabilidad y Administración de Analistas de Créditos y el Núcleo.
   La representación gráfica de las relaciones entre Módulos se puede apreciar en la figura siguiente:

En cada Base de Conocimiento (KB) (del Núcleo y de los Módulos) se deberá crear una Carpeta, y los objetos propios de dicha KB, deberán ubicarse dentro de esa Carpeta.

Es decir, en la **KB Núcleo** habrá que crear una Carpeta llamada Núcleo, en la **KB Créditos** será necesario crear una Carpeta llamada Créditos, en la **KB de Contabilidad** una Carpeta llamado Contabilidad, y en la **KB de Comisiones** una Carpeta llamada Comisiones.

Figura Nº 25. Esquema de Bases de Conocimiento y su relación



Fuente: Elaboración propia en Base a http://www.Genexus.com

La primera KB a ser definida será la **KB Núcleo**. Para ello habrá que identificar los objetos Genexus Transacciones, comunes a todos los módulos (por ejemplo, las transacciones Departamentos, Localidades, oficinas (Agencias), Productos, Clientes) y definirlos dentro de la **Carpeta Núcleo** de la **KB Núcleo**.

Una vez definida la Base de Conocimiento Núcleo, esta deberá distribuirse y consolidarse en cada una de las KBs asociadas a cada módulo o aplicación, para asegurarse el compartir todas los mismos objetos Genexus comunes, exactamente.

De modo que lo que se distribuirá será la **Carpeta Núcleo** de la **KB Núcleo**, y se consolidará en cada una de las KBs asociadas a una aplicación (**KB Créditos, KB Contabilidad y KB Comisiones**) así como en la **KB Corporativa**.

Recién luego de esto, cada desarrollador responsable de un módulo podrá comenzar a trabajar, debiendo crear todos los objetos propios de su módulo, en la Carpeta correspondiente a ese módulo en particular. Así, la **KB Créditos** tendrá 2 Carpetas: La **Carpeta Nucleo** con los objetos comunes a todos los módulos, y la **Carpeta Créditos** con los objetos propios que implementen ese módulo. Análogamente, las **KBs Contabilidad** y

Comisiones tendrán la **Carpeta Núcleo** y su **Carpeta propia**.

Cada KB asociada a una aplicación tendrá un ambiente de Prototipación, y un ambiente de Prueba en la plataforma de Producción. Estos modelos permitirán tener un diseño completo de la aplicación, minimizando los tiempos de desarrollo pues:

1. Se estará trabajando con KBs pequeñas, asegurando la integrabilidad con el resto de las aplicaciones

2. Se estarán realizando los primeros niveles de prueba de funcionalidad para el módulo en forma independiente asegurando un ciclo de prototipación dinámico

3. No se afectará al resto del desarrollo

Tan pronto que cada desarrollador de por finalizado su módulo, distribuirá la Carpeta propia de su KB (por ejemplo La **Carpeta Comisiones** de la **KB Comisiones**) y esta se consolidará en la **KB Corporativa**. No distribuirá la **Carpeta Núcleo** de la **KB Comisiones**, ya que esta **Carpeta Núcleo** solo se distribuye de la **KB Nucleo**).

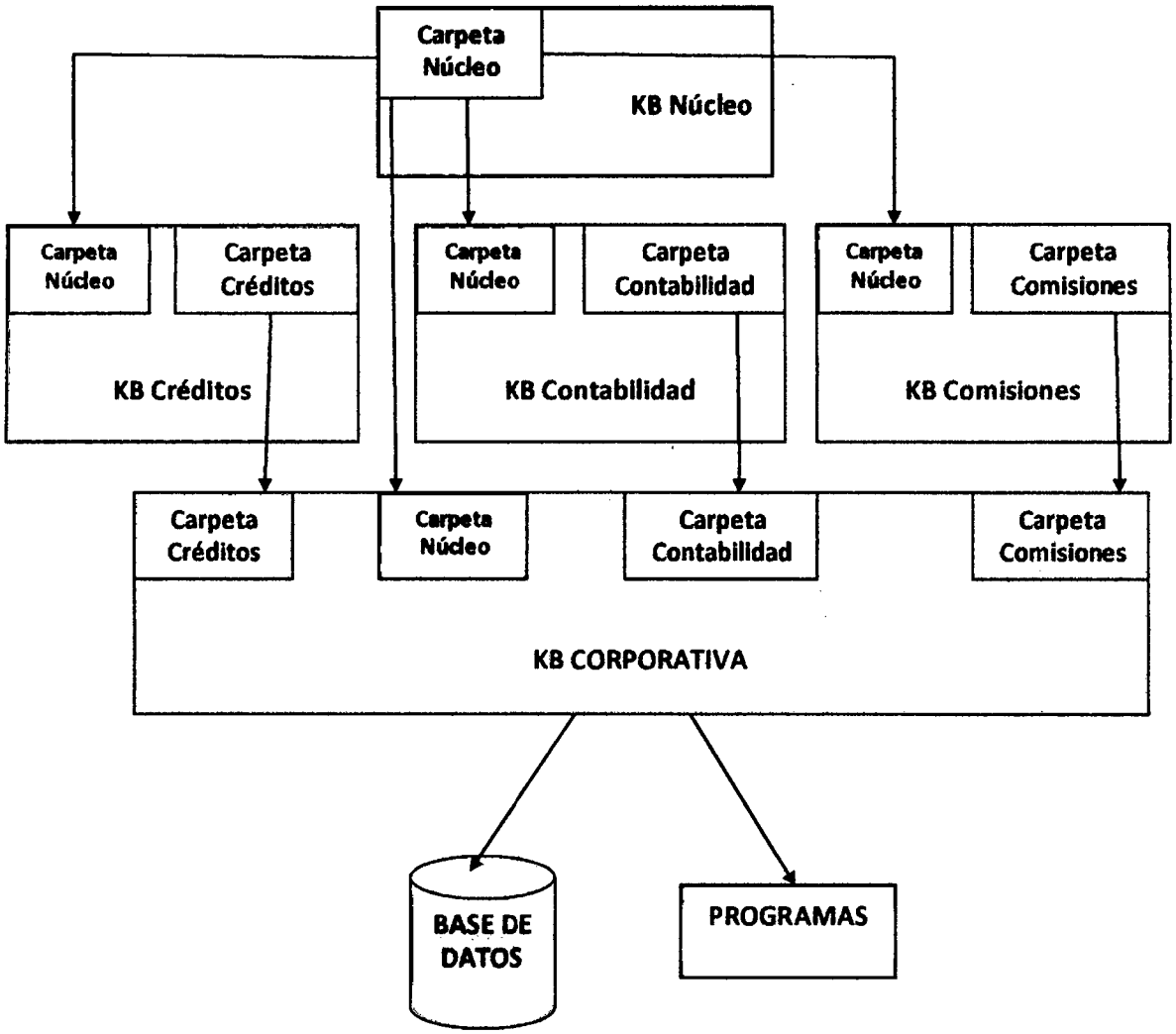
En la Figura Nº 26 se muestra un esquema de la distribución de los Módulos antes mencionados.

Si el desarrollador de un módulo se da cuenta que le resulta necesario para su aplicación, agregar un atributo en una transacción del Núcleo.

Esto tendrá un impacto en todas las KBs de las aplicaciones (o Módulos), y por tanto deberá administrarse con cuidado.

La forma de proceder es realizar el cambio en la **KB Núcleo**, y luego redistribuir el Núcleo (**Carpeta Núcleo** de la **KB Núcleo**) a todas las KBs.

Figura Nº 26. Distribución de Bases de Conocimiento de Módulos

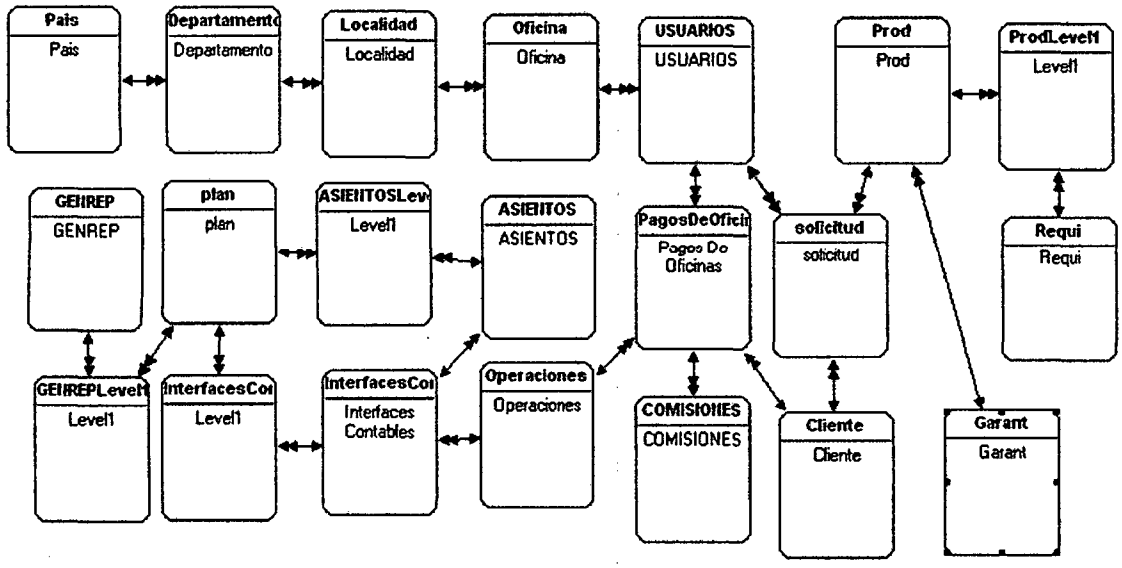


Fuente. Elaboración propia en base a http://www.genexus.com

5.2.2. FASE DE DISEÑO:

En la Etapa de **Análisis y Diseño** se identificaron las transacciones de las diferentes Bases de conocimiento y se incorporaron en la herramienta CASE Genexus, a continuación se muestra una versión del Diagrama de Bachmann del Modelo de Datos de la Base de Conocimiento corporativa generada haciendo uso de Genexus. Ver Fig.27

