

DESARROLLO BASADO EN CONOCIMIENTO

FILOSOFÍA Y FUNDAMENTOS TEÓRICOS DE GENEXUS

por Breogán Gonda y Nicolás Jodal

Copyright © Artech 1988 - 2007, todos los derechos reservados

Mayo de 2007

RESUMEN: SE TRATA DE MOSTRAR, DE UNA MANERA SIMPLE Y SISTEMÁTICA, LA FILOSOFÍA Y LOS FUNDAMENTOS TEÓRICOS DE GENEXUS. BUENA PARTE DE ESTOS ELEMENTOS SON CONOCIDOS POR LA COMUNIDAD GENEXUS, PERO ESTÁN RECOGIDOS DE MANERA SÓLO PARCIAL EN UNA SERIE DE DOCUMENTOS.

INTRODUCCIÓN

El propósito de este documento es ayudar a entender las ideas básicas y los conceptos teóricos que hicieron posible GeneXus, para que pueda comprenderse mejor su esencia y los nuevos desarrollos que se irán produciendo en el futuro.

El primer propósito de GeneXus era establecer un robusto modelo de datos. Una vez construido con todo rigor dicho modelo, se pudo pensar en una ampliación del mismo incorporándole elementos declarativos como reglas, fórmulas, pantallas, etc., tratando de describir las visiones de datos de los usuarios y, de esta manera, abarcar todo el conocimiento de los sistemas de negocios.

¿Por qué se trató de abarcar todo el conocimiento de los sistemas de negocios? Porque es una condición necesaria para poder proyectar, generar y mantener en forma 100% automática los sistemas computacionales de una empresa cualquiera. Parecía claro que la comunidad informática, en algún momento, optaría por este camino y era bueno anticiparse en recorrerlo tanto como fuera posible.

Pero, ¿cómo abarcar todo el conocimiento de los sistemas de negocios?, ¿cómo abarcar la casuística que tienen los sistemas de negocios?

La primera versión de GeneXus (100% declarativa) pretendía resolver el problema de generación y mantenimiento del 70% de los programas. De ellos se ocupaba el sistema en forma automática. El 30% restante debía ser programado y mantenido manualmente.

Los “prospects” entendieron que resolver automáticamente el 70% de su problema era muy bueno e, incluso, tuvieron dudas de que el mantenimiento automático fuera posible: los primeros que adoptaron GeneXus lo hicieron exclusivamente por el aumento de productividad en el desarrollo. Sin embargo, luego de algunos meses, esos clientes apreciaron mucho el gran aumento de productividad que obtenían en el desarrollo, pero apreciaron mucho más aún el mantenimiento automático.

Al año del lanzamiento, el proyecto GeneXus comenzó a sufrir una fuerte presión de sus clientes (el éxito del mantenimiento automático y las ventajas que ello les proporcionaba los llevó a exigir “mantenimiento automático para todo” lo que, como precondición exigía “generación automática de todo”).

Todo esto no tenía antecedentes en el mercado internacional (realmente, muchos años después, GeneXus sigue siendo el único producto en el mundo que resuelve estos problemas).

Pero, entonces, ¿Cuál es hoy el objetivo de GeneXus? **El objetivo de GeneXus es conseguir un muy buen tratamiento automático del conocimiento de los sistemas de negocios.**

Ésta es la esencia. Cumplido ese objetivo existe un gran conjunto de subproductos como, por ejemplo:

- Proyecto, generación y mantenimiento automáticos de la base de datos y los programas de aplicación necesarios para la empresa.
- Generación, a partir del mismo conocimiento, para múltiples plataformas.
- Integración del conocimiento de diversas fuentes para atender necesidades muy complejas con costos en tiempo y dinero muy inferiores a los habituales.

Generación para las tecnologías futuras, cuando esas tecnologías estén disponibles, a partir del conocimiento que, hoy, atesoran los clientes GeneXus en sus Knowledge Bases: GeneXus trabaja con el conocimiento puro, cuya validez es totalmente independiente de las tecnologías de moda.



Todo el desarrollo ha seguido ciertas líneas maestras. Este documento tiene por propósito explicitar estas líneas maestras, de una manera simple y conceptual.

NUEVO PARADIGMA: DESCRIBIR EN VEZ DE PROGRAMAR

¿De qué manera pretendemos describir la realidad?

Se pretende "describir" en vez de "programar". Se pretende maximizar las descripciones declarativas y minimizar las especificaciones procedurales.

¿Por qué “describir” en vez de “programar”?

Esta pretensión constituye un cambio esencial de paradigma e implica un choque cultural. Los cambios de paradigma tienen el problema de su lenta adopción pero luego, si son correctos (¡y éste sin duda lo es!), en algún momento se producen hechos que determinan su rápida adopción general.

¿Cuáles son los hechos a que nos referimos?: hoy la enorme mayoría de los sistemas se desarrollan y mantienen con programación manual pero, ¿cuál ha sido la evolución del mercado en los últimos 40 años (datos al año 2006)?

La complejidad de los sistemas ha aumentado en un 2000%

La productividad de los lenguajes de programación ha aumentado en un 150%

Esta situación es inmanejable para los negocios. Los costos crecen de una manera desmesurada.

Podemos separar esos costos en dos dimensiones: **Dinero y Tiempo**.

