DESENVOLVIMENTO BASEADO NO CONHECIMENTO

FILOSOFIA E FUNDAMENTOS TEÓRICOS DO GENEXUS

por Breogán Gonda e Nicolás Jodal

Maio de 2007

RESUMO: TENTA-SE MOSTRAR, DE UMA MANEIRA SIMPLES E SISTEMÁTICA, A FILOSOFIA E OS EMBASAMENTOS TEÓRICOS DO GENEXUS. BOA PARTE DESTES ELEMENTOS JÁ SÃO CONHECIDOS PELA COMUNIDADE GENEXUS, MAS ESTÃO COLETADOS PARCIALMENTE EM DIVERSOS DOCUMENTOS.

INTRODUÇÃO

O propósito deste documento é ajudar a entender as idéias básicas e os conceitos teóricos que fizeram possível GeneXus, para compreender melhor a sua essência e os novos desenvolvimentos que irão se produzir no futuro.

O primeiro propósito do GeneXus era estabelecer um robusto modelo de dados. Uma vez construído esse modelo com todo rigor, pôde-se pensar numa ampliação, incorporando elementos declarativos como regras, fórmulas, telas, etc., tentando descrever as visões de dados dos usuários e, assim, abranger todo o conhecimento dos sistemas de negócios.

Por que se tentou abranger todo o conhecimento dos sistemas de negócios? Porque é uma condição necessária para poder projetar, gerar e manter em forma 100% automática os sistemas computacionais de qualquer empresa. Era evidente que num determinado momento a comunidade informática optaria por este caminho e que seria bom antecipar-se em percorrê-lo tanto quando fosse possível.

Mas, como abranger todo o conhecimento dos sistemas de negócios? Como abranger a casuística que têm os sistemas de negócios?

A primeira versão do GeneXus (100% declarativa) pretendia resolver o problema de geração e manutenção de 70% dos programas. O sistema se ocupava deles em forma automática. Os 30% restantes deviam ser programados e mantidos manualmente.

Os "prospects" entenderam que resolver automaticamente 70% de seu problema era muito bom e até duvidaram que a manutenção automática fosse possível: os primeiros em adotar o GeneXus o fizeram exclusivamente pelo aumento de produtividade no desenvolvimento. Entretanto, alguns meses mais tarde, esses clientes apreciaram o grande aumento de produtividade que conseguiam no desenvolvimento, mas apreciaram ainda muito mais a manutenção automática.

Um ano depois do lançamento, o projeto GeneXus começou a sofrer uma forte pressão de seus clientes (o sucesso da manutenção automática e as vantagens que isso acarretava fizeram que eles exigissem "manutenção automática para tudo" o que, como pré-condição exigia "geração automática de tudo").

Isso não tinha antecedentes no mercado internacional (realmente, muitos anos depois, o GeneXus continua sendo o único produto no mundo que resolve estes problemas).

Mas, então, qual é hoje o objetivo do GeneXus? **O objetivo do GeneXus é conseguir um ótimo tratamento automático do conhecimento dos sistemas de negócios**.

Eis a essência. Atingido esse objetivo, existe um grande conjunto de subprodutos como:

- Projeto, geração e manutenção automáticos da base de dados e dos programas de aplicação necessários para a empresa.
- Geração, a partir do mesmo conhecimento, para plataformas múltiplas.
- Integração do conhecimento de diversas fontes para atender a necessidades muito complexas com custos em tempo e dinheiro bem inferiores aos habituais.

Geração para as tecnologias futuras, quando essas tecnologias estiverem disponíveis, a partir do conhecimento que hoje abrigam os clientes GeneXus nas suas Knowledge Bases: GeneXus trabalha com o conhecimento puro, cuja validez é totalmente independente das tecnologias da moda.

O desenvolvimento todo seguiu certas linhas mestras. Este documento tem o propósito de explicitar estas linhas mestras, de uma maneira simples e conceitual.

NOVO PARADIGMA: DESCREVER EM VEZ DE PROGRAMAR

Como pretendemos descrever a realidade?

Pretende-se "descrever" **em vez de** "programar". **Pretende-se maximizar as descrições declarativas e minimizar as especificações procedurais.**

Por que "descrever" **em vez de** "programar"**?**

Esta pretensão constitui uma mudança essencial de paradigma e implica um choque cultural. As mudanças de paradigma têm o problema de sua lenta aceitação mas, se forem corretos (e este sem dúvida o é!), em determinado momento ocorrem os fatos que acarretam sua rápida aceitação geral.

Quais são esses fatos a que nos referimos?: Hoje, a grande maioria dos sistemas se desenvolve e mantém com programação manual mas, qual tem sido a evolução do mercado nos últimos 40 anos (dados ao ano 2006)?

A complexidade dos sistemas aumentou em 2000%

A produtividade das linguagens de programação aumentou em 150%

Esta situação é insustentável para os negócios. Os custos crescem em forma desmedida.

Podemos separar esses custos em duas dimensões: **Dinheiro** e **Tempo**.

As empresas descobriram, graças à disponibilidade de comunicações rápidas e baratas, que podem atuar sobre a dimensão **Dinheiro** contratando o desenvolvimento e manutenção de seus sistemas em países de baixos salários e assim o fizeram intensamente nos últimos anos.