Figura Nº 27 – Modelo de Datos de Base de Conocimiento corporativa

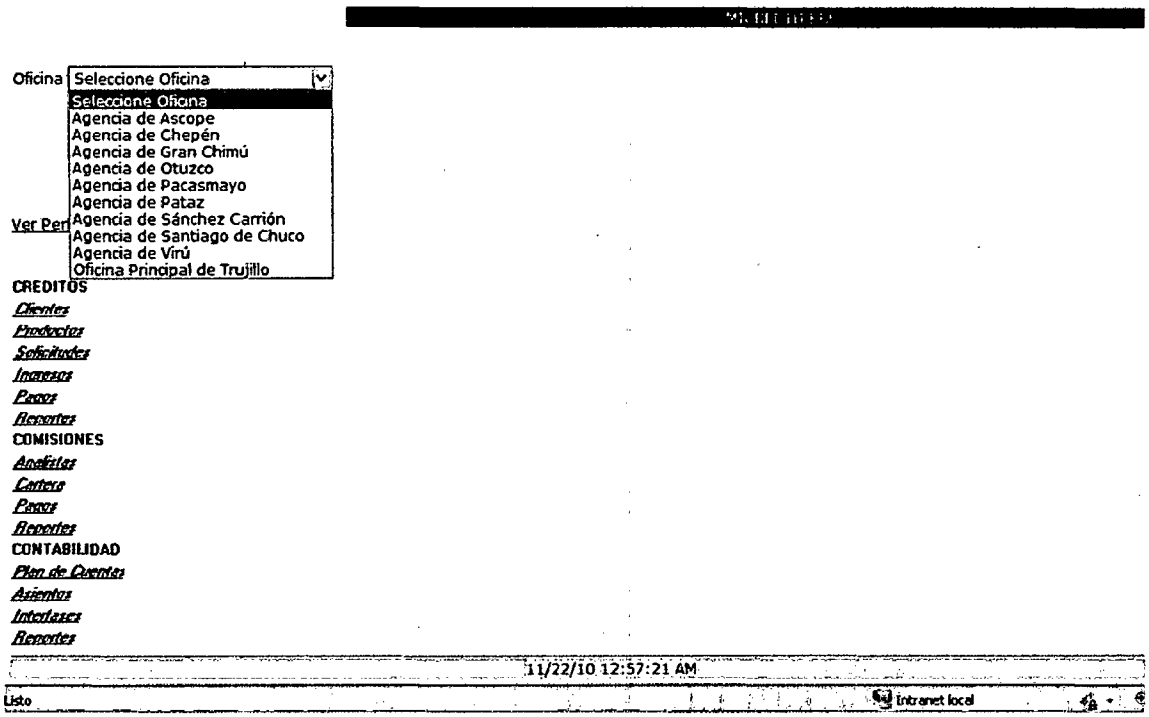


Fuente: Elaboración propia- Generada con Genexus

5.2.3. FASE DE PROTOTIPO.

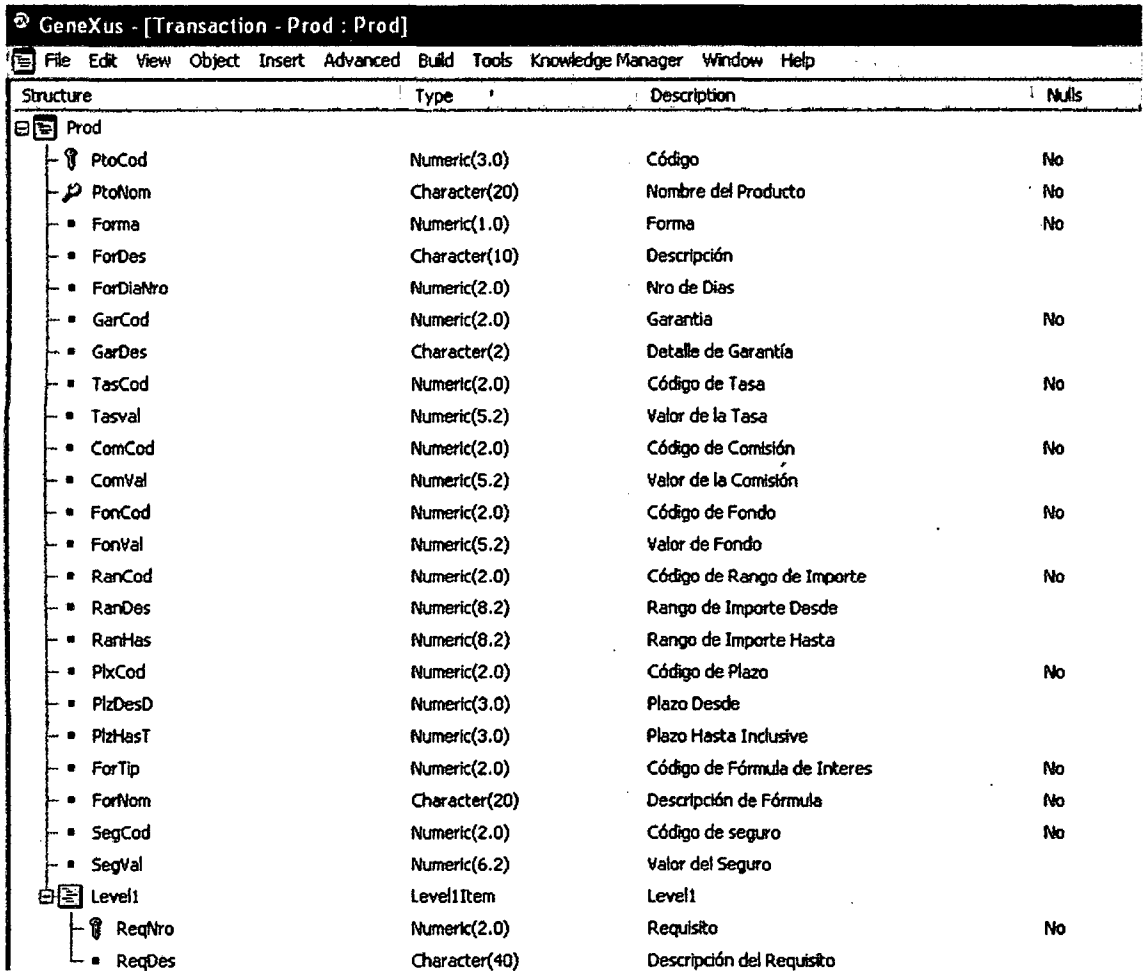
En la Etapa de **Prototipo** se definió la página principal de la aplicación la cual se muestra en la Fig. Nº 28

Figura Nº 28. Página Principal de la Aplicación Propuesta



Fuente: Elaboración propia usando CASE Genexus

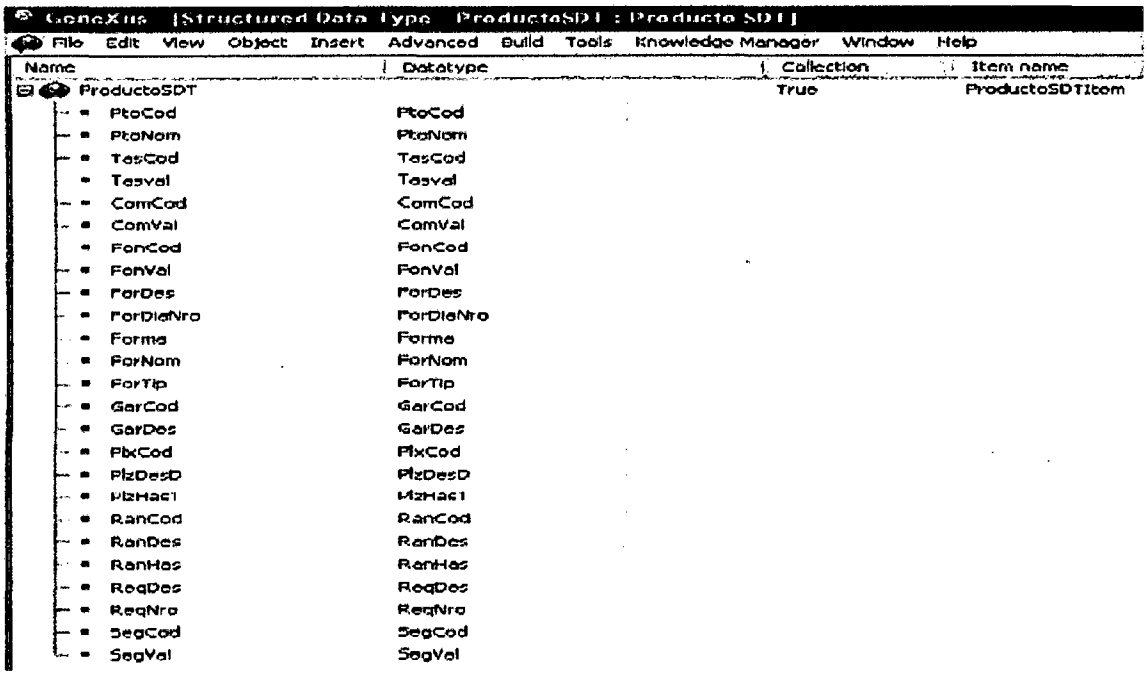
5.2.3.1. CREACION Y CONSUMO DE WEB SERVICES CON GENEXUS


GeneXus - [Transaction - Prod : Prod]

File Edit View Object Insert Advanced Build Tools Knowledge Manager Window Help

Structure	Type	Description	Nulls
Prod			
PtoCod	Numeric(3.0)	Código	No
PtoNom	Character(20)	Nombre del Producto	No
Forma	Numeric(1.0)	Forma	No
ForDes	Character(10)	Descripción	
ForDiaNro	Numeric(2.0)	Nro de Dias	
GarCod	Numeric(2.0)	Garantia	No
GarDes	Character(2)	Detalle de Garantía	
TasCod	Numeric(2.0)	Código de Tasa	No
Tasval	Numeric(5.2)	Valor de la Tasa	
ComCod	Numeric(2.0)	Código de Comisión	No
ComVal	Numeric(5.2)	Valor de la Comisión	
FonCod	Numeric(2.0)	Código de Fondo	No
FonVal	Numeric(5.2)	Valor de Fondo	
RanCod	Numeric(2.0)	Código de Rango de Importe	No
RanDes	Numeric(8.2)	Rango de Importe Desde	
RanHas	Numeric(8.2)	Rango de Importe Hasta	
PlxCod	Numeric(2.0)	Código de Plazo	No
PlzDesD	Numeric(3.0)	Plazo Desde	
PlzHasT	Numeric(3.0)	Plazo Hasta Inclusive	
ForTip	Numeric(2.0)	Código de Fórmula de Interes	No
ForNom	Character(20)	Descripción de Fórmula	No
SegCod	Numeric(2.0)	Código de seguro	No
SegVal	Numeric(6.2)	Valor del Seguro	
Level1	Level1Item	Level1	
ReqNro	Numeric(2.0)	Requisito	No
ReqDes	Character(40)	Descripción del Requisito	

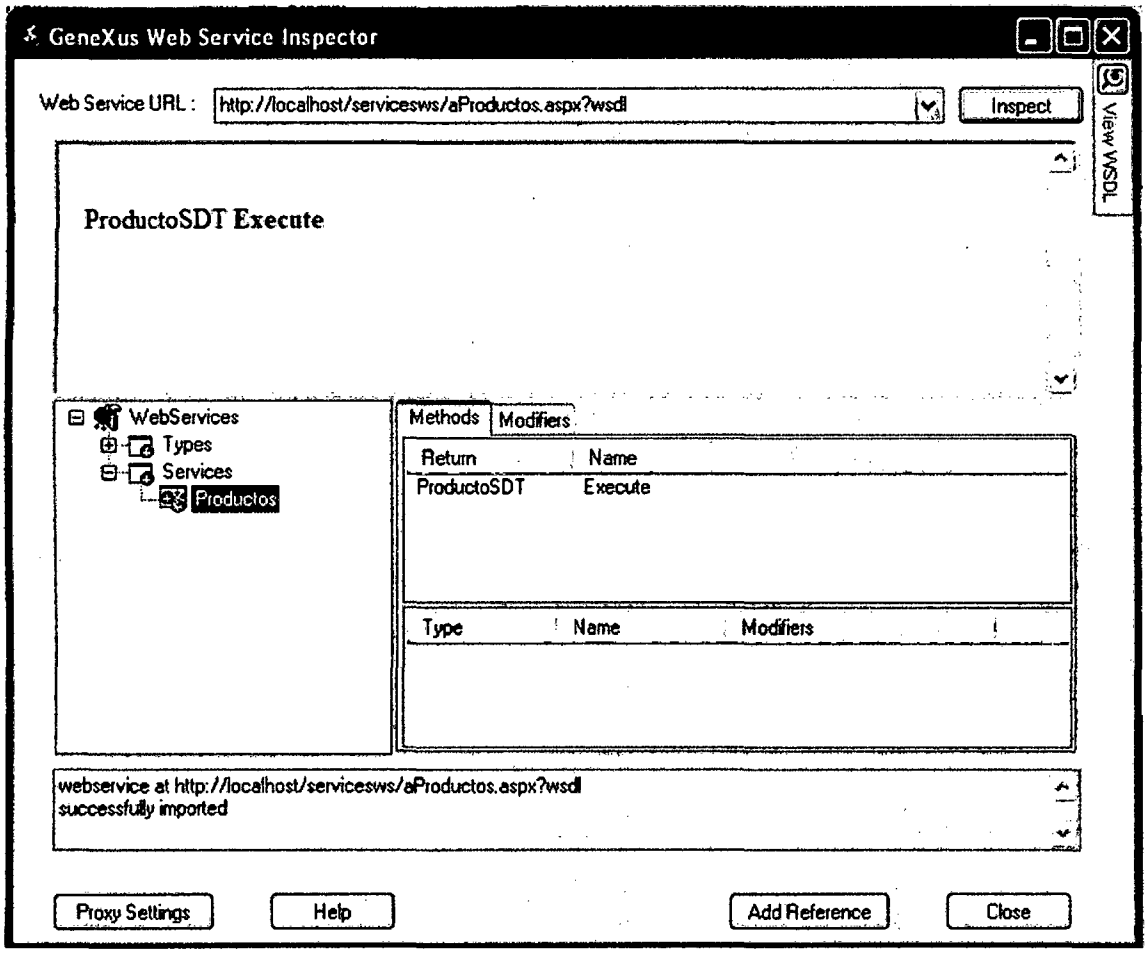
En base a la Transaction PROD (productos), vamos a definir un Web Services al que llamaremos PRODUCTOS.