Las empresas han descubierto, gracias a la disponibilidad de comunicaciones rápidas y baratas, que puede actuar sobre la dimensión **Dinero**, contratando el desarrollo y mantenimiento de sus sistemas en países de bajos salarios y lo ha hecho intensamente en los últimos años.

En realidad no se ha resuelto el problema del costo, sino que se cambió la unidad de medida: son básicamente las mismas horas de trabajo (es sustancialmente el mismo trabajo), pero el precio de la hora-hombre es mucho menor. Si medimos el costo en dinero, se ha producido un fuerte abaratamiento.

¿Qué ocurre con la dimensión **Tiempo**? Lo anterior no se aplica a los tiempos.

Los sistemas de negocios necesitan responder a nuevas necesidades: nuevos dispositivos, nuevos usuarios, nuevas modalidades de uso, nuevas posibilidades de integración y, por ello, se hacen cada día más y más complejos y los negocios están sujetos a un “time to market” cada vez más crítico y que no pueden cambiar. Como consecuencia, cada día más negocios se están perjudicando por los tiempos inadecuados de desarrollo de las nuevas soluciones y de mantenimiento de las actuales. Y hoy (pero sobre todo en un futuro próximo) nadie puede hacer buenos negocios sin los sistemas adecuados.

Esta situación no puede seguir así durante mucho tiempo, y primero lentamente, hoy más rápidamente, van apareciendo ejemplos de ello como:

Muchas empresas en casi todos los países más desarrollados están recurriendo más y más al “outsourcing”, generalmente acompañado por “offshoring” hacia países de bajos salarios, del desarrollo de sus sistemas, pero simultáneamente hay una corriente creciente de empresas que siguen el camino inverso, porque no han obtenido resultados convenientes con dicho “outsourcing”.

Cada vez más existen empresas que experimentan una gran violencia al despedir a sus técnicos y traspasar su trabajo a técnicos desconocidos de países lejanos. Para evitarlo necesitan que sus costos “in house” sean similares a los del “outsourcing”. Sólo la adopción de tecnologías de muy alto nivel lo permitirá.

Es necesario un dramático aumento de la productividad pero la productividad de los lenguajes de programación ha llegado hace ya bastante tiempo a una estabilización.

¿Cómo se resuelve el problema?, ¿con mejores lenguajes de programación?: ¡los hechos muestran que no! El problema no está en los lenguajes de programación, sino en la propia programación.

¿Cómo lograr, entonces, el aumento de productividad que se necesita? Haciendo desarrollo basado en conocimiento y no en programación: ¡la solución es **describir en vez de programar!**

En Artech estamos convencidos de ello y nos preparamos para esa explosión del mercado del desarrollo basado en conocimiento que se producirá en los próximos años.

MODELO DE DATOS

Históricamente la comunidad informática comenzó trabajando sólo con un modelo físico, que tenía muchas rigideces. Luego fue introduciendo otros modelos a los efectos de tener más flexibilidad y obtener cierta permanencia de las descripciones. El trabajo más importante sobre el tema lo ha sido el informe de 1978 del grupo **ANSI SPARC [1]** que introduce tres modelos:

Modelo Externo donde se representan las visiones externas.

Modelo Lógico o Conceptual: Es otro modelo (generalmente un modelo E-R) que se pretendía obtener por abstracción de la realidad.

Modelo Físico: Se refería fundamentalmente al esquema de la base de datos.

La idea era desarrollar paralelamente los tres modelos para que los programas sólo interactuaran con el Modelo Externo y, a partir del Modelo Lógico se hiciera un "mapping" entre esos programas y la base de datos (Modelo Físico) y que ello hiciera independientes a los programas de la base de datos.

El informe fue muy elogiado en su momento y, luego, casi olvidado. Sólo un fabricante a inicios de los 80 trató de implementar este esquema de tres modelos. Lo fue Cincom Systems para su producto **SUPRA** y para diferentes lenguajes accediendo a bases de datos SUPRA.

Más allá de todo esto, el informe ANSI SPARC sigue siendo válido hoy. La mayor diferencia es la tecnología disponible. En los años transcurridos, los cambios tecnológicos han sido muy importantes.

Analizando el asunto a la luz de la tecnología actual, concluimos que pueden existir varios modelos, pero **el modelo realmente importante para los usuarios y los desarrolladores, es el Modelo Externo**: en él recogemos el conocimiento exterior y todo lo demás como, otros modelos auxiliares que pudieran ayudar, debe y puede inferirse automáticamente a partir de dicho Modelo Externo.

En este trabajo ponemos el énfasis en el **Modelo Externo**, donde está el conocimiento genuino y donde reposa la esencia y la singularidad de GeneXus (el “**qué**”). Ante nuevas posibilidades de la tecnología y necesidades de los clientes, lo primero a hacer es ampliar dicho Modelo Externo.

CARACTERÍSTICAS QUE DEBE TENER NUESTRO MODELO

Ciertamente los objetos-tipo de GeneXus pueden evolucionar en el futuro pero obedeciendo a ciertos principios que es bueno recoger aquí.

Pero existe una pregunta previa: ¿quiénes detentan el conocimiento dentro de las organizaciones?

El único conocimiento objetivo y suficientemente detallado es el que poseen los usuarios a todo nivel sobre las visiones de los datos que ellos utilizan, o necesitan, o desean. No existe, en cambio, conocimiento objetivo sobre los propios datos.