Na realidade, não foi resolvida a questão dos custos, mas trocou-se a unidade de medida: são basicamente as mesmas horas de trabalho (é essencialmente o mesmo trabalho), mas o preço da hora-homem é bem menor. Se medirmos o custo em dinheiro, constataremos um forte abatimento.

O que acontece com a dimensão **Tempo**? O precedente não é aplicável aos tempos.

Os sistemas de negócios precisam responder a novas necessidades: novos dispositivos, novos usuários, novas modalidades de uso, novas possibilidades de integração e, por isso, tornam-se cada dia mais e mais complexos e os negócios estão sujeitos a um "time to market" cada vez mais crítico e que não pode mudar. Como conseqüência, cada dia mais negócios estão se prejudicando pelos tempos inadequados de desenvolvimento das novas soluções e de manutenção das atuais. E hoje -mas sobre tudo num futuro próximo- ninguém pode fazer bons negócios sem os sistemas adequados.

Esta situação não pode continuar assim por muito tempo, e ao poucos, vêm aparecendo exemplos disso tais como:

Em quase todos os países mais desenvolvidos, muitas empresas estão cada vez mais lançando mão do "outsourcing", geralmente acompanhado pelo "offshoring" do desenvolvimento de seus sistemas para países de baixos salários, porém, ao mesmo tempo, há uma corrente crescente de empresas que seguem o caminho inverso visto que não obtiveram resultados convenientes com o mencionado "outsourcing".

A cada vez mais existem empresas que experimentam grande violência demitindo seus técnicos e transferindo seu

trabalho para técnicos desconhecidos de países longínquos. Evitar isso exige que os custos "in house" sejam similares aos do "outsourcing", o que só será possível adotando tecnologias de muito alto nível.

É necessário um dramático aumento da produtividade, porém, a produtividade das linguagens de programação há muito tempo já atingiu uma estabilização.

Como resolver o problema? Com melhores linguagens de programação? Os fatos demostram que não! O problema não está nas linguagens de programação, mas na própria programação.

Como obter, então, o aumento de produtividade necessário? Fazendo desenvolvimento sustentado em conhecimento e não em programação: a solução é **descrever em vez de programar**!

Na Artech temos certeza disso e estamos preparados para essa explosão do mercado do desenvolvimento sustentado em conhecimento que se irá se produzir nos próximos anos.

MODELO DE DADOS

Historicamente a comunidade informática começou trabalhando com apenas um modelo físico com muitas rigidezes. Depois, foi introduzindo outros modelos com o objetivo de ganhar maior flexibilidade e obter certa permanência das descrições. O trabalho mais importante sobre a questão o foi o relatório de 1978 do grupo **ANSI SPARC [1]** que introduz três modelos:

Modelo Externo no qual se representam as visões externas.

Modelo Lógico ou Conceitual: É outro modelo (geralmente um modelo E-R) que se pretendia obter por abstração da realidade.

Modelo Físico: referia-se fundamentalmente ao esquema da base de dados.

A idéia era desenvolver paralelamente os três modelos para que os programas só interagissem com o Modelo Externo e, a partir do Modelo Lógico fazer um "mapping" entre esses programas e a base de dados (Modelo Físico) e com isso tornar independentes os programas da base de dados.

O relatório foi muito elogiado no momento e, depois, quase esquecido. Só um fabricante a inícios da década de 80 tentou implementar este esquema de três modelos. Foi o Cincom Systems para seu produto **SUPRA** e para diferentes linguagens acessando a base de dados SUPRA.

Fora isso, o relatório ANSI SPARC continua sendo válido até hoje. A maior diferencia é a tecnologia disponível. Nos anos que se passaram, as mudanças tecnológicas têm sido muito importantes.

Analisando a questão à luz da tecnologia atual, concluímos que podem existir vários modelos, mas **o modelo realmente importante para os usuários e os desenvolvedores, é o Modelo Externo**: nele coletamos o conhecimento exterior e o resto, como outros modelos auxiliares que poderiam ajudar, deve e pode se inferir automaticamente a partir do mencionado Modelo Externo.

Neste trabalho enfatizamos no **Modelo Externo**, que contém o conhecimento genuíno e no qual se assenta a essência e a singularidade do GeneXus (o "**quê**"). Face a novas possibilidades da tecnologia e necessidades dos clientes, o primeiro a fazer é ampliar esse Modelo Externo.

CARACTERÍSTICAS QUE DEVE TER NOSSO MODELO

Certamente os objetos-tipo do GeneXus podem evoluir no futuro obedecendo, no entanto, a certos princípios que seria bom mencionar aqui.

Mas existe uma pergunta prévia: quem são os detentores do conhecimento nas organizações?

O único conhecimento objetivo e bastante detalhado é aquele que os usuários possuem a todo nível sobre as visões dos dados que eles utilizam, precisam ou desejam. Não existe, em troca, conhecimento objetivo sobre os próprios dados.

É por isso que **devemos privilegiar o concreto sobre o abstrato**, porque os usuários, que podem ser múltiplos e ter os mais diversos perfis e papéis na empresa, lidam bem com elementos concretos de seu ambiente e nem tanto com conceitos abstratos: a representação do conhecimento deve ser simples, detalhada e objetiva.