GeneXus [Structured Data Type ProductoSDT : ProductoSDT]

File Edit View Object Insert Advanced Build Tools Knowledge Manager Window Help

Name	Datatype	Collection	Item name
ProductoSDT		True	ProductoSDTItem
PtoCod	PtoCod		
PtoNom	PtoNom		
TasCod	TasCod		
Tasval	Tasval		
ComCod	ComCod		
ComVal	ComVal		
FonCod	FonCod		
FonVal	FonVal		
ForDes	ForDes		
ForDiaNro	ForDiaNro		
Forma	Forma		
ForNom	ForNom		
ForTip	ForTip		
GarCod	GarCod		
GarDes	GarDes		
PlxCod	PlxCod		
PlzDesD	PlzDesD		
PlzHasT	PlzHasT		
RanCod	RanCod		
RanDes	RanDes		
RanHas	RanHas		
ReqDes	ReqDes		
ReqNro	ReqNro		
SegCod	SegCod		
SegVal	SegVal		

PRODUCTOSDT será una colección



En el campo "Web Service URL" colocaremos la siguiente URL http://localhost/servicesws/aProductos.aspx?wsdl

Y luego haremos click en botón **"Inspect"**, la pantalla nos muestra que el método del Web Service es "Execute" y el tipo de datos que devuelve es "ProductoSDT".

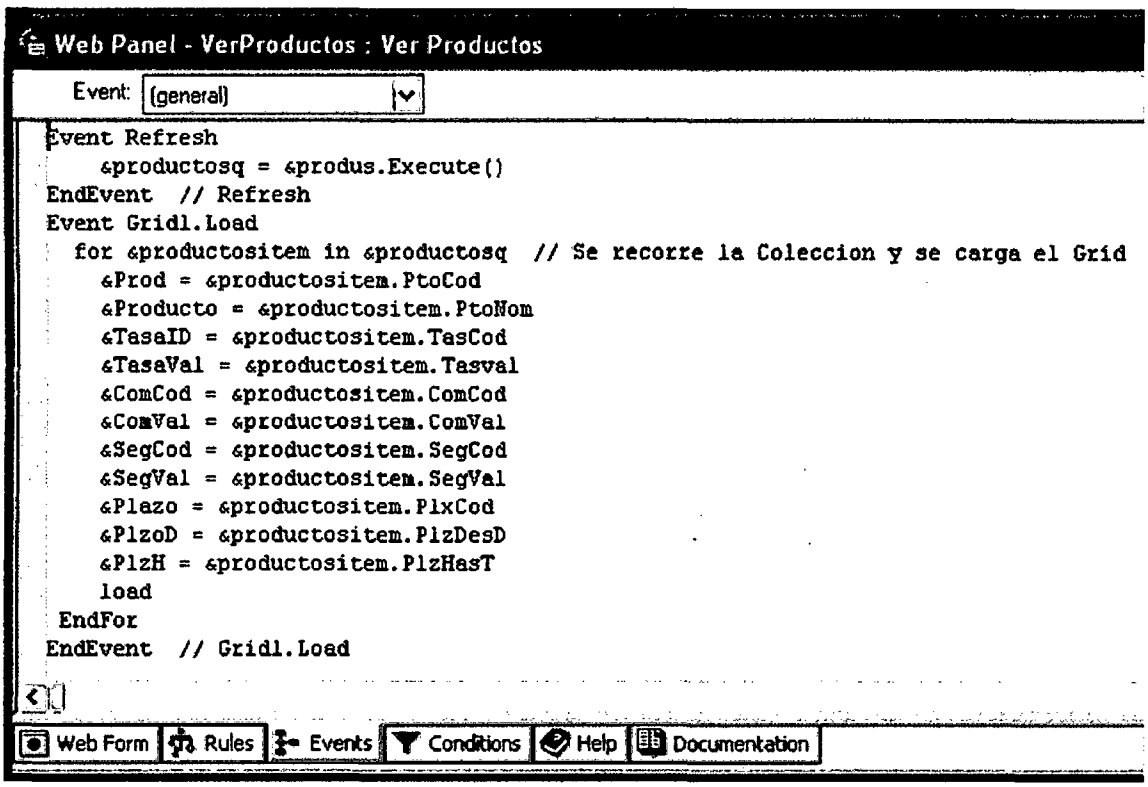
Cuando se hace click en el botón "Add Reference" se definen los tipos de datos además de toda la información necesaria para poder consumir el Web Service.

En Genexus pasamos a la etapa de prototipo forzando el impacto.

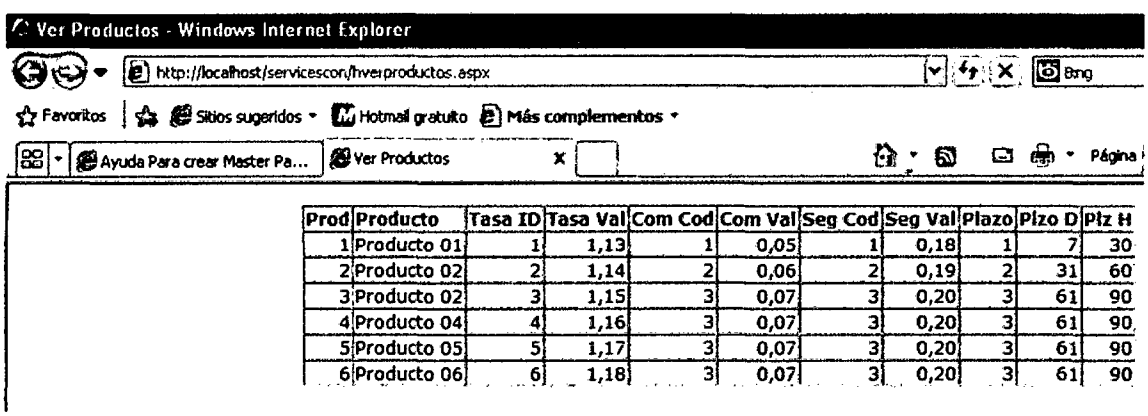
4. Crearemos el objeto Genexus Web Panel al que denominaremos **VerProductos,** para eso definiremos las variables siguientes: &Productosq y &ProductosItem

Se observa que los tipos de datos ProductoDST (colección de Productos), ProductoSDT_ProductoSDTItem (cada Item de la colección) y el Web Service **productos** se crearon automáticamente al inspeccionar el servicio con el **WSDL Inspector.**

El Form del Web Panel **VerProductos** contendrá un grid donde mostrará las variables conteniendo la información de interés y se deberá de programar los eventos **Refresh** y **load** del Web Panel, como se muestra a continuación:

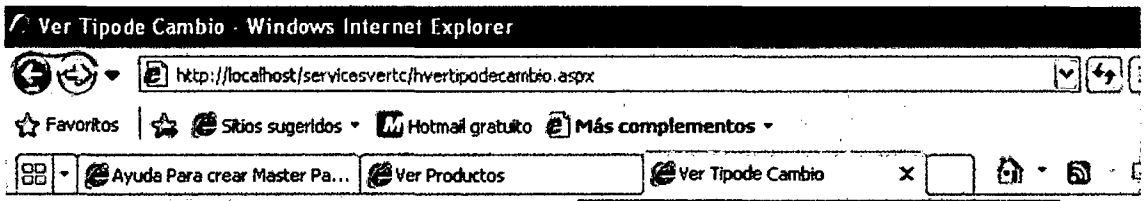


Finalmente ejecutamos el Web Service y obtenemos pantalla mostrando la información siguiente:



Prod	Producto	Tasa ID	Tasa Val	Com Cod	Com Val	Seg Cod	Seg Val	Plazo	Plzo D	Plz H
1	Producto 01	1	1,13	1	0,05	1	0,18	1	7	30
2	Producto 02	2	1,14	2	0,06	2	0,19	2	31	60
3	Producto 02	3	1,15	3	0,07	3	0,20	3	61	90
4	Producto 04	4	1,16	3	0,07	3	0,20	3	61	90
5	Producto 05	5	1,17	3	0,07	3	0,20	3	61	90
6	Producto 06	6	1,18	3	0,07	3	0,20	3	61	90

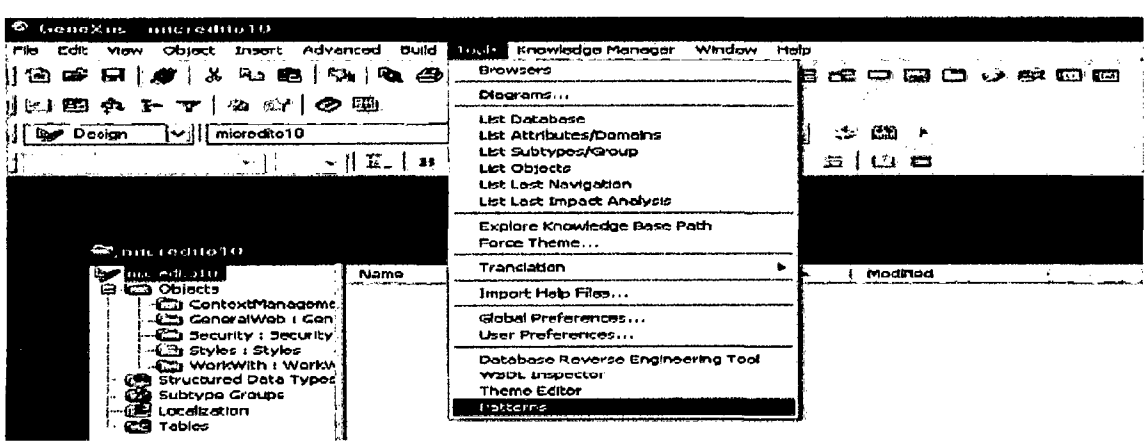
Siguiendo el mismo procedimiento anterior, se construyo un Web Service para publicar el Tipo de Cambio manejado por MicreditPeru, el mismo que se muestra a continuación:

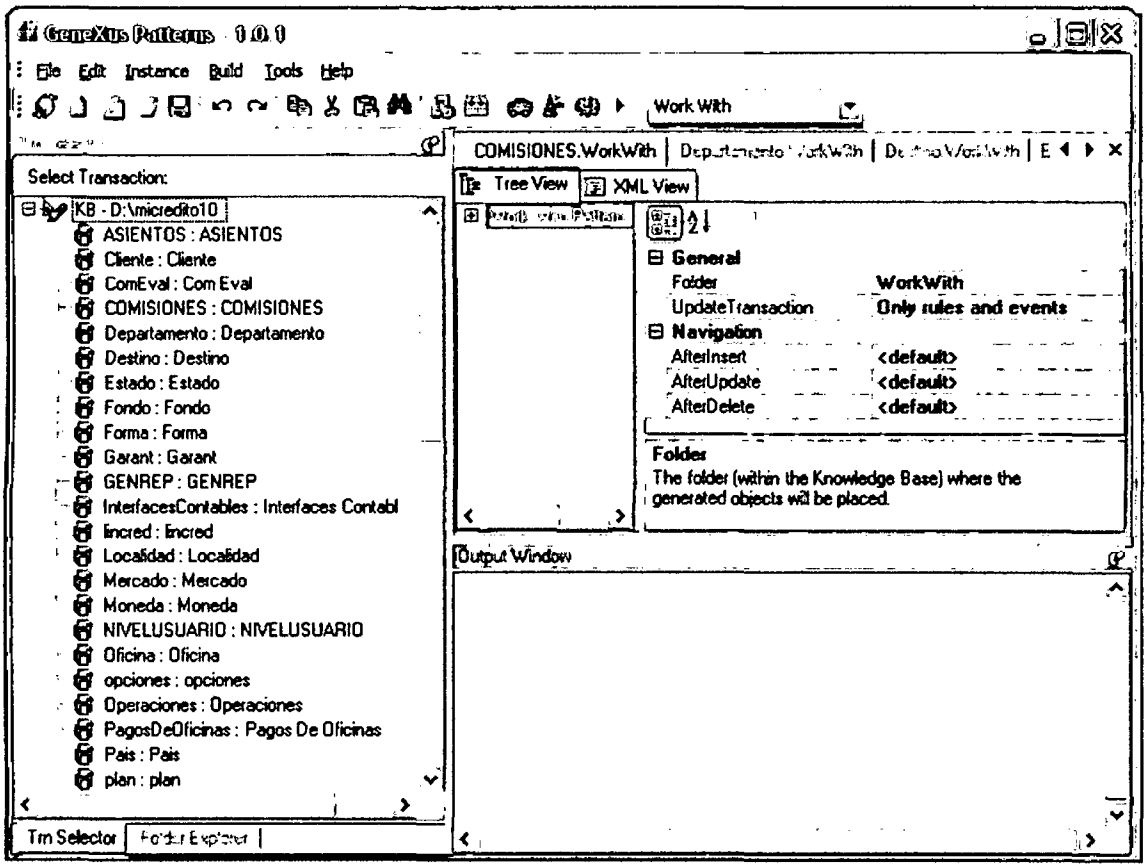


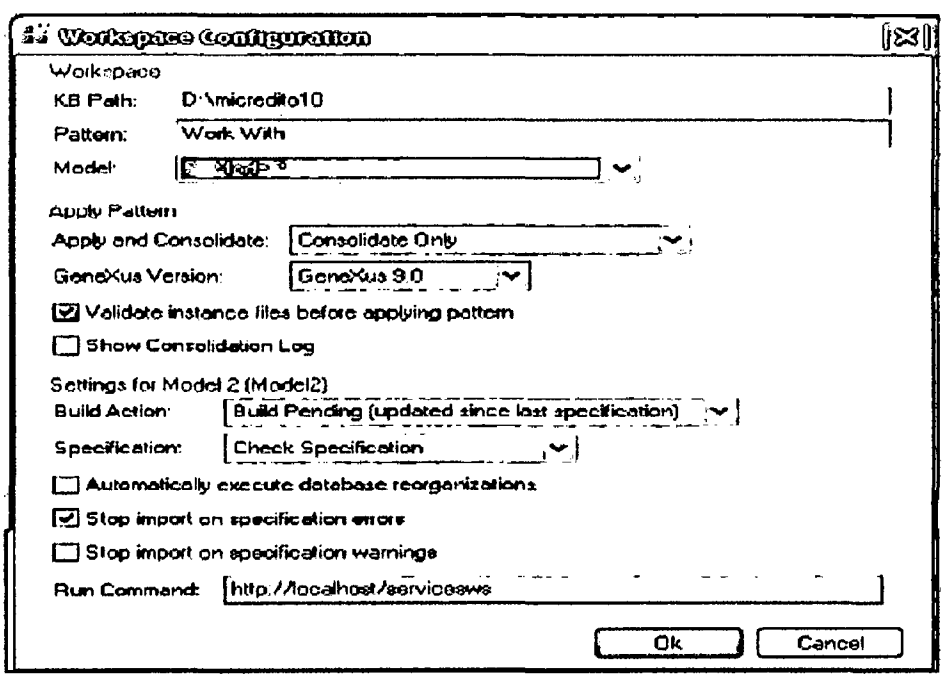
Mon Cod	Mon Nom	Mon Vig Fec	Mon Cam Val
2	Dolares	10/01/10	2.84
2	Dolares	09/01/10	2.82
2	Dolares	08/01/10	2.81
2	Dolares	07/01/10	2.80
2	Dolares	06/01/10	3.01
2	Dolares	05/01/10	3.02
2	Dolares	04/01/10	3.03
2	Dolares	03/01/10	3.04
2	Dolares	02/01/10	3.05
2	Dolares	01/01/10	3.06
3	Euros	10/01/10	3.97
3	Euros	09/01/10	3.94
3	Euros	08/01/10	3.93
3	Euros	07/01/10	3.94
3	Euros	06/30/10	3.95
3	Euros	05/01/10	3.96
3	Euros	04/01/10	3.97
3	Euros	03/01/10	3.98
3	Euros	02/01/10	3.99
3	Euros	01/01/10	4.00

5.2.3.2. APLICACIÓN DE PATRONES CON GENEXUS

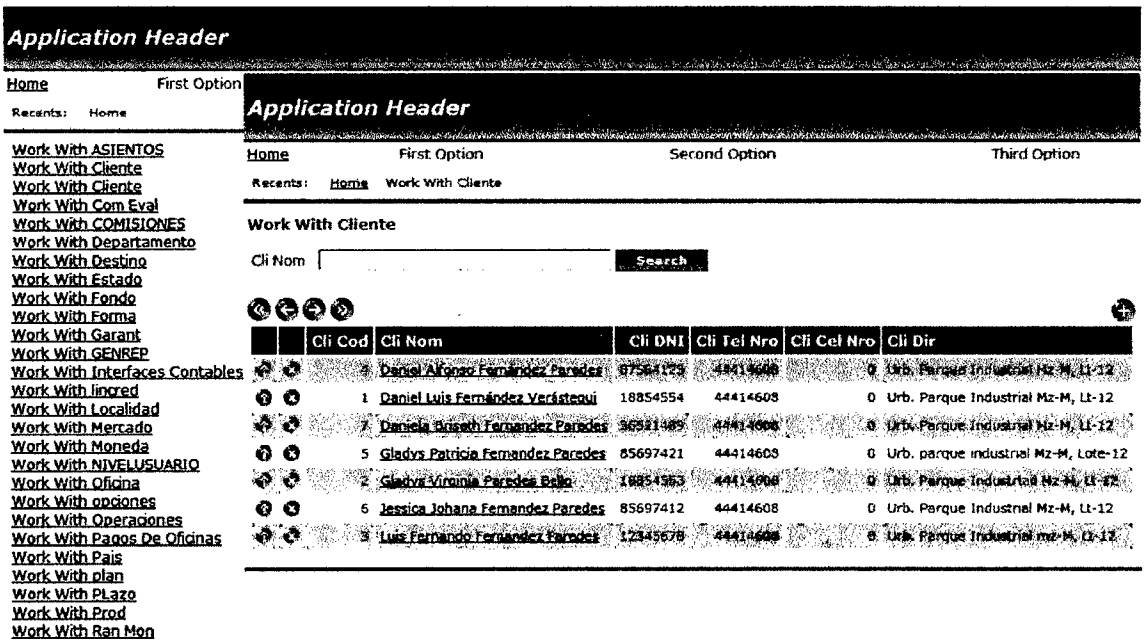
En esta sección aplicaremos el concepto de patrón o pattern **"Work With"** a todas las tranasacciones de la Base de Conocimiento corporativa.

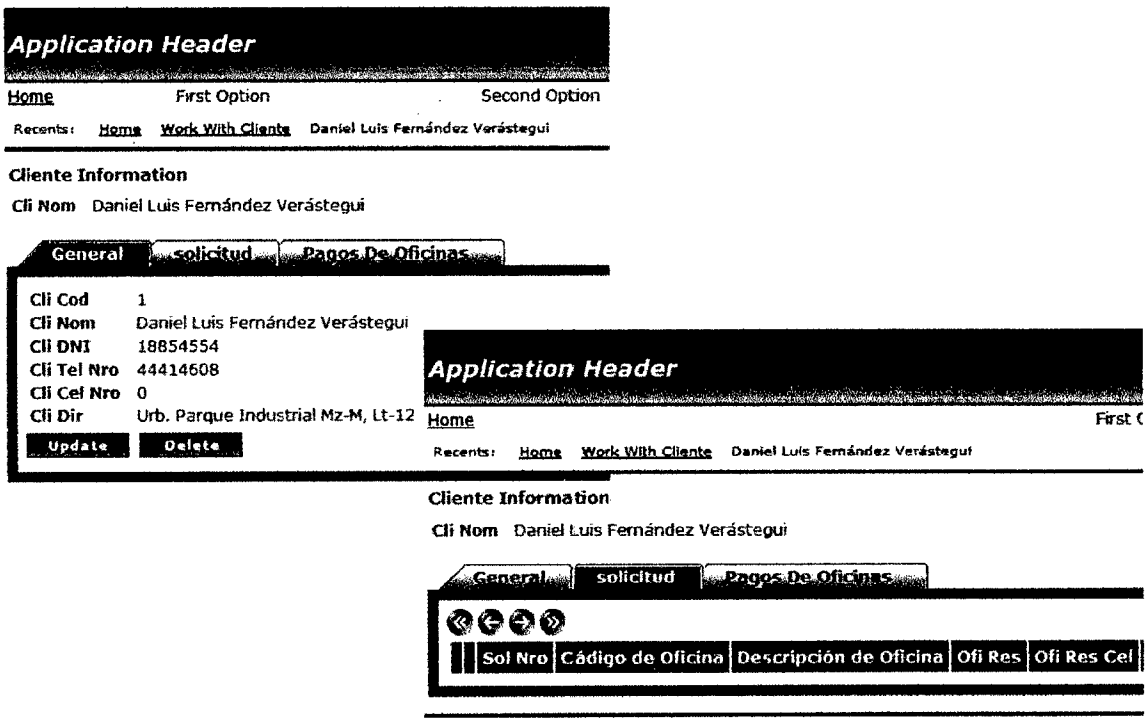



GeneXus Patterns
File Edit Instance Build Tools Help
Work With
Select Transaction:
KB - D:\micredito10
ASIENTOS : ASIENTOS
Cliente : Cliente
ComEval : Com Eval
COMISIONES : COMISIONES
Departamento : Departamento
Destino : Destino
Estado : Estado
Fondo : Fondo
Forma : Forma
Garant : Garant
GENREP : GENREP
InterfacesContables : Interfaces Contabl
lincred : lincred
Localidad : Localidad
Mercado : Mercado
Moneda : Moneda
NIVELUSUARIO : NIVELUSUARIO
Oficina : Oficina
opciones : opciones
Operaciones : Operaciones
PagosDeOficinas : Pagos De Oficinas
Pais : Pais
plan : plan
Trn Selector
COMISIONES.WorkWith
Tree View
XML View
General
Folder
WorkWith
UpdateTransaction
Only rules and events
Navigation
AfterInsert
<default>
AfterUpdate
<default>
AfterDelete
<default>
Folder
The folder (within the Knowledge Base) where the generated objects will be placed.
Output Window


Workspace Configuration
Workspace
KB Path:
D:\micredito10
Pattern:
Work With
Model:
Apply Pattern
Apply and Consolidate:
Consolidate Only
GeneXus Version:
GeneXus 9.0
Validate instance files before applying pattern
Show Consolidation Log
Settings for Model 2 (Model2)
Build Action:
Build Pending (updated since last specification)
Specification:
Check Specification
Automatically execute database reorganizations
Stop import on specification errors
Stop import on specification warnings
Run Command:
http://localhost/aservicesws
Ok
Cancel

RESULTADO DE LA EJECUCION DE PATTERN O PATRON WORK WITH DE GENEXUS.