Por ello **debemos privilegiar lo concreto sobre lo abstracto**, porque los usuarios, que serán múltiples y tendrán los perfiles y roles más diversos en la empresa, se manejan bien con elementos concretos de su ambiente y mal con conceptos abstractos: la representación del conocimiento debe ser simple, detallada y objetiva.

Parece adecuado basarnos en un “Modelo Externo” que recoja estas visiones. Un “Modelo Externo” no puede contener ningún elemento físico o interno como: archivos, tablas, entidades, relaciones entre entidades, índices o cualquier otro que pueda deducirse automáticamente del mismo.

¿qué elementos son deseables y qué elementos son inaceptables en GeneXus?

Estamos tratando de hacer un modelo que tenga las siguientes características:

Conocimiento adecuado para su tratamiento automático. Modelo con el cual un software de computador pueda trabajar automáticamente. O sea que el conocimiento de nuestro modelo debe ser “entendible” y “operable” automáticamente.

Consistencia. Modelo siempre consistente.

Ortogonalidad. Los objetos que en un determinado momento constituyen el modelo deben ser independientes entre sí. La adición de un nuevo objeto o su modificación o eliminación no implicarán la necesidad de modificar ningún otro objeto.

Proyecto, generación y mantenimiento automáticos de la base de datos y los programas. Entre las cosas que un software de computador puede hacer automáticamente con el modelo, son esenciales el proyecto, generación y mantenimiento automáticos de la base de datos y los programas.

Escalabilidad. Los ítems anteriores expresan las características cualitativas que debe tener nuestro modelo. Pero se pretende resolver problemas grandes. Para ello necesitamos que los mecanismos de almacenamiento, acceso e inferencia del modelo actúen eficientemente con independencia del tamaño del problema.

MARCO DE REFERENCIA

¿Cómo representar el conocimiento de una manera rigurosa? Debemos establecer un conjunto de objetos-tipo predefinidos a través de los cuales nos sea posible representar bien la realidad y que sean automáticamente entendibles y operables.

Un camino que, en general, ha seguido la industria, es partir de un modelo de datos E-R o similar. Pero ¿Cómo obtener el modelo de datos correcto? Un modelo de datos corporativo necesita cientos o aún miles de tablas. Tradicionalmente se diseñaban los modelos de datos mediante un proceso de abstracción (identificar, a través de interlocutores que tuvieran el conocimiento suficiente de la empresa, sus “objetos relevantes” y sus “relaciones” e introducir en el modelo las correspondientes “entidades” y “relaciones”).

Este proceso de abstracción era fuertemente subjetivo: ¿quién en la organización tenía un buen conocimiento de los datos?, ¿quién tenía un buen conocimiento de los “objetos” y “relaciones” “relevantes” que eran el input del modelo de datos? La respuesta es clara y contundente: nadie. Entonces debía recurrirse al conocimiento de múltiples personas y cada una de ellas iba agregando su propia subjetividad.

¿Dónde estaba el conocimiento objetivo y suficientemente detallado necesario para construir nuestro modelo? Ciertamente no existía en la organización ese conocimiento sobre los datos. Sin embargo cada usuario, a cualquier nivel, conocía muy bien las “visiones de datos” que utilizaba, o que necesitaba, o que deseaba: **nadie conoce los “datos” pero todos trabajan permanentemente con visiones de esos datos.**

Se optó por tomar esas “visiones de datos” como input básico del modelo pero, ¿Cómo describirlas objetivamente?

Era necesario establecer un sólido marco de referencia sobre el cual hacer las demás descripciones.

Ese marco de referencia está constituido por los **Atributos**.

Elemento semántico fundamental: Atributo

Existen atributos, cada atributo tiene un nombre, un conjunto de características y un significado.

Para tener un modelo de gran potencia expresiva debemos representar en él, además de los elementos sintácticos que toda la comunidad informática maneja sin dificultades, elementos semánticos. Pero es difícil trabajar automáticamente con la semántica.

Para tratar automáticamente un elemento semántico, es necesario darle una representación sintáctica.

Se escogió un sólo elemento semántico para ser representado en nuestro modelo: el **significado** de cada atributo.

El segundo elemento básico es entonces: “significado”, que en nuestro modelo definimos de la siguiente manera: significado (atr) = nombre (atr)

Los nombres de los atributos pasan a ser esenciales para el modelo → Son necesarias normas claras para asignarlos de manera de asegurar su consistencia a través de todo el modelo.

A ese efecto, adoptamos la **URA (Universal Relational Assumption)** que, sustancialmente, significa que un atributo tendrá el mismo nombre en todos los lugares donde aparezca y que no habrá dos atributos diferentes (con significado diferente) que compartan el mismo nombre. Debe procurarse además que los nombres sean auto-explicativos y muy fáciles de entender por terceros.

Cliente	Nombre de Cliente	Factura	Fecha de Factura	Importe de Factura	Cantidad de Línea de Factura	Precio de Línea de Factura	Producto	Descripción de Producto	Tabla
									Clientes
									Factura
									Línea de Factura
									Producto

El tercer elemento básico es, entonces, la URA (Universal Relational Assumption).

La realidad nos ha mostrado que la URA no es suficiente por lo que establecimos **subtipos de atributos** y, luego, **subtipos de grupos de atributos**.

El cuarto elemento está constituido, entonces, por los “subtipos”.