Parece adequado basear-nos em um "Modelo Externo" que reúna estas visões. Um "Modelo Externo" não pode conter nenhum elemento físico ou interno tais como arquivos, tabelas, entidades, relações entre entidades, índices ou qualquer outro que possa ser deduzido automaticamente do mesmo.

Quais elementos são desejáveis e quais elementos são inaceitáveis no GeneXus?

Estamos tentando criar um modelo com as seguintes características:

Conhecimento adequado para seu tratamento automático. Modelo com o qual um software de computador possa trabalhar automaticamente. Ou seja, o conhecimento de nosso modelo deve ser "compreensível" e "operável" automaticamente.

Consistência. Modelo sempre consistente.

Ortogonalidade. Os objetos que num determinado momento constituem o modelo devem ser independentes entre si. A adição de um novo objeto, sua modificação ou eliminação não acarretará a necessidade de modificar nenhum outro objeto.

Projeto, geração e manutenção automáticos da base de dados e dos programas. Dentre as coisas que um software de computador pode fazer automaticamente com o modelo, são essenciais o projeto, a geração e a manutenção automáticos da base de dados e dos programas.

Escalabilidade. Os itens anteriores expressam as características qualitativas que deve ter nosso modelo. Mas pretende-se resolver problemas grandes. Para isso, precisamos que os mecanismos de armazenamento, acesso e inferência do modelo atuem eficientemente com independência do tamanho do problema.

MARCO DE REFERÊNCIA

Como representar o conhecimento de uma maneira rigorosa? Devemos estabelecer um conjunto de objetos-tipo predefinidos através dos quais seja possível representar bem a realidade e que sejam automaticamente

compreensíveis e operáveis.

Um caminho que seguiu a indústria na maioria dos casos, é partir de um modelo de dados E-R ou similar. Mas como obter o modelo de dados correto? Um modelo de dados corporativo precisa de centenas ou ainda milhares de tabelas. Tradicionalmente, os modelos de dados eram desenhados mediante um processo de abstração (através de interlocutores que tivessem o conhecimento suficiente da empresa, identificar seus "objetos relevantes" e suas "relações" e introduzir no modelo as correspondentes "entidades" e "relações").

Este processo de abstração era fortemente subjetivo: quem, na organização, tinha um bom conhecimento dos dados? Quem tinha um bom conhecimento dos "objetos" e "relações" "relevantes" que constituíam o input do modelo de dados? A resposta é clara e contundente: ninguém. Era preciso, então, solicitar o conhecimento de multiplicidade de pessoas e cada uma delas ia adicionando sua própria subjetividade.

Onde estava o conhecimento objetivo e suficientemente detalhado necessário para construir nosso modelo? Certamente não existia na organização esse conhecimento sobre os dados. Entretanto, cada usuário, a qualquer nível, conhecia muito bem as "visões de dados" que utilizava, que necessitava, ou que desejava: **ninguém conhece os "dados" mas todos trabalham permanentemente com visões desses dados**.

Optou-se por tomar essas "visões de dados" como input básico do modelo mas, como descrevê-las objetivamente?

Era necessário estabelecer um sólido marco de referência sobre o qual fazer as demais descrições.

Esse marco de referência está constituído pelos **Atributos**.

Elemento semântico fundamental: Atributo

Existem atributos, cada atributo tem um nome, um conjunto de características e um significado.

Para se ter um modelo de grande potência expressiva, além dos elementos sintáticos que toda a comunidade informática maneja sem dificuldades, devemos representar nele elementos semânticos. Mas é difícil trabalhar automaticamente com a semântica.

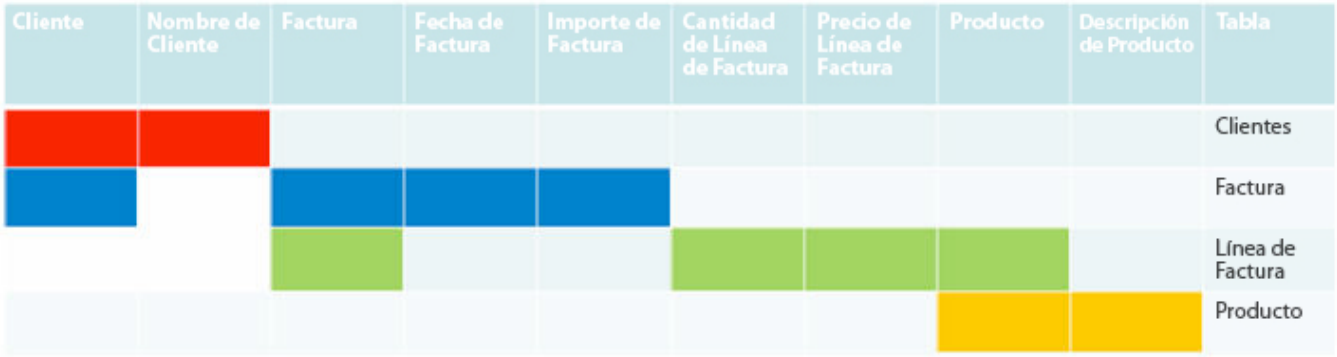
Para tratar automaticamente um elemento semântico, precisamos dar-lhe uma representação sintática.

Escolheu-se um único elemento semântico para ser representado em nosso modelo: o **significado** de cada atributo.