CAPITULO VI

ANALISIS DE RESULTADOS

6.1. ANALISIS DE RESULTADOS

En este capítulo, tomando como base la Metodología de Investigación descrita en el capítulo III, se analizan los resultados obtenidos de la aplicación de la Metodología MDDS-GX propuesta, en los casos de estudio descritos en el capítulo V, de este trabajo de Tesis de Maestría. En primer lugar, en la sección 6.1.1, se analiza el logro de los objetivos planteados al inicio de este trabajo, y que han sido presentados en la sección 1.2. del capítulo I. y en la sección 6.1.2, se indican los principales aportes de esta Tesis de Maestría.

6.1.1. ANALISIS DEL LOGRO DE OBJETIVOS

En la sección 1.2 del capítulo I de esta Tesis, se planteó el objetivo general de este trabajo de investigación: "Especificar una Metodología Ágil para Desarrollo de Software, apoyada en la Herramienta CASE GENEXUS, que cumpla con los principales requisitos que debe tener una Metodología de Desarrollo de Software, que permita superar o mitigar las limitaciones de las Metodologías de Desarrollo de Software actuales, desarrollar software que satisfaga los requerimientos funcionales y no funcionales de la empresa PYME de microfinanzas MICREDITPERU, así como, los requerimientos generalmente aceptados de las PYMES de microfinanzas de la localidad de Trujillo – Perú. ". Para lograr este objetivo, se plantearon un conjunto de objetivos específicos. A continuación, se pasa a analizar el resultado obtenido para cada uno de ellos:

1. Con respecto al **objetivo específico 1**. En el momento de construir una Metodología, se han de considerar unos requisitos deseables.

1.1. La Metodología debe ajustarse a los objetivos.

La Metodología MDDS-GX especificada tuvo como punto de partida los objetivos de eliminar o mitigar las desventajas que presentan las metodologías de desarrollo de software actuales (sección 4.1.1.) y satisfacer los requerimiento de la Empresa MICREDITPERU y los de las empresas de microfinanzas de la localidad de Trujillo - Perú (sección 5.2.1.)

1.2. **La Metodología debe cubrir el ciclo entero de desarrollo de software.** La Metodología propuesta, cubre el ciclo completo del desarrollo de software, (sección 4.2.2.) desde la identificación de requerimientos, la captura de las visiones de los usuarios, definición de los objetos Genexus, la etapa de diseño de la base de datos, y del prototipo de las interfases, la implementación o generación de programas fuentes, las pruebas, la obtención del modelo de producción, la implementación y el mantenimiento de las bases de datos y de los programas de la aplicación.

1.3. **La Metodología debe integrar las distintas fases del ciclo de desarrollo.**

_ **Rastreabilidad.** La Metodología propuesta presenta (sección 4.2.2.) bucles que van del estudio del sistema de software (requerimientos) al modelo de diseño, del modelo de diseño al modelo de prototipo, del modelo de diseño al modelo de producción y del modelo de producción al de implementación. Permitiendo llevar a cabo pruebas, conjuntamente con los usuarios, y poder llevar a cabo modificaciones tanto los datos como los programas.

_ **Fácil interacción entre etapas del ciclo de desarrollo**. Por ser una metodología evolutiva e incremental, permite ir incorporando requerimientos previamente priorizados al modelo de diseño, prototipo y producción y una vez que están probados y validados, llevar a cabo su implementación.

1.4. **La Metodología debe incluir la realización de validaciones.** En la Metodología propuesta, (sección 4.2.2.) las validaciones son llevadas a cabo por los usuarios expertos en los procesos a los cuales apoyará la aplicación, quienes conjuntamente con el analista Genexus, llevará a cabo las pruebas de los avances en el desarrollo de la aplicación y de ser necesario indicará se realicen las modificación que requieran, hacer uso de la herramienta CASE Genexus, las modificaciones se harán en forma automática, no corrigiendo los programas (parchándolos) sino volviéndolos a generar como nuevos.

1.5. **La Metodología debe soportar la determinación de la exactitud del sistema a través del ciclo de desarrollo**. Debido a que la Metodología

tiene como base la estrecha comunicación entre usuarios y analistas Genexus y que además según el ciclo de vida (sección 4.2.2.) la Metodología parte de la identificación de las visiones de los usuarios que no son otra cosa que aquellos datos, reglas del negocio, reportes, consultas con las que el usuario está familiarizado, facilita que las considere en las pruebas y validaciones.

1.6. **La Metodología debe ser la base de una comunicación efectiva.** Debido a que en la Metodología propuesta (principio Nº 1 de sección 4.2.1) el usuario tiene un papel activo, esto es esta involucrado en el logro de los avances de la aplicación y conoce las reglas de negocio y los datos e información a obtener en cada incremento de los requerimiento, es factible la gestión de los analistas Genexus. Debido también que la metodología tiene como base la comunicación efectiva entre usuario y analista Genexus.

1.7. **La Metodología debe funcionar en un entorno dinámico orientado al usuario a lo largo de todo el ciclo de vida del desarrollo se debe producir una transferencia de conocimientos hacia el usuario.**
Teniendo en cuenta que la Metodología MDDS-GX hace uso de prototipos (sección 4.2.2.3.) para que el usuario pueda probar las aplicaciones que conjuntamente con el analista Genexus va construyendo, probando e implementando, la comunicación se hace más dinámica y los ajustes necesarios son solicitados y obtenidos más rápidamente(principio Nº 15, sección 4.2.1.).

1.8. **La Metodología debe especificar claramente los responsables de resultados.** El equipo mínimo (principio Nº 18, sección 4.2.1.) para desarrollar una aplicación haciendo uso de la Metodología MDDS-GX está conformado por dos personas un usuario experto en los procesos del negocio y un analista Genexus, quienes serán los responsables de obtener el software.

1.9. **La Metodología debe poder emplearse en un entorno amplio de proyectos de desarrollo de software.**

- **Variedad**. La Metodología propuesta permite desarrollar aplicaciones Win y Web, la misma que deberá adecuarse a los requerimientos particulares de cada aplicación.

- **Tamaño, ciclo de vida**. La Metodología MDDS-GX apoyada con la herramienta CASE Genexus propuesta, permite desarrollar aplicaciones corporativas integrando las diferentes aplicaciones de una empresa y conformando varios equipos de trabajo.(sección 5.2.1.).

- **Complejidad.** La Metodología MDDS-GX propuesta sirve para desarrollar sistemas de software de distinta complejidad, es decir puede comprender el desarrollo de software para un departamento, varios departamentos o diversas empresas (secciónes 5.2.1., 5.2.2. y 5.2.3.).

- **Entorno**. Debido a que la Metodología MDDS-GX propuesta está apoyada por la herramienta Genexus, la cual está basada en la administración del conocimiento del negocio, que tienen los usuarios de una empresa, este conocimiento es independiente de la plataforma de desarrollo y puede fácilmente implementarse en diversas plataformas en forma independiente de estas (es parametrizable como se vio en paso 6 de sección 5.1.1.2.2.).

1.10. La Metodología se debe de poder enseñar. Debido a que la Metodología propuesta no hace uso de diagramas que pudieran dificultar su aprendizaje y que más bien hace uso de objetos Genexus con los cuales el usuario está familiarizado, como son las transacciones, los reportes, los web panels, ..., (secciónes 4.2., 4.2.2.2., 4.2.2.3. y 4.2.2.4.) es que esta es captada fácilmente por ellos y debido también a que la curva de aprendizaje en el uso de la herramienta que automatiza su ciclo de desarrollo es corta, hace que despierte un mayor interés en su aprendizaje.

1.11. La Metodología debe estar soportada por herramientas CASE. La Metodología propuesta, esta soportada por la herramienta CASE Genexus, (sección 4.2.) la cual automatiza todo el ciclo de desarrollo del software incluyendo la creación y mantenimiento de la Base de Datos, reduciendo drásticamente los tiempos de desarrollo, minimizando los costos en personal de desarrollo y sobre todo los costos en mantenimiento de los sistemas de software, toda vez que automatiza la

generación de los programas que hayan sufrido modificaciones por el mantenimiento propio de los cambios en las necesidades propias del negocio.

1.12. **La Metodología debe soportar la eventual evolución del sistema.** La herramienta CASE Genexus que apoya a la metodología propuesta está basada en la administración del conocimiento del negocio, la misma que puede actualizarse en función a la necesidad de dar respuesta a la competencia o a los cambios naturales en el contexto, esto le permite poder implementar este conocimiento en cualquier otra plataforma tecnológica en forma independiente. Por otro lado el mantenimiento es automático, esto es, por ejemplo, si se dan cambios en la Base de Datos, la herramienta CASE Genexus, (sección 5.1.1.5.1.) permite llevar a cabo un análisis de impacto, identificando aquellos programas que serán reemplazados por otros programas (nuevos) adecuados los mismos que serán generados automáticamente, que consideran los cambios en la Base de Datos.

1.13. **La Metodología debe contener actividades conducentes a mejorar el proceso de desarrollo de software.** Está pendiente la incorporación de aplicaciones que permitan obtener automáticamente métricas para poder hacer estimaciones que evidencien la efectividad de la aplicación del proceso con relación a cualquier producto software resultante de la aplicación del proceso y mejorarlo.

La Metodología propuesta, por ser nueva, aún no dispone de un conjunto de mediciones de proceso para identificar la calidad y coste asociado a cada etapa del proceso. Sería ideal que estas estén automatizadas e incorporadas al mismo nivel de la herramienta CASE Genexus.

1.14. **La Metodología debe permitir desarrollar software que satisfaga los requerimientos funcionales y no funcionales de la empresa PYME de microfinanzas MICREDITPERU, así como, los requerimientos generalmente aceptados de las PYMES de microfinanzas de la localidad de Trujillo – Perú.**

La Metodología de desarrollo propuesta permite satisfacer los requerimientos de la Empresa MICREDITPERU y los requerimientos generalmente aceptados por las empresas de microfinanzas de la localidad de Trujillo(secciónes 5.2.1., 5.2.2.., 5.2.3., 5.2.3.1. y 5.2.3.2.).

1.15. La Metodología debe superar o mitigar las limitaciones de las metodologías de desarrollo de software actuales.

La Metodología propuesta mitiga o elimina las limitaciones que presentan la principales metodologías ágiles actuales estudiadas (secciones 2.1.2.8.) , en cuanto a la oportunidad y calidad del desarrollo de software, toda vez que se apoya en la herramienta CASE Genexus, la cual cubre todo el ciclo de vida del desarrollo de software, incluyendo el mantenimiento automático del software, actividad que al llevarlo a cabo en forma llamémosle artesanal, insume una gran cantidad de recursos de programación, limitando el desarrollo de aplicaciones nuevas e incrementando los costos y lo que es peor el no poder contar con un software actualizado en forma oportuna.