RESUMIENDO: el **atributo**, tal cual lo hemos caracterizado aquí (con las consideraciones sobre **significado**, **URA** y **subtipos**), es el elemento semántico fundamental de la teoría de GeneXus. Los **atributos** constituyen el marco de referencia sobre el cual edificamos los modelos GeneXus.



DESCRIPCIÓN DE LA REALIDAD

Luego de introducir el marco de referencia, describiremos la realidad a través de un conjunto instancias de “objetos-tipo”, que especificaremos basándonos en sus atributos y mediante los cuales recogeremos las características de la realidad a representar.

TRANSACCIONES

Cada usuario tiene una o múltiples visiones de los datos que utiliza cotidianamente.

Entre estas visiones, podemos pensar en un primer tipo: el que agrupa aquellas que se utilizan para manipular los datos (introducirlos, modificarlos, eliminarlos y visualizarlos en forma limitada), a estas visiones de usuarios les hemos llamado **Transacciones** y constituyen el primer objeto-tipo de GeneXus.

Cada transacción tiene un conjunto de elementos:

Estructura de los datos de la Transacción: los datos se presentan de acuerdo a una estructura y debemos tener una manera de recoger con todo rigor esas estructuras. A continuación veremos la estructura de una transacción que todos conocemos: la de una factura simple.

[FACTURA]

*Código_de_Factura **
Fecha_de_Factura
Nombre_de_Cliente
Dirección_de_Cliente
*(Código_de_Producto**
Descripción_de_Producto
Cantidad_Vendida_Producto_en_la_Factura
Precio_Venta_Producto_en_la_Factura
Importe_Línea_Factura)
Sub_Total_Factura
Descuento_Factura
IVA_Factura
Total_Factura

En esta representación existen tres grupos de atributos: un “prólogo o cabezal”, que ocurre una sola vez y que aparece al principio un “cuerpo” que ocurre un cierto número de veces y un “epílogo” o “pie” que ocurre una sola vez. Dentro de éste prólogo, aparece al lado del atributo **Código_de_Factura** un *, lo que significa que, para cada **Factura** (cada ocurrencia del Prólogo y del Epílogo) existe un único **Código_de_Factura**; luego existe entre paréntesis un conjunto de atributos: ello significa que este conjunto de atributos puede ocurrir múltiples veces dentro de cada **Factura** y solemos llamar cuerpo a este grupo repetitivo: el cuerpo debe tener un identificador, que identifica bien cada una de sus líneas dentro de la **Factura**. En este caso lo es **Código_de_Producto** y esa condición se señala colocándole un * a la derecha. Luego del cierre del grupo repetitivo encontramos un epílogo o pie.

Este Diagrama de Estructura de Datos y el actualmente utilizado por GeneXus que tiene características más gráficas, toman como antecedente los introducidos en los trabajos de Warnier-Orr [2] y Jackson [3].

Reglas: probablemente habrá un conjunto de reglas que afectan a la transacción, como las siguientes.

No existirán dos Facturas con el mismo **Código_de_Factura**.

El **Código_de_Factura** se atribuirá de manera correlativa.

La **Fecha_de_Factura** tomará como opción por defecto la fecha del día de su emisión.

La **Fecha_de_Factura** no podrá ser menor a la fecha de emisión de dicha Factura.

No se admitirá ninguna **Factura** que deje negativo el **Stock_del_Producto** para alguno de sus productos.

No se admitirá ninguna **Factura** que haga que el **Saldo_Deudor_del_Cliente** involucrado supere su límite de crédito.

Cuando la aceptación de una Factura determine que el

Stock_del_Producto < Punto_de_Pedido_Producto, para alguno de sus productos, se deberá activar la rutina de **Aprovisionamiento (Producto)**.

Fórmulas: probablemente habrá un conjunto de fórmulas que afectan a la transacción, como, por ejemplo, las siguientes.

Importe_Línea_Factura = Cantidad_Vendida_Producto_en_la_Factura * Precio_Venta_Producto_en_la_Factura

Sub_Total_Factura = sumatoria dentro de la **Factura (Código_de_Factura)** de **Importe_Línea_Factura**

Descuento_Factura = función de cálculo del descuento (**Código_de_Factura**)

IVA_Factura = función de cálculo del IVA (**Sub_Total_Factura – Descuento_Factura**)

Total_Factura = Sub_Total_Factura – Descuento_Factura + IVA_Factura

Saldo_Deudor_del_Cliente = sumatoria de facturas impagadas (**Código_de_Cliente**)

Elementos de presentación: Es necesario asociar al objeto elementos de presentación como formatos de pantallas para su despliegue en Windows o en un Browser, estilos gráficos y de diálogo, etc.

Las transacciones son declarativas y capturando el conocimiento que existe en ellas y sistematizándolo en una Knowledge Base se obtiene buena parte de la descripción de la realidad que se busca.

CONSIDERACIONES SOBRE LAS TRANSACCIONES Y LA TEORÍA RELACIONAL DE CODD [4].

Una interpretación ligera de este objeto, puede hacer pensar que nuestro modelo entra en abierta contradicción con el Modelo Relacional de Codd: existen nuevos elementos como fórmulas, que no incluye el Modelo Relacional de Codd y grupos repetitivos, explícitamente excluidas en él.