O segundo elemento básico é então: "significado", que no nosso modelo definimos da seguinte maneira: **significado (atr) = nome (atr)**

Os nomes dos atributos passam a ser essenciais para o modelo São necessárias normas claras para que sejam atribuídos de maneira a garantir sua consistência através do modelo todo.

A esses efeitos, adotamos a **URA (Universal Relational Assumption)** que, essencialmente, significa que um atributo terá o mesmo nome em todos os lugares onde ele vier a aparecer, e que não haverá dois atributos diferentes (com significado diferente) compartilhando o mesmo nome. Além disso, é necessário que os nomes sejam auto-explicativos e muito fáceis de entender por terceiros.

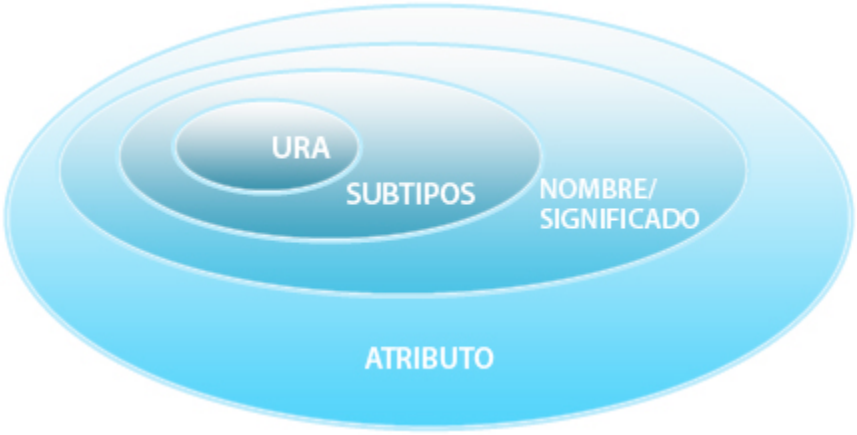


O terceiro elemento básico é, então, a URA (Universal Relational Assumption).

A realidade nos mostrou que a URA não basta, portanto, estabelecemos **subtipos de atributos** e, a seguir, **subtipos de grupos de atributos**.

O quarto elemento está constituído, então, pelos "subtipos".

RESUMINDO: o **atributo**, tal qual o caracterizamos aqui (com as considerações sobre **significado**, **URA** e **subtipos**), é o elemento semântico fundamental da teoria do GeneXus. Os **atributos** constituem o marco de referência sobre o qual edificamos os modelos GeneXus.



DESCRIÇÃO DA REALIDADE

Depois de introduzir o marco de referência, descreveremos a realidade através de um conjunto de instâncias de "objetos-tipo", que especificaremos com base nos seus atributos e mediante os quais iremos coletar as características da realidade a representar.

TRANSAÇÕES

Cada usuário tem uma ou múltiplas visões dos dados que utiliza cotidianamente.

Entre estas visões, podemos pensar num primeiro tipo: aquele que reúne aquelas que se utilizam para manipular os dados (introduzir, modificar, eliminar e visualizá-los em forma limitada), estas visões de usuários são chamadas de **Transações** e constituem o primeiro objeto-tipo do GeneXus.

Cada transação tem um conjunto de elementos:

Estrutura dos dados da Transação: os dados se apresentam conforme uma estrutura e devemos ter uma maneira de coletar essas estruturas com todo rigor. A seguir, veremos a estrutura de uma transação que todos conhecemos: a de uma Nota Fiscal simples.

[FATURA]
*Código_de_Fatura **
Data_de_Fatura
Nome_de_Cliente
Endereço de_Cliente
*(Código_de_Produto**
Descrição_de_Produto
Quantidade_Vendida_Produto_na_Fatura
Preço_Venda_ Produto_na_Fatura
Importe_Linha_Fatura)
Sub_Total_Fatura
Desconto_Fatura
IVA_Fatura
Total_Fatura

Nesta representação existem três grupos de atributos: um "prólogo ou cabeçalho", que ocorre uma única vez e que aparece no início, um "corpo" que ocorre um certo número de vezes, e um "epílogo" ou "pé" que ocorre uma única vez. Dentro deste prólogo, ao lado do atributo, aparece ***Código_de_Fatura*** um *****, o que significa que para cada ***Fatura*** (cada ocorrência do Prólogo e do Epílogo) existe um único ***Código_de_Fatura;*** depois, há entre parênteses um conjunto de atributos: isso significa que este conjunto de atributos pode ocorrer múltiplas vezes dentro de cada ***Fatura*** e costumamos chamar este grupo repetitivo de corpo: o corpo deve ter um identificador, que identifica bem cada uma de suas linhas dentro da ***Fatura.*** Neste caso o é ***Código_de_Produto*** e essa condição é apontada colocando um * à direita. Depois do encerramento do grupo repetitivo encontramos um epílogo ou pé.

Este Diagrama de Estrutura de Dados bem como o utilizado atualmente pelo GeneXus que tem características mais gráficas, tomam como antecedente os introduzidos nos trabalhos do Warnier-Orr **[2]** e Jackson **[3]**.

Regras: provavelmente haverá um conjunto de regras que afetam à transação, como as seguintes.

Não haverá duas Faturas com o mesmo ***Código_de_Fatura***.

O ***Código_de_Fatura*** será atribuído correlativamente.