Como resultado de lo aprendido en la construcción de los casos de estudio del capitulo V, las limitaciones o desventajas de las principales Metodologías ágiles actuales, que se han eliminado o mitigado, son:

a. La falta de documentación del diseño. La base de conocimiento conteniendo los objetos Genexus transacciones, de la cual Genexus deriva el diseño de la bases de datos y programas de mantenimiento, está permanentemente actualizada. Lo mismo ocurre con los demás objetos Genexus como son los reportes, las pantallas (objetos Genexus work panels y web panels) y los procedimientos. Así mismo Genexus facilita el mantener documentación descriptiva actualizable en línea.

b. Los problemas derivados de la comunicación oral. Este tipo de problema se ha superado al tener almacenado y permanentemente actualizada la base de conocimiento.

c. Falta de calidad. No existe necesidad de probar el código, toda vez que este es generado automáticamente con cero errores, por los diferentes generadores de programas fuente con que cuenta Genexus, como son: .NET (C#), java, Ruby, RPG, Cobol, Visual Foxpro y Visual Basic. Este tipo

de limitación se ha mitigado debido a la generación de prototipos, los cuales de acuerdo a la Metodología MDDS-GX, son continuamente probados por los usuarios y los analistas Genexus, después de cada iteración, ya sea a nivel de la fase de diseño, prototipo o producción.

d. La fuerte dependencia de las personas. Al contar los resultados de la fase de estudio de la metodología propuesta (sección 4.2.2.1.), con la base de conocimiento y con la documentación en línea, esta limitación se ha mitigado, no obstante, siempre existirá cierta dependencia de las personas que tengan experiencia en las aplicaciones, mas ya no será tan crítica, como el no contar con documentación alguna.
e. Falta de procesos de revisión del código. Esta limitación se ha eliminado, como se mencionó en el punto c), al estar la Metodología MDDS-GX apoyada por la herramienta case Genexus, esta genera automáticamente el código de los programas con cero errores.
f. Falta de reusabilidad. Esta limitación se ha mitigado al existir la base de conocimiento, el poder usar componentes web y aplicar patrones (sección 5.2.3.2.).
g. Sobre costos y retrasos derivados de la refactorización continua. Esta limitación se ha eliminado, por los mismos motivos dados en el punto c)
h. Restricciones en cuanto a tamaño de los proyectos abordables. Esta limitación se ha mitigado, toda vez que la Metodología MDDS-GX facilita el desarrollo de sistemas de software a nivel de toda una empresa (sección 5.2.1.).
i. Rigidez. Esta limitación se ha mitigado, no obstante que el uso de la Metodología MDDS-GX, propuesta exige expertiz tanto por parte del usuario en los procesos del negocio, como por parte del analista Genexus en la aplicación de la herramienta.
j. Los Cambios en el modelo de datos son "pesados". Esta limitación ha sido eliminada, toda vez que la herramienta Genexus que apoya a la Metodología propuesta, permite cambiar fácilmente la base de datos, así como generar automáticamente programas que permiten migrar los datos a la nueva estructura y los programas que deben modificarse a raíz de estos cambios en la estructura de la base de datos.(sección 5.1.1.5.)

k. Problemas derivados del fracaso de los proyectos ágiles. Si un proyecto ágil fracasa no hay documentación o hay muy poca; lo mismo ocurre con el diseño. La comprensión del sistema se queda en las mentes de los desarrolladores.. Esta limitación se ha mitigado, por las razones expuestas en el punto d).

2. **Establecer las principales características que debe reunir una Metodología para desarrollar el Software de una PYME de Microfinanzas localizada en la ciudad de Trujillo -Perú, asegurando la Calidad del Software.**

   Se establecieron las principales características que debería reunir la Metodología propuesta para poder satisfacer el requerimiento, llamémosle común a la mayoría de la empresas de microfinanzas de la localidad de Trujillo, incluyendo la Empresa MICREDITPERU, que es la de contar con un software que integre las aplicaciones de Créditos, Contabilidad y Comisiones valiéndonos de las ventajas que proporciona la Internet (sección 5.2.1.).

3. **Determinar las limitaciones de las metodologías ágiles de desarrollo de software actuales.**

   Se estudio las Metodologías de desarrollo de software ágiles identificándose sus desventajas (sección 2.1.2.8.), así también, que es la Metodología Formal (sección 2.1.2.9.) la que mejor elimina los problemas de ambigüedades en la determinación de los requerimientos, toda vez que está basada en la aplicación de la matemática, así también, es una limitación común a todas la metodologías la oportunidad de lograr cumplir con los tiempos comprometidos para la elaboración de las aplicaciones y con el mantenimiento de las mismas al cambiar los requerimientos, ya sea en las funcionalidades o en el diseño de la Base de Datos.

4. **Determinar las mejores prácticas usadas en las principales Metodologías de Desarrollo de** Software **existentes, tanto de las Tradicionales como de las Ágiles.**

   Se estudiaron las principales metodologías de desarrollo de software tanto tradicionales como ágiles, (secciones 2.1.1. y 2.1.2.) identificándose las

mejores prácticas, muchas de ellas se incorporaron en la Metodología propuesta MDDS-GX.

5. **Determinar las principales características y funcionalidades de la Herramienta CASE Genexus.**

   Se estudió, analizó y evaluó las características y funcionalidades de la herramienta CASE Genexus (sección 4.1.2.1.), adquiriéndose expertiz en su manejo (su aplicación en el capítulo V).

6. **Especificar la Metodología a proponer, teniendo en cuenta los objetivos 1., 2., 3. y 4.**

   En el capítulo IV se especificó la metodología MDDS-GX apoyada con la herramienta CASE Genexus, propuesta en este trabajo de Tesis de Maestría.

7. **Validar la Metodología a proponer mediante su aplicación a 2 casos de estudio: Aplicación de Facturación de una Empresa de venta de Equipos de cómputo y de la PYME de microfinanzas MICREDITPERU de la ciudad de Trujillo - Perú.**

   En el capítulo V, se explica la aplicación la metodología MDDS-GX en 2 casos de estudio, en ambos casos en ambiente web, para la plataforma .NET, haciendo uso del administrador de Bases de datos SQL SERVER 2005 Express Edition.

   En el primero de ellos, en el desarrollo de una aplicación de facturación de una empresa dedicada a la venta de equipos de cómputo (sección 5.1.), en donde, a efectos de que la Metodología MDDS-GX no quede a un nivel abstracto, se detalló el uso de las funcionalidades de la herramienta CASE Genexus, la cual genera automáticamente código fuente de programas, tanto para la creación como para el mantenimiento de las Base de Datos y de la aplicación misma.

   En el segundo caso de estudio, se aplico la Metodología a las empresas de microfinanzas (sección 5.2.1.), la cual a efectos de satisfacer los requerimientos de los usuarios, estuvo conformada por 3 aplicaciones, una de Créditos, otra de Contabilidad y una tercera para la administración de las comisiones de los analistas de créditos. Esto dio lugar a la creación de 3 bases de conocimiento determinándose como estructurar estas bases de conocimiento, haciéndose uso de internet y de tecnologías subyacentes a

ella, tales como AJAX y Web services. En este caso se crearon la Base de datos corporativa (sección 5.2.2.), un prototipo (sección 5.2.3.), se creó un procedimiento para crear un Web service y otro para consumirlo (sección 5.2.3.1.) y se aplicó un patrón (sección 5.2.3.2.) a todas los objetos transacciones, haciendo uso de la herramienta CASE Genexus.

6.1.2. PRINCIPALES APORTES

El trabajo realizado ha supuesto un conjunto de aportaciones relativas al tema de investigación abordado en esta Tesis de Maestría, la definición de una Metodología Agil para Desarrollo de Software Apoyada en la Herramienta CASE Genexus, que se exponen a continuación:

Una nueva Metodología para el Desarrollo de Software: Como aportación principal de esta Tesis de Maestría se ha definido una nueva Metodología Agil para el Desarrollo de Software Apoyada en la herramienta CASE Genexus, que este caso concreto se ha aplicado al desarrollo de software para las empresas de microfinanzas de la localidad de Trujillo. En concreto se ha propuesto una guía para la construcción de aplicaciones corporativas, tanto Windows como Web. Este nuevo enfoque metodológico permite a los desarrolladores de software aprovechar al máximo los beneficios del paradigma de computación que proporciona las herramientas CASE, en este caso la herramienta CASE Genexus, la cual está basada en la Administración del conocimiento, hecho que la hace independiente de la plataforma en que se implemente, en la generación automática de programas, cubriendo todo el ciclo de desarrollo de software, teniendo sus principales bondades, la creación y mantenimiento de la Base de Datos normalizada automáticamente, hasta la 3ra forma normal, el de lograr el desarrollo y mantenimiento automático de las aplicaciones en tiempos muy por debajo del empleado haciendo uso de la programación manual, en la obtención de software de calidad.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

En este capítulo se presentan las conclusiones de este trabajo de Tesis de Maestría.

1. Es factible especificar una Metodología Ágil para Desarrollo de Software, apoyada en la Herramienta CASE GENEXUS, que cumpla con los principales requisitos que debe tener una Metodología de Desarrollo de Software, que permita superar o mitigar las limitaciones de las metodologías de desarrollo de software actuales, desarrollar software que satisfaga los requerimientos funcionales y no funcionales de la empresa PYME de microfinanzas MICREDITPERU, así como, los requerimientos generalmente aceptados de las PYMES de microfinanzas de la localidad de Trujillo – Perú.
2. La Metodología MDDS-GX propuesta, es factible de aplicarla en otros tipos de aplicaciones, como es el caso de estudio visto en el capítulo V.
3. El apoyo de la herramienta CASE Genexus con que cuenta la Metodología propuesta, agiliza las etapas de Desarrollo y Mantenimiento de software, toda vez que permite la generación automática de código de los programas fuentes de las aplicaciones y de la creación y mantenimiento de la Base de Datos.

RECOMENDACIONES

Como continuación de este trabajo de Tesis de Maestría, existen diversas líneas de investigación que quedan abiertas y en las que es posible continuar trabajando. Algunas de ellas, están más directamente relacionadas con este trabajo de Tesis de Maestría y son el resultado de cuestiones que han ido surgiendo durante la realización del mismo.

1. En primer lugar, una de las principales líneas de investigación abiertas está relacionada con el desarrollo de software, que permita obtener métricas de la eficiencia de las diferentes fases de la metodología.
2. Aplicar la Metodología para los casos de estudio tratados en esta Tesis, pero implementarlos en otras plataformas, usando el generador Java.
3. Aplicar la Metodología propuesta en esta tesis de maestría en nuevos casos de estudio, para detectar así futuras mejoras o extensiones a la metodología.

GLOSARIO DE TERMINOS

AI Se denomina Inteligencia Artificial (Artificial Intelligence) a la rama de la ciencia informática dedicada al desarrollo de agentes racionales no vivos; entiéndase a un agente como cualquier artefacto capaz de percibir su entorno, es decir tener la capacidad de procesar tales percepciones y actuar en consecuencia.

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

ASP, Active Server Pages , también conocido como ASP clásico, es una tecnología de Microsoft del tipo "lado del servidor" para páginas web generadas dinámicamente, que ha sido comercializada como un anexo a Internet Information Services (IIS).

La tecnología ASP está estrechamente relacionada con el modelo tecnológico y de negocio de su fabricante. Intenta ser solución para un modelo de programación rápida ya que "programar en ASP es como programar en Visual Basic y C#", por supuesto con muchas limitaciones y algunas ventajas específicas en entornos web.

Lo interesante de este modelo tecnológico es poder utilizar diversos componentes ya desarrollados como algunos controles ActiveX así como componentes del lado del servidor, tales como CDONTS, por ejemplo, que permite la interacción de los scripts con el servidor SMTP que integra IIS.

Se facilita la programación de sitios web mediante varios objetos integrados, como por ejemplo un objeto de sesión basada en cookies, que mantiene las variables mientras se pasa de página a página.

Es limitado a solo funcionar con IIS, por lo que su uso es cuestionado por la mayoría de los programadores web quienes prefieren otros lenguajes de programación del lado del servidor como por ejemplo PHP, Perl, Java Etc.

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

BI Se denomina Inteligencia Empresarial o Inteligencia de Negocios (del inglés Business Intelligence) al conjunto de estrategias y herramientas enfocadas a la administración y creación de conocimiento mediante el análisis de datos existentes en una organización o empresa.

BPMS Acrónimo de Business Process Management System; sistema de administración de procesos de negocio.

Business Component Business Component6 es una propiedad de los objetos Transacción GeneXus que permite que dichos objetos se ejecuten sin interfaz de usuario.

Conocimiento Explícito El conocimiento explícito (*Explicit Knowledge*) generalmente se evidencia en cualquier tipo de documentación, libros, fórmulas, contratos, diagramas de proceso, casos de estudio, manuales, etc. Es tangible, visible y puede ser accedido por cualquier tercero.

C# o C# (pronunciado si sharp en inglés) es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma.NET, similar al de Java aunque incluye mejoras derivadas de otros lenguajes (entre ellos Delphi).