El Modelo Relacional y nuestro modelo tienen objetivos diferentes. Veámoslo más adelante, pero debemos tener claro que en ningún momento Codd afirmó que “la realidad está compuesta por un conjunto de relaciones”, lo que sería un disparate, sino que “para almacenar adecuadamente los datos y operar con ellos, el modelo relacional es muy adecuado”.

El Modelo Relacional está orientado a obtener una buena representación de los datos en una Base de Datos, obedeciendo algunas condicionantes muy deseables: eliminar la redundancia e introducir un pequeño conjunto de reglas, de manera de evitar las mayores fuentes de inconsistencia de los datos e introducir un conjunto de operadores que permitan manipular los datos a buen nivel.

El Modelo Externo será utilizado para obtener y almacenar el conocimiento. Este modelo pretende la representación más directa y objetiva posible de la realidad. Por ello, tomamos las visiones de los diferentes usuarios. Estas visiones son almacenadas en el modelo. Luego, se captura todo el conocimiento contenido en ellas y se lo sistematiza para maximizar las capacidades de inferencia.

Dentro de este contexto, el Modelo Relacional (o más bien un super-conjunto de él) es utilizado para representar y/o manipular los datos: corresponde al llamado “Modelo Interno o Físico” al que se refiere el informe ANSI SPARC referido.

En nuestra teoría se utiliza fuertemente, el Modelo Relacional.



PROCEDIMIENTOS

Las Transacciones, como hemos visto, nos permiten de manera declarativa describir las visiones de datos de los usuarios y, como consecuencia, aconsejar el conocimiento necesario para proyectar, generar y mantener de manera totalmente automática, la Base de Datos y los programas que los usuarios necesitan para manipular sus propias visiones de datos.

Pero esto no es suficiente: existe la necesidad de procesos “batch” o similares. Estos procesos no se pueden inferir a partir del conocimiento aportado por las visiones de datos.

Cuando fue liberada la primera versión de GeneXus, las transacciones resolvían aproximadamente el 70% del problema, generando los programas correspondientes. Faltaba el 30% de los programas.

Para lograr un 100% de cobertura a las necesidades de los clientes, fue necesario recurrir a un objeto que permitiera descripciones procedurales: un lenguaje procedural de alto nivel.

¿Cómo debe ser ese lenguaje? Hemos considerado los mejores lenguajes procedurales disponibles en el mercado y analizado sus características. Se encontraron construcciones interesantes como la conocida **For Each ... End For**, que permite trabajar en forma simple y a buen nivel con conjuntos de registros.

Sin embargo, todos los lenguajes analizados tienen un problema serio: en ellos es necesario referirse a elementos físicos de bajo nivel como tablas y, por ello, el desarrollador debe saber en el momento de escribir el programa, de qué tabla debe considerar cada atributo.

Para GeneXus ésta es una característica inaceptable porque una consecuencia de ella sería: **¡si un atributo cambia de tabla la descripción deja de ser válida!**

Dicho de otra manera: las tablas no integran el Modelo Externo y, por ello, cualquier especificación que las contenga puede no ser ortogonal, por lo que no podríamos asegurar el mantenimiento automático de las aplicaciones generadas.

La solución fue diseñar un lenguaje que utiliza sintaxis inspirada en las más avanzadas construcciones de los mejores lenguajes pero donde en ningún caso se utilice como argumento cualquier elemento que no pertenezca al Modelo Externo.

Así, por ejemplo, en vez de referirnos a tablas, nos referiremos a los conjuntos de atributos necesarios para ejecutar una acción dada y es el sistema en tiempo de generación automática, el que determina cuáles son las tablas involucradas y, si es el caso, los índices utilizados y como hacerlo y, con ello, genera el programa.

No entraremos aquí al detalle de la sintaxis del lenguaje. Es un lenguaje procedural, con instrucciones para manipular los datos que no pierden validez ante los cambios estructurales de la base de datos y que tienen las instrucciones lógicas y aritméticas normales de un lenguaje procedural de alto nivel.

De esta manera se han podido resolver todos los problemas casuísticos encontrados, respetando siempre las líneas maestras de consistencia y ortogonalidad de GeneXus.

¿Cuales son los principales puntos altos y bajos de este objeto?

Punto muy alto: sus capacidades procedurales complementan las capacidades declarativas de las Transacciones (y, según veremos más delante de otros objetos GeneXus) y nos aseguran la resolución de cualquier problema de sistemas de negocios.

Punto bajo: la tarea de escribir descripciones procedurales permite una productividad menor que la de hacer descripciones declarativas, como es el caso de las Transacciones.

Para resolver esta situación últimamente se han agregando y se agregarán en un futuro próximo, elementos que disminuyen mucho la necesidad de descripciones procedurales.

Los “objetos-tipo” más importantes son “**Transacción**” y “**Procedimiento**”. Con ellos se puede hacer una muy razonable descripción de la realidad aunque, con el tiempo, se ha introducido un conjunto importante de nuevos objetos, algunos para lograr que nuestra descripción sea completa (**GXflow** que permite describir los flujos de trabajo de las operaciones del negocio y generar el código necesario para utilizar el servidor especializado en administrarlos), otros para permitir diálogos más amigables con los usuarios a medida que las nuevas arquitecturas disponibles lo permitían (**Work Panels, Web Panels, GXportal**), otros para facilitar la reutilización

del conocimiento permitiendo un aumento de productividad muy importante (**Business Components**, **Data Providers**, **Mini-Pros**, **Patterns**), otros para llevar al terreno de los usuarios finales la formulación de las consultas a las Bases de Datos (**GXquery**) o a las DataWarehouses (**GXplorer**). Estos objetos pueden combinarse para resolver las necesidades más complejas.