A ***Data_de_Fatura*** tomará como opção por defeito a data do dia de sua emissão.

A ***Data_de_Fatura*** não poderá ser menor à data de emissão da mencionada Fatura.

Não será admitida nenhuma ***Fatura*** que deixar negativo o ***Estoque_do_Produto*** para algum de seus produtos.

Não será admitida nenhuma ***Fatura*** que faça que o ***Saldo_Devedor_do_Cliente*** envolvido ultrapasse seu

limite de crédito.

Quando a aceitação de uma Fatura determine que o ***Estoque_do_Produto*** < ***Ponto_de_Pedido_Produto,*** para algum de seus produtos, deverá ser ativada a rotina de ***Abastecimento (Produto).***

Fórmulas: provavelmente haverá um conjunto de fórmulas que afetam a transação, como as seguintes.

Importe_Linha_Fatura = Quantidade_Vendida_Produto_na_Factura * Preço_Venda_Produto_na_Fatura

Sub_Total_Fatura = sumatória dentro da ***Fatura (Código_de_Fatura)*** de ***Importe_Linha_Fatura***

Desconto_Fatura = função de cálculo do desconto (***Código_de_Fatura)***

IVA_Factura = função de cálculo do IVA ***(Sub_Total_Fatura – Desconto_Fatura)***

Total_Fatura = Sub_Total_Fatura – Desconto_Fatura + IVA_Fatura

Saldo_Devedor_do_Cliente = sumatória de faturas impagas ***(Código_de_Cliente)***

Elementos de apresentação: É necessário associar ao objeto elementos de apresentação como formatos de telas para seu desdobramento no Windows ou num Browser, estilos gráficos e de diálogo, etc.

As transações são declarativas e capturando o conhecimento que existe nelas e sistematizando-o numa Knowledge Base obtém-se boa parte da descrição da realidade procurada.

CONSIDERAÇÕES SOBRE AS TRANSAÇÕES E A TEORIA RELACIONAL DO CODD [4].

Uma interpretação ligeira deste objeto, pode levar a pensar que nosso modelo entra em franca contradição com o Modelo Relacional do Codd: existem novos elementos como fórmulas, que não inclui o Modelo Relacional do Codd e grupos repetitivos, explicitamente excluídas nele.

O Modelo Relacional e o nosso modelo têm objetivos diferentes. Isso será visto mais adiante, mas devemos ter claro que em nenhum momento Codd afirmou que "a realidade está composta por um conjunto de relações", o que seria um absurdo, mas sim que "para armazenar adequadamente os dados e operar com eles, o modelo relacional é muito adequado".

O Modelo Relacional está voltado a obter uma boa representação dos dados numa Base de Dados, obedecendo a algumas condicionantes muito desejáveis: eliminar a redundância e introduzir um pequeno conjunto de regras, e evitar assim as maiores fontes de inconsistência dos dados e introduzir um conjunto de operadores que permitam manipular os dados a bom nível.

O Modelo Externo será utilizado para obter e armazenar o conhecimento. Este modelo pretende a representação mais direta e objetiva possível da realidade. Por isso, tomamos as visões dos diferentes usuários. Estas visões são armazenadas no modelo. A seguir, captura-se todo o conhecimento contido nelas, sistematizando-o para maximizar as capacidades de inferência.

Neste contexto, o Modelo Relacional (ou melhor, um super-conjunto dele) é utilizado para representar e/ou manipular os dados: corresponde ao chamado "Modelo Interno ou Físico" ao que refere o relatório ANSI SPARC mencionado.

Na nossa teoria utiliza-se fortemente o Modelo Relacional.

PROCEDIMENTOS

As Transações, como foi visto, permitem de maneira declarativa descrever as visões de dados dos usuários e, como conseqüência, entesourar o conhecimento necessário para projetar, gerar e manter de maneira totalmente automática, a Base de Dados e os programas que os usuários necessitem para manipular suas próprias visões de dados.

Mas isto não é suficiente: existe a necessidade de processos "batch" ou similares. Estes processos não se podem inferir a partir do conhecimento fornecido pelas visões de dados.

Quando foi liberada a primeira versão do GeneXus, as transações resolviam cerca de 70% do problema, gerando os programas correspondentes. Faltava 30% dos programas.

Para se obter 100% de cobertura às necessidades dos clientes, foi necessário lançar mão de um objeto que permitisse descrições procedurais: uma linguagem procedural de alto nível.

Como deve ser essa linguagem? Consideramos as melhores linguagens procedurais disponíveis no mercado e analisado suas características. Encontraram-se construções interessantes como a conhecida **For Each** … **End For**, que permite trabalhar em forma simples e em bom nível com conjuntos de registros.

Entretanto, todas as linguagens analisadas têm um problema sério: nelas é necessário referir-se a elementos físicos de baixo nível como tablelas e, por isso, o desenvolvedor deve saber no momento de escrever o programa, de quais tabelas deve considerar cada atributo.

Para o GeneXus esta é uma característica inaceitável porque uma conseqüência dela seria: **se um atributo mudar a tabela, a descrição deixará de ser válida**!

Em outras palavras: as tabelas não integram o Modelo Externo e, por isso, qualquer especificação que as contiver pode não ser ortogonal, por isso não poderíamos garantir a manutenção automática das aplicações geradas.