La creación del nombre del lenguaje, C#, proviene de dibujar dos signos positivos encima de los dos signos positivos de "C++", queriendo dar una imagen de salto evolutivo, del mismo modo que ocurrió con el paso de C a C++.

Diagrama de Bachman: creados por Charles William Bachman, prominente informático estadounidense, especialmente en el área de las bases de datos.

Recibió el Premio Turing en 1973 por "sus sobresalientes contribuciones a la tecnología de las bases de datos". Fue elegido como miembro distinguido de la British Computer Society en 1977 por su trabajo pionero en sistemas de bases de datos.

Describe gráficamente el esquema lógico de una base de datos jerárquica.

- Los rectángulos representan los tipos de registro.
- Los arcos representan la relación existente entre los tipos de registro.
- Los arcos se etiquetan porque puede existir más de una relación entre dos tipos de registro (no en el modelo jerárquico).
- La flecha simple-doble indica que existe un único padre para cada hijo, pero pueden existir cero o más hijos de un padre.

DLL Acrónimo de Dynamic Linking Library (Bibliotecas de Enlace Dinámico);

término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda del programa por parte del sistema operativo.

Entidad Colección de atributos que describen un objeto (persona, lugar, cosa) o un evento de interés para el Modelo de Datos.

Extensión Es uno o varios módulos que brindan una nueva funcionalidad a la plataforma GeneXus. Estos módulos son archivos de tipo DLL que se instalan en un directorio especifico de la instalación de GeneXus.

Framework Un framework, en el desarrollo de software es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado.

Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros componentes para ayudar a desarrollar y unificar los diferentes componentes de un proyecto.

Generación de código, [Herrington, Jack, 2008] La generación de código es una herramienta muy valiosa que puede tener un impacto impresionante en productividad y la calidad en proyectos de ingeniería de software.

GeneXus GeneXus [Márquez Lisboa Daniel, 2006] es una herramienta inteligente, basada en conocimiento puro, que permite a las empresas estar a la vanguardia en las últimas tecnologías sin tener que aprender a usarlas; y les permite aumentar su productividad y concentrarse en el negocio al automatizar el diseño, desarrollo y mantenimiento de aplicaciones de negocio multiplataforma, que fácil y rápidamente pueden transformar los cambios de la realidad en oportunidades que permitan expandir el negocio y hacerlo más rentable.

Producida en Uruguay por la empresa Artech, su primera versión fue liberada al mercado para su comercialización en 1989. Sus creadores, Breogán Gonda y Nicolás Jodal, han sido galardonados con el Premio Nacional de Ingeniería en 1995 otorgado por el "Proyecto GeneXus".

Gestión del conocimiento (del inglés Knowledge Management) es un concepto aplicado en las organizaciones, que busca transferir el conocimiento y la experiencia existente entre sus miembros, de modo que pueda ser utilizado como un recurso disponible para otros en la organización.

Usualmente el proceso implica técnicas para capturar, organizar, almacenar el conocimiento de los trabajadores, para transformarlo en un activo intelectual que preste beneficios y se pueda compartir.

En la actualidad, las tecnologías de información permiten contar con herramientas que apoyan la gestión del conocimiento en las empresas, apoyando en la recolección, la transferencia, la seguridad y la administración sistemática de la información, junto con los sistemas diseñados para ayudar a hacer el mejor uso de ese conocimiento.

En detalle, se refiere a las herramientas y a las técnicas diseñadas para preservar la disponibilidad de la información llevada a cabo por los individuos dominantes y facilitar la toma de decisiones, así como reducir el riesgo. Es un mercado del software y un área en la práctica de la consultoría, relacionada a disciplinas tales como inteligencia competitiva.

Un tema particular de la administración del conocimiento es que el conocimiento no se puede codificar fácilmente en forma digital, tal como la intuición de los individuos dominantes que viene con años de

experiencia y de poder reconocer los diversos patrones del comportamiento que alguien con menos experiencia no puede reconocer.

El proceso de la Administración del Conocimiento, también conocido en sus fases de desarrollo como "aprendizaje corporativo" o "aprendizaje organizacional", tiene principalmente los siguientes objetivos:

- Identificar, recoger y organizar el conocimiento existente.
- Facilitar la creación de nuevo conocimiento.
- Apuntalar la innovación a través de la reutilización y apoyo de la habilidad de la gente a través de organizaciones para lograr un mejor desempeño en la empresa.

HTML, siglas de *HyperText Markup Language* (Lenguaje de Marcado de Hipertexto), es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. HTML se escribe en forma de "etiquetas", rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir un script (por ejemplo Javascript), el cual puede afectar el comportamiento de navegadores web y otros procesadores de HTML.

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

Inferencia Acto de procesar o derivar una conclusión basado solamente en información que ya se conoce.

IT Acrónimo de Information Technology; Tecnologías de la Información.

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

JavaBeans Modelo de componentes creado por Sun Microsystems para la

construcción de aplicaciones en Java. Se encapsulan varios objetos en un único objeto, para hacer uso de un sólo objeto en lugar de varios más simples.

Debe satisfacer las siguientes convenciones:

Definir un constructor sin argumentos.

Sus propiedades deben ser accesibles mediante métodos *get* y *set* que siguen una convención de nomenclatura estándar.

Lenguaje Se considera como un conjunto de oraciones, que usualmente es infinito y se forma con combinaciones de palabras del diccionario. Es necesario que esas combinaciones sean correctas (con respecto a la sintaxis) y tengan sentido (con respecto a la semántica).

Normalización de bases de datos. El proceso de normalización de bases de datos consiste en aplicar una serie de reglas a las relaciones obtenidas tras el paso del modelo entidad-relación al modelo relacional.

Las bases de datos relacionales se normalizan para:

- Evitar la redundancia de los datos.
- Evitar problemas de actualización de los datos en las tablas.
- Proteger la integridad de los datos.

En el modelo relacional es frecuente llamar tabla a una relación, aunque para que una tabla sea considerada como una relación tiene que cumplir con algunas restricciones:

- Cada tabla debe tener su nombre único.
- No puede haber dos filas iguales. No se permiten los duplicados.
- Todos los datos en una columna deben ser del mismo tipo.

Formas Normales:

Las formas normales son aplicadas a las tablas de una base de datos. Decir que una base de datos está en la forma normal N es decir que todas sus tablas están en la forma normal N.

En general, las primeras tres formas normales son suficientes para cubrir las necesidades de la mayoría de las bases de datos. El creador de estas 3 primeras formas normales (o reglas) fue Edgar F. Codd.

Primera Forma Normal (1FN)*: Una tabla está en Primera Forma Normal si:*

- Todos los atributos son atómicos. Un atributo es atómico si los elementos del dominio son indivisibles, mínimos.
- La tabla contiene una clave primaria.
- La clave primaria no contiene atributos nulos.
- No debe de existir variación en el número de columnas.

Una columna no puede tener múltiples valores. Los datos son atómicos. (Si a cada valor de X le pertenece un valor de Y y viceversa)

Esta forma normal elimina los valores repetidos dentro de una BD

Segunda Forma Normal (2FN)*:*

Dependencia Funcional. Una relación está en 2FN si está en 1FN y si los atributos que no forman parte de ninguna clave dependen de forma completa de la clave principal. Es decir que no existen dependencias parciales. (Todos los atributos que no son clave principal deben depender únicamente de la clave principal).

Tercera Forma Normal (3FN)*: La tabla se encuentra en 3FN si es 2FN y si no existe ninguna dependencia funcional transitiva entre los atributos que no son clave.*

Dependencia funcional transitiva

Sean X, Y, Z tres atributos (o grupos de atributos) de la misma entidad. Si Y depende funcionalmente de X y Z de Y, pero X no depende funcionalmente de Y, se dice entonces que Z depende transitivamente de X. Simbólicamente sería:

X ⟶Y ⟶Z entonces X ⟶Z

FechaDeNacimiento ⟶Edad

Edad ⟶Conducir

FechaDeNacimiento ⟶Edad ⟶Conducir

Entonces tenemos que FechaDeNacimiento determina a Edad y la Edad determina a Conducir, indirectamente podemos saber a través de FechaDeNacimiento a Conducir (En muchos países, una persona necesita ser mayor de cierta edad para poder conducir un automóvil, por eso se mencionamos este ejemplo).

Plataforma .Net Plataforma de desarrollo de software respaldada y desarrollada por Microsoft.

Paradigma de Programación Un paradigma de programación representa un enfoque particular o filosofía para la construcción de software. Tipos de paradigma:

Imperativo o por procedimientos: considerado el más común y está representado por lenguajes como C o BASIC.

Declarativo, por ejemplo SQL.

Funcional: representado por la familia de lenguajes LISP, ML o Haskell.

Lógico: representado por el lenguaje Prolog.

Orientado a objetos: representado por lenguajes como Smalltalk, C#, Java.

Política Cada organización es responsable de definir las políticas que gobiernan su negocio; en este contexto una política es una guía que gobierna las decisiones de negocio.

Una política puede requerir cientos incluso miles de reglas, que cambian a lo largo del tiempo. El proceso por el cual una empresa administra los cambios en las políticas se le denomina ciclo de vida de las reglas de negocio.

Programación Declarativa Un programa es declarativo, si describe las características de "algo" en vez de definir como crearlo. Las sentencias se declaran sin tener en cuenta ningún flujo, simplemente se define que hacer sin definir cómo hacerlo.

Programación Funcional La programación funcional17 es un paradigma de programación declarativa basado en la utilización de funciones matemáticas.

Programación Imperativa En la programación imperativa también denominada procedural; se debe especificar qué se desea obtener (utilizando sentencias de código) y cómo se desea realizarlo (utilizando estructuras de control).

Los programas imperativos son un conjunto de instrucciones que le indican al computador cómo realizar una tarea.

Programación Lógica En los lenguajes de programación lógica, los programas consisten de un conjunto de sentencias que especifican que un objetivo deseado es verdadero. Esto difiere con la programación imperativa en donde los programas especifican un conjunto de sentencias que se deben de ejecutar para obtener el objetivo deseado.

No existen sentencias de control como por ejemplo *If-then-else* en este tipo de lenguajes lo que lo hace un lenguaje sumamente potente y difícil a la vez. Está representado por ejemplo por el lenguaje Prolog; a su vez surgen modelos más simples que soportan un subconjunto de elementos del paradigma como por ejemplo Lógica de Primer Orden.

Programación orientada a Objetos La Programación orientada a objetos es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas.

Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo, y encapsulamiento.

Su uso se popularizó a principios de la década de 1990; algunos ejemplos de lenguajes de programación que soportan la orientación a objetos son: Java, C#, Smalltalk.

Prolog Prolog es un lenguaje de programación lógico declarativo. La lógica del programa se expresa en términos de relaciones, y la ejecución es gatillada por el disparo de consultas sobre dichas relaciones.

Se basa en una implementación procedural de las cláusulas de Horn.

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos, creado por el programador japonés Yukihiro "Matz" Matsumoto, quien comenzó a trabajar en Ruby en 1993, y lo presentó públicamente en 1995. Combina una sintaxis inspirada en Python y Perl con características de programación orientada a objetos similares a Smalltalk. Comparte también funcionalidad con otros lenguajes de programación como Lisp, Lua, Dylan y CLU. Ruby es un lenguaje de

programación interpretado en una sola pasada y su implementación oficial es distribuida bajo una licencia de software libre.

RUP Acrónimo de Rational Unified Process25; framework de desarrollo de software iterativo creado por Rational Software Corporation, una división de IBM desde 2003. Es un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

SDT Structured Data Type u Objeto Estructurado26, es la representación GeneXus de cualquier estructura compleja.

SOA Acrónimo de Services Oriented Architecture (Arquitectura orientada a servicios); estilo arquitectónico cuyo objetivo es asegurar bajo acoplamiento entre agentes de software que interactúan.