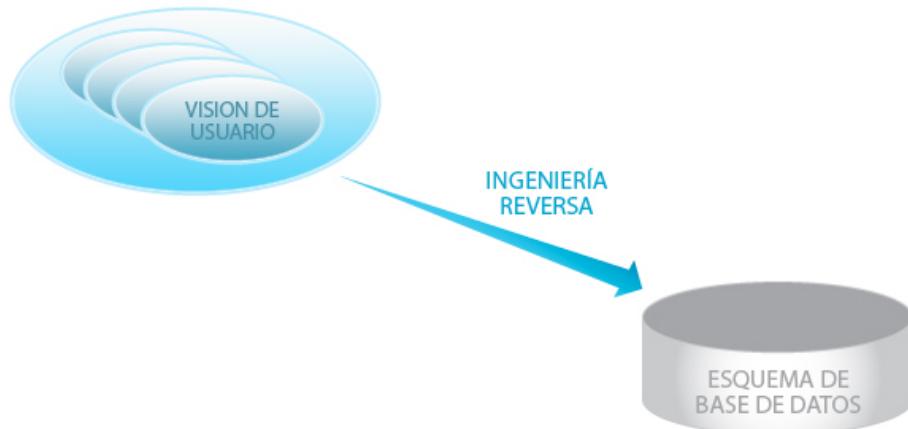
¿Es completo el conjunto de “objetos-tipo” de GeneXus?: No es algo que pueda demostrarse teóricamente. Empíricamente diremos que lo ha sido: los más de 40000 desarrolladores GeneXus en todo el mundo, desarrollan sistemas importantes 100% con GeneXus, lo que constituye una muestra muy representativa de la comunidad informática en general y de la propia realidad y nos muestra que sí, que es completo para las necesidades de los sistemas de negocios de hoy.

Pero no es un conjunto cerrado: probablemente en el futuro aparezcan nuevas necesidades, que nos lleven a introducir nuevos “objetos-tipo”, para satisfacerlas.

LA IMPORTANCIA DEL MODELO RELACIONAL EN LA CONSTRUCCIÓN DE LA KNOWLEDGE BASE

Dado un conjunto de visiones de datos es razonable pensar que exista un único modelo relacional mínimo que lo satisfaga. No lo demostraríamos aquí, pero si esta aserción es cierta, podemos encontrar un procedimiento de ingeniería inversa que, partiendo del conjunto de visiones de datos nos dé como resultado el esquema de dicha base de datos relacional mínima.

Lograr éste procedimiento de ingeniería inversa ha sido el primer gran resultado de investigación del proyecto GeneXus. Luego, construir la necesaria Knowledge Base sobre él ha sido un muy importante y exitoso trabajo permanente.



MODELO EXTERNO Y KNOWLEDGE BASE

La **Knowledge Base** inicialmente tiene asociado un conjunto mecanismos de inferencia y contiene reglas generales [5] que son independientes de cualquier aplicación particular. Al describir la realidad del usuario objeto se almacenan las descripciones en el **Modelo Externo**.

El sistema, automáticamente, captura todo el conocimiento contenido en el **Modelo Externo** y lo sistematiza, agregándolo también a la **Knowledge Base**. Adicionalmente, sobre el conocimiento anterior, el sistema infiere lógicamente un conjunto de resultados que ayudan a mejorar la eficiencia de las inferencias posteriores.

GeneXus trabaja permanentemente sobre la Knowledge Base. Todo el conocimiento de la Knowledge Base es equivalente al contenido en el Modelo Externo (subconjunto de ella), ya que consiste en el propio Modelo Externo más reglas y mecanismos de inferencia independientes de dicho Modelo Externo y un conjunto de otros elementos que son automáticamente inferidos a partir del mismo.

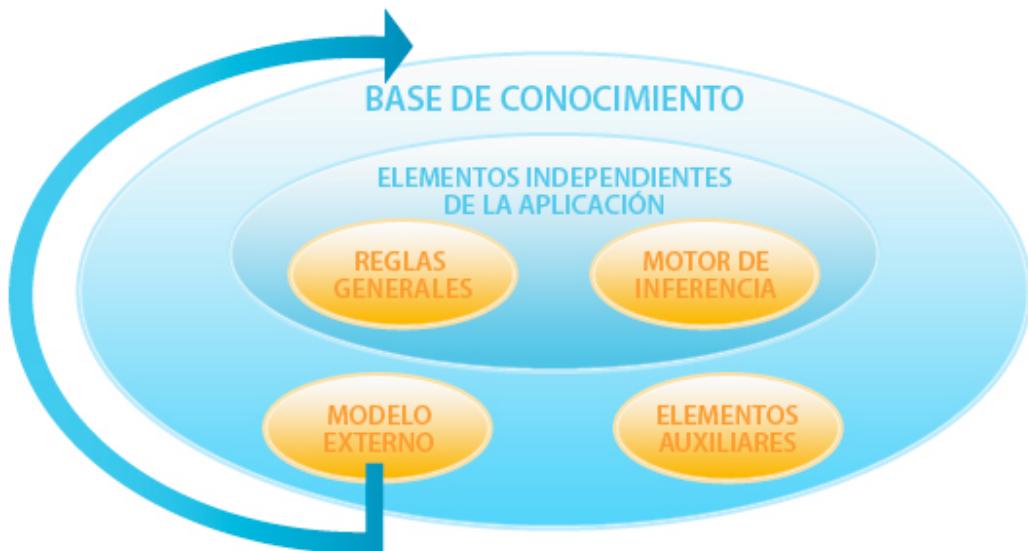
El desarrollador puede alterar, modificando objetos de la realidad del usuario, el Modelo Externo y las modificaciones se propagarán automáticamente a todos los elementos que lo necesiten: otros elementos de la Knowledge Base, Base de Datos y programas de la aplicación. De la misma manera, el desarrollador no puede alterar directamente ningún elemento que no pertenezca al Modelo Externo.