A solução foi desenhar uma linguagem que utilizasse uma sintaxe inspirada nas mais avançadas construções das melhores linguagens mas na qual em hipótese nenhuma fosse utilizado como argumento algum elemento não pertencente ao Modelo Externo.

Assim, por exemplo, em vez de nos referir a tabelas, referiremos aos conjuntos de atributos necessários para executar uma ação dada e é o sistema, em tempo de geração automática, quem determina quais são as tabelas envolvidas e, se for o caso, os índices utilizados e como fazê-lo e, com isso, gera o programa.

Não entraremos aqui ao detalhe da sintaxe da linguagem. É uma linguagem procedural, com instruções para manipular os dados que não perdem validez em face às mudanças estruturais da base de dados e que têm as instruções lógicas e aritméticas normais de uma linguagem procedural de alto nível.

Desta maneira, foi possível resolver todos os problemas casuísticos encontrados, respeitando sempre as linhas mestras de consistência e ortogonalidade do GeneXus.

> Quais são os principais pontos altos e baixos deste objeto?
>
> **Ponto muito alto**: suas capacidades procedurais complementam as capacidades declarativas das Transações (e, conforme veremos mais adiante de outros objetos GeneXus) e garantem a resolução de qualquer problema de sistemas de negócios.
>
> **Ponto baixo**: a tarefa de escrever descrições procedurais permite menor produtividade que a de fazer descrições declarativas, como é o caso das Transações.

Para resolver esta situação ultimamente tem se adicionando, e irão se adicionar num futuro próximo, elementos que diminuem em muito a necessidade de descrições procedurais.

Os "objetos-tipo" mais importantes são **"Transação"** e **"Procedimento"**. Com eles é possível fazer uma muito razoável descrição da realidade embora, com o tempo, introduziram-se um conjunto importante de novos objetos, alguns dentre eles para conseguir que nossa descrição fosse completa (**GXflow** que permite descrever os fluxos de trabalho das operações do negócio e gerar o código necessário para utilizar o servidor especializado em gerenciá-los), outros para permitir diálogos mais amigáveis com os usuários a medida que as novas arquiteturas disponíveis o permitissem (**Work Painels**, **Web Painels**, **GXportal**), outros para facilitar a reutilização do conhecimento permitindo um aumento da produtividade muito importante (**Business Components**, **Data Providers**, **Mini-Procs**, **Patterns**), outros para levar ao terreno dos usuários finais a formulação das consultas às Bases de Dados (**GXquery**) ou às DataWarehouses (**GXplorer**). Estes objetos podem combinar-se para resolver as necessidades mais complexas.

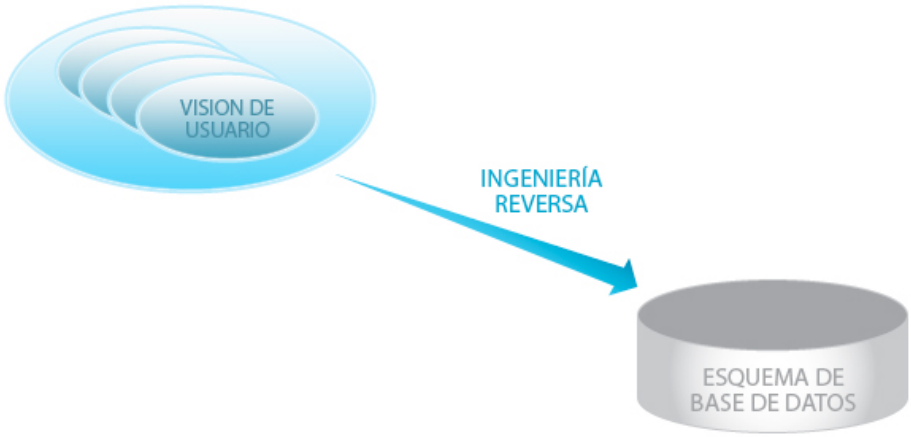
É completo o conjunto de "objetos-tipo" do GeneXus?: Não é algo que possa ser demonstrado teoricamente. Mas podemos afirmá-lo empiricamente: os mais de 40000 desenvolvedores GeneXus no mundo todo desenvolvem sistemas importantes 100% com o GeneXus. Isso constitui uma amostra bem representativa da comunidade informática em geral e da própria realidade e prova que, efetivamente, é completo para as necessidades dos sistemas de negócios de hoje.

Mas não é um conjunto fechado: provavelmente no futuro irão aparecer novas necessidades que nos levem a introduzir novos "objetos-tipo", para satisfazê-las.

A IMPORTÂNCIA DO MODELO RELACIONAL NA CONSTRUÇÃO DA KNOWLEDGE BASE

Dado um conjunto de visões de dados é razoável pensar que exista um único modelo relacional mínimo que o satisfaça. Não o demonstraremos aqui, mas se esta asserção for certa, podemos encontrar um procedimento de engenharia inversa que, partindo do conjunto de visões de dados, dê como resultado o esquema da mencionada base de dados relacional mínima.

Obter este procedimento de engenharia inversa foi o primeiro grande resultado de pesquisa do projeto GeneXus. Depois, construir a necessária Knowledge Base sobre ele foi um muito importante e bem-sucedido trabalho permanente.