SOAP (siglas de Simple Object Access Protocol) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Este protocolo deriva de un protocolo creado por David Winer en 1998, llamado XML-RPC. SOAP fue creado por Microsoft, IBM y otros y está actualmente bajo el auspicio de la W3C. Es uno de los protocolos utilizados en los servicios Web.

SQL Acrónimo de *Structures Query Language*; lenguaje de consulta estructurado para acceder a bases de datos relacionales.

SQL Server 2005 Express Edition, SQL Server Express es una versión de SQL Server 2005 diseñada para ayudar a los desarrolladores a construir aplicaciones robustas y fiables ofreciendo una sencilla pero potente base de datos que es además gratuita. SQL Server Express ya está incluida como opción al instalar cualquier herramienta Visual Studio Express , por lo que no necesitas descargarla aparte si ya estás instalando cualquiera de ellas.

Stakeholder Interesado de una manera u otra en un proyecto de software.

Test Unitario Método para asegurarse que unidades individuales de código funcionan correctamente; conforma la parte más pequeña de una aplicación que se puede testear.

Transacción (GeneXus) Objeto GeneXus que brinda información de los datos de la aplicación y cómo el usuario va a acceder a la aplicación para

realizar el CRUD (insertar, leer, modificar o borrar datos). GeneXus a partir de las transacciones construye la Base de Datos, normalizada a tercera forma normal, y también genera programas para hacer las operaciones mencionadas.

UML Acrónimo de Unified Modeling Language, lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por la OMG. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

Visual Studio IDE desarrollado por Microsoft para trabajar con la plataforma .Net.

Vocabulario Verbalización formal muy cercano al lenguaje natural de los conceptos y elementos del modelo de negocio.

W3C El consorcio World Wide Web es una comunidad internacional para el desarrollo de estándares Web.

WCF Acrónimo de Windows Communication Foundation; plataforma de mensajería que forma parte de la API de la Plataforma .NET 3.0. Fue creado con el fin de permitir una programación rápida de sistemas distribuidos y el desarrollo de aplicaciones basadas en arquitecturas orientadas a servicios.

WebPanel (GeneXus) Objeto GeneXus

Web Service Un servicio web es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes como por ejemplo Internet.

Workflow Define la secuencia (flujo) en la cual se debe de realizar el trabajo para completar un proceso.

WSDL son las siglas de Web Services Description Language, un formato XML que se utiliza para describir servicios Web (algunas personas lo leen como wisdel). La versión 1.0 fue la primera recomendación por parte del W3C y

la versión 1.1 no alcanzó nunca tal estatus. La versión 2.0 se convirtió en la recomendación actual por parte de dicha entidad.

WSDL describe la interfaz pública a los servicios Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.

Así, WSDL se usa a menudo en combinación con SOAP y XML Schema. Un programa cliente que se conecta a un servicio web puede leer el WSDL para determinar qué funciones están disponibles en el servidor. Los tipos de datos especiales se incluyen en el archivo WSDL en forma de XML Schema. El cliente puede usar SOAP para hacer la llamada a una de las funciones listadas en el WSDL.

El WDSL nos permite tener una descripción de un servicio web. Especifica la interfaz abstracta a través de la cual un cliente puede acceder al servicio y los detalles de cómo se debe utilizar.

XMI Acrónimo de XML Metadata Interchange; especificación para el Intercambio de metadata vía XML.

Puede ser utilizada para intercambiar cualquier metamodelo cuya metadata pueda ser expresada utilizando el estándar MOF; el uso más común es como formato de intercambio entre herramientas de modelado utilizando modelos UML.

XML Acrónimo de eXtensible Markup Language; estándar creado por la W3C partiendo de las especificaciones de SGML32.

XML Schema XML Schema es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML. Se consigue así, una percepción del tipo de documento con un nivel de abstracción alto. Fue desarrollado por la W3C y alcanzó el nivel de recomendación en mayo de 2001.

BIBLIOGRAFIA

Alistar Cockburn and Jim Highsmith, Crystal Clear A Human – Powereel Metodology for Small Teams, (Agile Development Series), (Paper back, Oct, 29, 2004), Series Editors.

Alvarez, Borriero, Valenzani, and Oliveri. Entorno de desarrollo Web para Genexus. Tech. rep., Facultad de Ingeniería - Universidad ORT, 2009.

Arias, F. J., Moreno, J., and Ovalle, D. A. Integración de Ontologías y Capacidades de Razonamiento en Agentes de Software Inteligentes para la Simulación del Proceso de Negociación de Contratos de Energía Eléctrica. *Avances en Sistemas e Informática* (2007).

Artech Consultores S.R.L., Visión General de Genexus, Ultima actualización Octubre del 2008.

Artech Consultores S.R.L.. Curso GeneXus 9.0, Diciembre del 2006.

Artech Consultores S.R.L. Primeros Pasos con GeneXus 9.0, Ultima actualización Abril del 2006

Artech Consultores S.R.L. GeneXus – Grow thru knowledge – Curso Desarrollo de aplicaciones para Internet con GeneXus versión 9.0 – Marzo del 2007.

Avison et, al. Action Research, Avison. D., Lan, F., Myers, M., y Nielsen, A. Communications of the ACM, 1999.

Beck, K. "Extreme programming Explained Embrace Change", Pearson Education, 1999, Addison-Wesley, 2000.

BOOCH Grady et al. El lenguaje Unificado de Modelado, Primera Edición, Editorial Addison Wesley, 1999.

Bunge, M. La Investigación Científica. Ed. Ariel, S.A. Barcelona, 1976

Callaos, Nagib. METODOLOGIAS ABIERTAS Y CERRADAS. Callaos y Asociados Ingenieros Consultores C.A, abril 1991.

Carranza, Zalatiel Análisis de Sistemas de Software, primera reimpresión 2008, Fondo Editorial Universidad de Lima, ISBN # 9972-45-190-9.

Chen, M.; Norman, R.: A Framework for Integrated CASE. IEEE Software, march, p. 18-22.,1992.

Codd, E. F. A Relational Model of Data for Large Shared data Banks. Commun. ACM 13, 6 (1970), 377–387.

Chikofsky Elliot J., Rubenstein Burt L. (1998), CASE: Reliability Engineering for Information Systems, IEEE computer society Press technology series, Software development, Computer-Aided Software Engineering (CASE), IEEE March, p.11-16.

Fowler Martin, Beck Kent, Brant John, Opdyke William y Roberts Don. Refactoring: Improving the desingn of existing code. Addison Wesley, 1999.

Herrington. Jack, Code Generation in Action, Ed. Manning, 2008

Highsmith, Jim. "Extreme Programing". EBusiness Applicatión Development, Cutter Consortium, Febrero del 2000.

Highsmith Jr., Orr K. "Adaptative Software Development: A Collaborative Approach to Managin Complex Systems", Dorset House, 2000.

Kent Beck, "RE: (OTUG) XP and Documentation". Rational's Object Technology User Group Mailing List, 23 de Marzo de 1999.

Kendall, Kenneth y Kendall, Julie E. - ANALISIS Y DISEÑO DE SISTEMAS - Editorial Prentice Hall – 3ª Edición - 1999.

Kerr, E. : CASE the Potential and the Pit Falis: QED Information Sciencies, The Weliesley, Massachusetts, 52,56,110, 112 p., 1989.

Ken Schwaber, Agile Project Management with Scrum, Microsoft Press, January 2004, 163pp, ISBN 0-7356-1993-X24

Keremer, C. : Learning Curve Models For Integrated CASE tool Management. Mit Center for Information System Research, IEEE Software; 23-28., 1992.

Laudon, Kenneth C. y Laudon, Jane P. - Sistemas de Información Gerencial – Editorial Prentice Hall, sexta edición – 2002.

Liskov, B.H. y V. Berzins.- "An appraisal of program especifications", en software specificatrions techniques, N. Gehani y A.T. Mackkittrik (eds). Addison – Wesley, 1986, p3.

Maeda, John. "The Subtlety of Simplicity." Laws of Simplicity. Cambridge, 2005.

Maier, Mark W., and Eberhardt Rectin. The Art of Systems Architecting. Boca Ratón, FL: CRC Press, 2002.

McLeod Jr., Raymond. - Sistemas de Información Gerencial - Editorial Prentice Hall Hispanoamericana, S.A. México – 7ª Edición - 2000.

Marcos, Esperanza – Software engineering research versus software development – SIGSOFT Software Engineering Notes, vol 30 Issue 4, Publisher ACM, July 2005.

Marcos, E. y Marcos, A. – An Aristotelian Approach to the Methodological Research: a Method for Data Models Construction. En: Information Systems – The Next Generation. Ed. L. Brooks and C. Kimble. Mc Graw-Hill, pp. 532-543, 1998

Márquez Lisboa Daniel – GENEXUS, Desarrollo basado en el conocimiento – Editorial Grupo Magro, primera edición - 2006

Nor04 - Darrell Norton. "Lean Software Development Overview", http://dotnetjunkies.com/WebLog/darrell.norton/articles/4306.aspx, 2004.

Opdyke, William F. Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks. Tesis de Doctorado, University of Illinois at Urbana-Champaing, 1992.

Pekka Abrahamsson. "Agile Software development methods: A minitutorial". VTT Technical Research Centre. http://www.vtt.fi/virtual/agile/seminar2002/Abrahamsson_agile_methods_minitutorial.pdf, 2002. Pérez, M. Arquitectura para Ambientes CASE Integrados. Tesis Doctoral. UCV. 1999.

Piattini, M. y Daryanani, S., Elementos y herramientas en el desarrollo de sistemas de información. Una visión actual de la tecnología CASE, Ra-Ma, Madrid., 1995.

Poppendieck M., Poppendieck T. "Lean Software Development: An Agile Toolkit for software Development managers", Addison Wesley, 2003.

Pressman, Roger S. – INGENIERÍA DEL SOFTWARE: UN ENFOQUE PRÁCTICO, 6ta edición – Editorial Mc Graw Hill – 2005.

Ramón Flores: Instituto del Perú – El Fin Social como parte inherente de las Microfinanzas – 2009

http://www.institutodelperu.org.pe

Ś Finanzgruppe, Sparkassenstiftung fur internacionales Kooperation microfinanzas: 10 Tésis, 2003.

Sánchez, Cesar: GESTIOPOLI.com S - TLC y Microfinanzas en el Perú.

http://www.gestiopolis.com/economia/tlc-y-microfinanzas-en-el-peru.htm

Smith, D. (1989): Evaluating and Selecting CASE Tools. Systems Development Management, Auerbach.

Sommerville, I., Ingeniería del Software (7th Edition), Addison Wesley, 2005

Stapleton J. "Dsdm Dynamic Systems Development Method: The Method in Practice". Addison-Wesley. 1997.

Stepen Palmer, Practical Guide to Feature Driven Development, Editorial Prentice Hall Iberia ISBN 01-13-067615-2., 2002.

Susman y Evered, An Assesment of the Scientific Merit of Action Research. Susman, G. Evered, R. Administrative Science Qarterly, 1978.

Tom Mens y Arie Van Deursen. "Refactopring: Emerging trends and open problems". http://homepages.cwi.nl/~arie/papers/refactoring/reface03.pdf, Octubre del 2003.

Wadsworth, What is Participatory Action Research?. Wadsworth, Y. Action Research International, 1998. Disponible en:

http://scu.edu.au/schools/gcm/ar/aripywadsworth98.html.

Waterfield c., Ramsing N. – CGAP/World Bank – Grupo Consultivo de Ayuda a la Población más Pobre – Sistema de información gerencial para instituciones de microfinanzas – Serie de instrumentos técnicos Nº 1., 1998.

http://www.redcamif.org/uploads/tx_rtgfiles/TechnicalTool_01_Spanish.pdf

Whitten*, J. L., Bentley*, L. D., and Dittman K. C., (1998) Systems analysis and design methods (4th ed.) Richard D. Irwin Publishing. (ISBN # 0-256-19906-X (724 pages)).