Este trabajo se ocupa únicamente del **Modelo Externo** (el “**qué**”) y se abstiene de tratar la **Knowledge Base**, que lo contiene y lo mantiene, (y que forma parte del “**cómo**”: un sofisticado conjunto de herramientas matemáticas y lógicas sobre las que se basan los procesos de inferencia y resultados intermedios utilizados para aumentar la eficiencia de dichas inferencias). No es necesario conocer nada del “cómo” para hacer un muy buen uso de GeneXus.

¿Qué es más importante?: ¿el “qué” o el “cómo”? ninguno funciona sin el otro, pero si el “qué” no es correcto, todo está equivocado.

De lo anterior se puede inferir algo muy importante, que va mucho más allá de la implementación actual de GeneXus: **TODO** el conocimiento está contenido en el Modelo Externo y, por ello, mañana podríamos soportar la **Knowledge Base** de una manera totalmente diferente y el conocimiento de nuestros clientes seguiría siendo utilizable sin problema alguno.

En resumen: La Knowledge Base y los mecanismos de inferencia asociados constituyen una gran “máquina de inferencia”. Un buen ejemplo de esto es: dada una visión de datos podemos inferir en forma totalmente automática el programa necesario para manipularla.



DÍA “D” Y PROTOTIPACIÓN

La prototipación es un gran recurso y utilizándolo adecuadamente se evita una exagerada dependencia de la prueba final o, aún, de la puesta en marcha (el Día “D”, donde todo ocurre y la Ley de Murphy se aplica especialmente).

Por las características de GeneXus, todo se puede probar en cualquier momento y, en particular, se puede prototipar cada objeto en el momento oportuno para hacerlo. Esta posibilidad es siempre muy importante y es consecuencia directa de la teoría de GeneXus y, en particular, de su propagación automática de los cambios, característica muy importante y exclusiva de GeneXus.

RESUMEN DE CONCEPTOS BÁSICOS DE GENEXUS

Tratamiento automático del conocimiento. Se basa en la existencia de un modelo de la realidad objeto: **Modelo Externo** con el cual un software de computador pueda trabajar automáticamente. En particular, que permita:

Consistencia: **Modelo siempre consistente.**

Ortogonalidad: los objetos que constituyen el Modelo Externo son independientes entre sí. La adición de uno nuevo, su modificación o eliminación no implicarán la necesidad de modificar ningún otro objeto.

Generación y mantenimiento automáticos de la base de datos y los programas.

Modelo Externo: El conocimiento esencial corresponde a un "Modelo Externo", que no puede contener ningún elemento físico o interno: archivos, tablas, entidades, relaciones entre entidades, índices o cualquier elemento que pueda deducirse automáticamente de dicho "Modelo Externo".

Integración: GeneXus tiene un conjunto de objetos propios que están totalmente integrados por concepción pero a partir de su versión Rocha permitirá otros niveles de integración: la de otros módulos o productos desarrollados por Artech o por terceros.

TENDENCIAS DE GENEXUS

GeneXus es un producto con una permanente evolución. Las tendencias de esta evolución las podemos representar sobre cuatro dimensiones: **COMPLETITUD, PRODUCTIVIDAD, UNIVERSALIDAD y USABILIDAD.**

COMPLETITUD: Mide el porcentaje de código del sistema del usuario que es generado y mantenido automáticamente por GeneXus.

Desde 1992 la completitud alcanzada por GeneXus es siempre de un 100%, Éste es un compromiso fundamental porque implica una propagación automática de los cambios, que implica una disminución dramática de los costos de desarrollo y mantenimiento.

UNIVERSALIDAD: Mide la cobertura de las diferentes plataformas importantes disponibles en el mercado. Hoy GeneXus soporta todas las plataformas “vivas”, entendiéndose por plataformas vivas aquellas que tienen una importante y creciente base instalada.

De esta manera, GeneXus brinda a los clientes una gran libertad de elección: si una aplicación es desarrollada con GeneXus el cliente puede siempre pasarla a otra plataforma soportada sin costos importantes.

PRODUCTIVIDAD: Mide el aumento de productividad del desarrollo con GeneXus sobre el desarrollo con programación manual.

El objetivo de aumento de productividad de GeneXus fue, durante muchos años, de un 500%, lo que era suficiente.

Pero los sistemas que necesitan los clientes son cada día más complejos porque existen nuevos dispositivos y nuevas necesidades, sobre todo de sistemas que cumplan con las necesidades, por complejas que sean, pero se mantengan simples para los usuarios que, en general, no serán entrenables.