MODELO EXTERNO E KNOWLEDGE BASE

A **Knowledge Base** tem inicialmente associado um conjunto de mecanismos de inferência e contém regras gerais **[5]** independentes de qualquer aplicação particular. Ao descrever a realidade do usuário objeto armazenam-se as descrições no **Modelo Externo.**

O sistema, automaticamente, captura todo o conhecimento contido no **Modelo Externo** e o sistematiza, acrescentando-o também à **Knowledge Base**. Adicionalmente, sobre o conhecimento anterior, o sistema infere logicamente um conjunto de resultados que ajudam a melhorar a eficiência das inferências posteriores.

GeneXus trabalha permanentemente sobre a Knowledge Base. Todo o conhecimento da Knowledge Base é equivalente ao contido no Modelo Externo (subconjunto dela), já que consiste no próprio Modelo Externo mais regras e mecanismos de inferência independentes desse Modelo Externo e um conjunto de outros elementos automaticamente inferidos a partir dele.

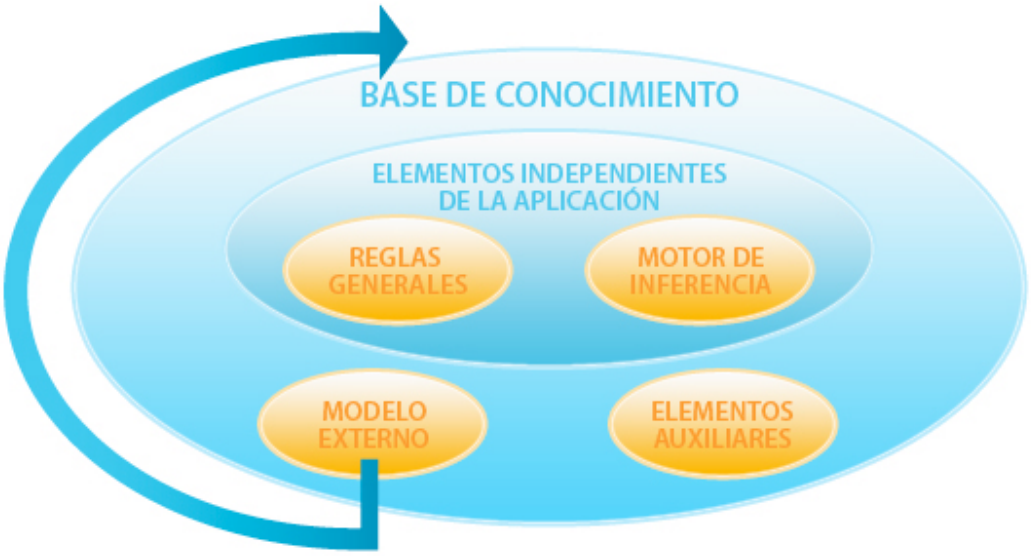
O desenvolvedor pode alterar o Modelo Externo, modificando objetos da realidade do usuário, e as modificações irão se propagar en forma automática a todos os elementos necessários: outros elementos da Knowledge Base, Base de dados e programas da aplicação. Da mesma maneira, o desenvolvedor não pode alterar diretamente nenhum elemento não pertencente ao Modelo Externo.

Este trabalho ocupa-se apenas do **Modelo Externo** (o **"quê"**) e se abstém de tratar a **Knowledge Base**, que o contém e o mantém, (e que forma parte do **"como"**: um sofisticado conjunto de ferramentas matemáticas e lógicas que embasam os processos de inferência e resultados intermédios utilizados para aumentar a eficiência dessas inferências). Não é necessário conhecer nada do "como" para fazer um ótimo uso do GeneXus.

O que é mais importante?: O "quê" ou o "como"? Nenhum funciona sem o outro, mas se o "quê" não for correto, tudo estará errado.

Do anterior pode-se inferir algo muito importante, que vai muito além da implementação atual do GeneXus: **TODO** o conhecimento está contido no Modelo Externo e, por isso, amanhã poderíamos suportar a **Knowledge Base** de uma maneira totalmente diferente e o conhecimento de nossos clientes continuaria sendo utilizável sem nenhum problema.

Em resumo: A Knowledge Base e os mecanismos de inferência associados constituem uma grande "máquina de inferência". Um bom exemplo disto é: dada uma visão de dados podemos inferir em forma totalmente automática o programa necessário para sua manipulação.



DIA"D" E PROTOTIPAÇÃO

A prototipação é um grande recurso e utilizando-o adequadamente evita-se uma exagerada dependência do teste final ou, ainda, da posta em andamento (o Dia "D", onde tudo ocorre e a Lei de Murphy é particularmente aplicada).

Pelas características do GeneXus, tudo pode-se provar a qualquer momento e, em particular, pode-se prototipar cada objeto no momento oportuno para fazê-lo. Esta possibilidade é sempre muito importante e é conseqüência direta da teoria do GeneXus e, em particular, de sua propagação automática das mudanças, característica relevante e exclusiva do GeneXus.

RESUMO DE CONCEITOS BÁSICOS DO GENEXUS

Tratamento automático do conhecimento. Baseado na existência de um modelo da realidade objeto: **Modelo Externo** com o qual um software de computador possa trabalhar automaticamente. Em particular, que permita:

Consistência: Modelo sempre consistente.

