Microsoft® BizTalk® Server 2010

BizTalk Server - Como funcionam os mapas

Os mapas, ou transformações, são um dos componentes mais comuns nos processos de integração. Funcionam como tradutores essenciais no desacoplamento entre os diferentes sistemas a interligar. Este artigo tem como objectivo explicar como os mapas são processados internamente pelo motor do produto à medida que exploramos o editor de mapas do BizTalk Server.

Sandro Pereira

setembro 2023
Versão 1.0

Índice

Introdução

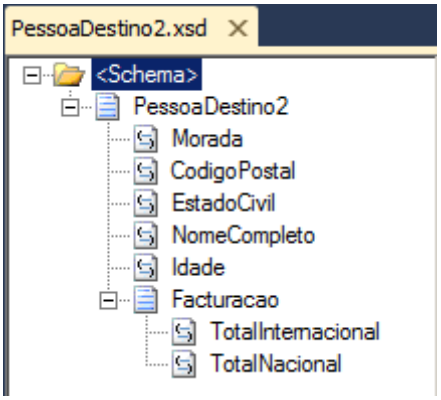
Os mapas, ou transformações, são um dos componentes mais comuns nos processos de integração. Funcionam como tradutores essenciais no desacoplamento entre os diferentes sistemas a interligar. Este artigo tem como objectivo explicar como os mapas são processados internamente pelo motor do produto à medida que exploramos o editor de mapas do BizTalk Server.

Este artigo tem como base o exemplo do artigo "BizTalk Server - Princípios básicos dos Mapas" onde é explicado em detalhe as funcionalidades básicas dos mapas e como podem ser implementadas. Pretende ser uma nota introdutória e destinada a quem está a dar os primeiros passos nesta tecnologia.

Conforme explicado no artigo anterior, quando estamos a efectuar uma transformação de mensagens são 5 as funcionalidades comuns que normalmente nos surgem:

- Mapeamento simples de um determinado valor (cópia directa)
- Concatenação de valores
- Selecções condicionadas
- Scripts customizados
- Adicionar novos dados

Tendo por base estas funcionalidades vamos explicar como o motor de mapas do BizTalk traduz e processa estas ligações internamente. Para melhor compreendermos o seu funcionamento efectuamos algumas alterações à estrutura do esquema (*schema*) do documento final: acrescentamos um elemento opcional ("EstadoCivil") e desorganizamos intencionalmente a estrutura do documento.



Modelo de processamento dos Mapas

Embora tradicionalmente a informação seja extraída da origem à medida que vai sendo processada, na realidade nos modelos baseado em XSLT o que acontece é exactamente o contrário: Cada elemento no destino provoca a procura pelo correspondente na origem. Vejamos um exemplo tradicional:

- A origem é percorrida de inicio ao fim do ficheiro;
- A informação é extraida da origem na ordem exacta que é encontrada;
- As regras de mapeamento são construidas à medida que a origem é percorrida.

O BizTalk também utiliza esta técnica nas conversões dos ficheiros texto (*Flat Files*) para formato XML (transformações de sintaxe, também explicado no artigo anterior), no entanto as transformações nos mapas utilizam uma abordagem diferente, como iremos verificar mais à frente neste artigo.

Um dos factores importantes quando utilizamos ferramentas de integração é também termos em atenção as tecnologias *standards* existentes, e foi isso o que os criadores do produto fizeram. O BizTalk trabalha internamente, quase exclusivamente, com documentos XML, sendo que fazia sentido utilizarem uma tecnologia *standard* para efectuarem este tipo de transformações, para isso o W3C definiu o XSLT (*Extensible Stylesheet Language Transformation*) como o formato padrão para representar as transformações entre documentos XML.

Desta forma todas as ligações e *functoids* que são visiveis graficamente na grelha de mapeamentos não são mais do que uma representação gráfica de um documento XSLT que permite transformar o documento de origem num determinado formato especificado pelo esquema de destino.

Podemos dizer que os mapas de BizTalk têm sempre o seu foco no documento final, fazendo assim sentido que as regras de transformação sejam processadas na sequência requerida para o criar. Quando o mapa é compilado, essas regras são traduzidas em queries XPATH e funções XSLT por forma a transformar a informação pretendida, ou seja, as regras de mapeamento são construidas a partir da estrutura de destino e não da origem como algumas ferramentas tradicionais.