Adicionalmente, esos sistemas deben ser desarrollados cada vez en menos tiempo: el “time to market” es cada vez más crítico.

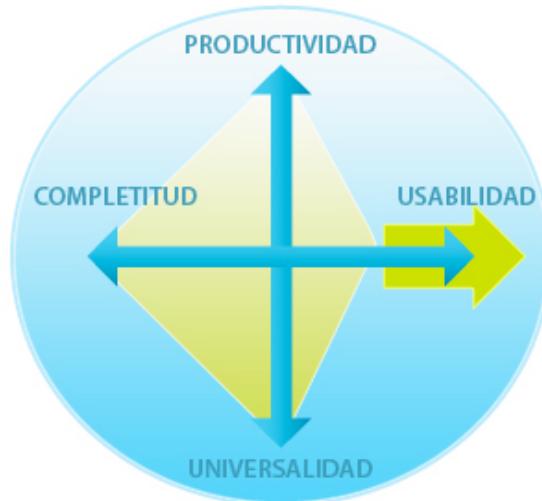
En el año 2004 se modificaron esos objetivos pasando al 1000% para la versión 9 y el 2000% para la versión Rocha.

USABILIDAD: Mide la facilidad de uso. Y pretende hacerlo con carácter general: facilitar el uso a los usuarios técnicos, pero también aumentar el alcance permitiendo que cada vez más usuarios no técnicos puedan utilizar GeneXus con provecho.

El solo hecho de utilizar GeneXus implica un gran aumento de la usabilidad: un desarrollador GeneXus puede desarrollar excelentes aplicaciones para una determinada plataforma sin necesitar de un conocimiento profundo de esa plataforma lo que significa un nivel mínimo importante de usabilidad.

Pero pretendemos ir mucho más allá: pretendemos que muchos más usuarios, cada vez con menos pre requisitos técnicos, puedan beneficiarse del uso de GeneXus.

La versión Rocha mejorará fuertemente la usabilidad y esa tendencia seguirá en las próximas versiones.



¿CÓMO SEGUIMOS?, ¿ESTÁ NUESTRO MODELO EXTERNO ESCRITO EN PIEDRA?

Ciertamente no, los principios son permanentes y la experiencia muestra la validez de lo hecho. Pero siempre debemos observar a GeneXus críticamente de manera de ampliarlo para que nos permita describir de la manera más sencilla posible los nuevos casos que nos vaya planteando la realidad.

BIBLIOGRAFÍA Y REFERENCIAS

[1] ANSI-SPARC: [Final report of the ANSI/X3/SPARC DBS-SG relational database task group](#) (portal ACM, citation id=984555.1108830)

[2] WARNIER-ORR: **Warnier, J.D. Logical Construction of Programs.** Van Nostrand Reinhold, N.Y., 1974.; **Orr, K.T. Structured Systems Analysis.** Englewood Cliffs, NJ, 1977: Yourdon Press; [Kenneth T. Orr, Structured Systems Development](#), Prentice Hall PTR, Upper Saddle River, NJ, 1986.

[3] JACKSON: **Jackson, M.A. Principles of Program Design.** London: Academic Press, 1975.

[4] CODD: [E. F. Codd, A relational model of data for large shared data banks](#), Communications of the ACM, v.13 n.6, p.377-387, June 1970

[5] REGLAS:

Sin perjuicio de las reglas particulares que tiene asociado el modelo de una determinada empresa, existen reglas generales, de validez permanente e independientes de cualquier aplicación. A título de ejemplo, veremos lo siguiente:

Necesidad de la Consistencia: No pretendemos aquí tratar el tema detalladamente pero la principal fuente de reglas, como en todas las ciencias, es la consistencia: suponemos que la realidad es consistente y, como consecuencia, toda representación que hagamos de ella necesariamente debe serlo.

Partiendo de éste principio, podemos inferir muchas reglas de validez general que enriquecerán nuestro modelo.

Un caso particular, simple pero muy importante, es el de la **integridad referencial**:

1. Dado X, atributo simple o compuesto, que es llave de una determinada tabla T1, si X aparece también en una tabla T2, dada una fila de T2, debe existir una fila de T1 donde T1.X = T2.X
2. Dado X, atributo simple o compuesto, que es llave de una determinada tabla T1, e Y, atributo simple o compuesto tal que Y = subtipo(X), si Y aparece en una tabla T2, dada una fila de T2, debe existir una fila de T1 donde T1.X = T2.Y
3. Dado Y, atributo simple o compuesto, que aparece en una Tabla T2 tal que no existe una tabla T1 donde Y es llave ni X (tal que Y = subtipo(X)) es llave , Y sólo puede aparecer en la Tabla T2.

CONTENIDO

INTRODUCCIÓN	1
NUEVO PARADIGMA: DESCRIBIR EN VEZ DE PROGRAMAR	3
MODELO DE DATOS	4
MARCO DE REFERENCIA.....	6
DESCRIPCIÓN DE LA REALIDAD	8
MODELO EXTERNO Y KNOWLEDGE BASE.....	12
RESUMEN DE CONCEPTOS BÁSICOS DE GENEXUS.....	14
TENDENCIAS DE GENEXUS	14
BIBLIOGRAFÍA Y REFERENCIAS	16