Ortogonalidade: os objetos que constituem o Modelo Externo são independentes entre si. Adicionar um novo, sua modificação ou eliminação não implicará a necessidade de modificar nenhum outro objeto.

Geração e manutenção automáticas da base de dados e dos programas.

Modelo Externo: O conhecimento essencial corresponde a um "Modelo Externo", que não pode conter nenhum elemento físico ou interno: arquivos, tabelas, entidades, relaciones entre entidades, índices ou qualquer elemento que possa ser deduzido automaticamente do mencionado "Modelo Externo".

Integração: GeneXus tem um conjunto de objetos próprios totalmente integrados por concepção mas a partir de sua versão Rocha permitirá outros níveis de integração: a de outros módulos ou produtos desenvolvidos pela Artech ou por terceiros.

TENDÊNCIAS DO GENEXUS

GeneXus é um produto com uma permanente evolução. As tendências desta evolução podem ser representadas sobre quatro dimensões: **COMPLETUDE, PRODUTIVIDADE, UNIVERSALIDADE** e **USABILIDADE**.

COMPLETUDE: Mede a percentagem de código do sistema do usuário que é gerado e mantido automaticamente pelo GeneXus.

Desde 1992 a completude alcançada pelo GeneXus é sempre de 100%, Este é um compromisso fundamental porque implica uma propagação automática das mudanças, que acarreta uma diminuição dramática dos custos de desenvolvimento e manutenção.

UNIVERSALIDADE: Mede a cobertura das diferentes plataformas importantes disponíveis no mercado. Hoje GeneXus suporta todas as plataformas “vivas”, entendendo-se por plataformas vivas aquelas que têm uma importante e crescente base instalada.

Desta maneira, GeneXus oferece aos clientes uma grande liberdade de eleição: se uma aplicação for desenvolvida com o GeneXus o cliente pode sempre transferi-la para outra plataforma suportada sem custos importantes.

PRODUTIVIDADE: Mede o aumento de produtividade do desenvolvimento com o GeneXus sobre o desenvolvimento com programação manual.

O objetivo de aumento de produtividade do GeneXus foi, durante muitos anos, de 500%, o que era suficiente.

Mas os sistemas de que precisam os clientes são a cada dia mais complexos porque existem novos dispositivos e novas necessidades; por cima de tudo, precisam de sistemas que preencham suas necessidades, por complexas que elas forem, e que, no entanto, permaneçam simples para os usuários que, em geral, não poderão ser treinados.

Adicionalmente, esses sistemas devem ser desenvolvidos cada vez em tempo menor: o “time to market” é cada vez mais crítico.

No ano 2004 modificaram-se esses objetivos passando a 1000% para a versão 9 e a 2000% para a versão Rocha.

USABILIDADE: Mede a facilidade de uso. E pretende fazê-lo com caráter geral: facilitar o uso aos usuários técnicos, mas também aumentar o alcance permitindo que cada vez mais usuários não técnicos possam utilizar GeneXus com proveito.

Só o fato de utilizar GeneXus acarreta um grande aumento da usabilidade: um desenvolvedor GeneXus pode desenvolver excelentes aplicações para uma determinada plataforma sem precisar de um conhecimento profundo dessa plataforma o que significa um nível mínimo importante de usabilidade.

Mas pretendemos ir muito além: pretendemos que muitos mais usuários, cada vez com menos pré-requisitos técnicos, possam beneficiar do uso do GeneXus.

A versão Rocha irá melhorar fortemente a usabilidade e essa tendência continuará nas próximas versões.

CÓMO CONTINUAMOS? ESTÁ NOSSO MODELO EXTERNO ESCRITO EM PEDRA?

Certamente não, os princípios são permanentes e a experiência mostra a validez do feito. Mas sempre devemos observar O GeneXus criticamente para ampliá-lo e assim poder descrever da maneira mais simples possível os novos casos que a realidade vai colocar .

BIBLIOGRAFIA E REFERÊNCIAS

[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] REGRAS:

Sem prejuízo das regras particulares que tem o modelo associado de uma determinada empresa, existem regras gerais, de validez permanente e independentes de qualquer aplicação. A título de exemplo, veremos o seguinte:

Necessidade da Consistência: Não pretendemos aqui tratar o tema detalhadamente, mas a principal fonte de regras, como em todas as ciências, é a consistência: supomos que a realidade é consistente e, como conseqüência, toda representação que dela façamos necessariamente deve sê-lo.

Partindo deste princípio, podemos inferir muitas regras de validez geral que enriquecerão nosso modelo.

Um caso particular, simples mas muito importante, é o da **integridade referencial**:

1. Dado X, atributo simples ou composto, que é chave de uma determinada tabela T1, se X aparece também em uma tabela T2, dada uma fila de T2, deve existir uma fila de T1 onde T1.X = T2.X
2. Dado X, atributo simples ou composto, que é chave de uma determinada tabela T1, e E, atributo simples ou composto tal que E = subtipo(X), se E aparece em uma tabela T2, dada uma fila de T2, deve existir uma fila de T1 onde T1.X = T2.Y
3. Dado E, atributo simples ou composto, que aparece em uma Tabela T2 tal que não existe uma tabela T1 onde E é chave nem X (tal que E = subtipo(_X)) é chave , E só pode aparecer na Tabela T2.

CONTEÚDO