Sendo assim, os mapas de BizTalk seguem o seguinte modelo:

- O motor de mapeamento do BizTalk percorre a estrutura de destino do inicio ao fim;
- As regras de mapeamento são construidas e executadas conforme os links são encontrados na estrutura de destino;
- A informação é extraida da origem quando um link é encontrado na estrutura de destino.

Nota: Podemos utilizar um XSLT criado por uma aplicação externa e incluí-lo no mapa através de XSLT custom script ou importando um ficheiro XSLT que efectua a transformação total do mapa (obviamente o editor gráfico do mapa de BizTalk não representará as ligações visualmente).

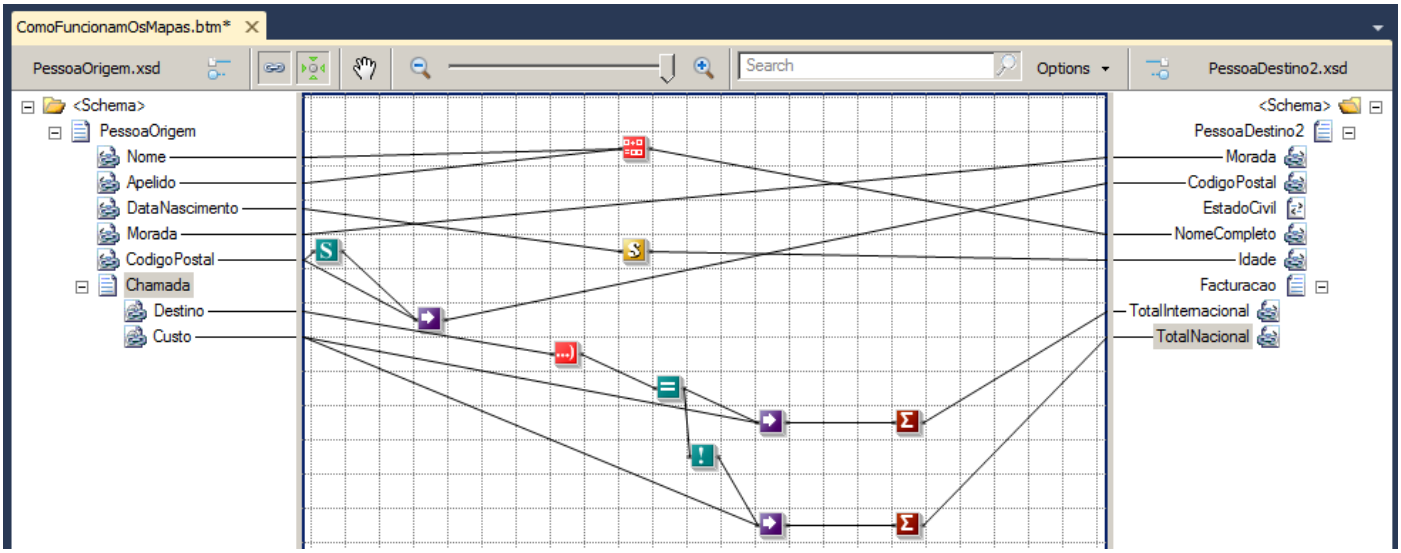
Desconstruindo um mapa

Neste artigo vamos utilizar as operações básicas de mapeamento descritas anteriormente, e analisar as decisões tomadas pelo "compilador" de mapas do *BizTalk Mapper Designer*. Basicamente neste mapeamento existem dois esquemas similares, no qual pretendemos mapear os elementos da origem no seu correcto destino e ao qual implementamos os seguintes desafios:

- Concatenar o Nome e Apelido por forma a obtermos o nome completo (concatenação de valores);
- Mapear a Morada no seu elemento de destino (mapeamento simples de um determinado valor)
- Transformar a data de nascimento em idade (scripts customizados);
- O Código Postal só deverá ser mapeado se tiver uma *string* valida, caso contrário não será mapeado (selecções condicionadas);
- O elemento Estado Civil é opcional e como não temos elementos para o mapear, o mesmo deverá ser ignorado (adicionar novos dados).
- Adicionalmente, iremos efectuar uma lógica de mapeamento mais avançado para calcular os totais de chamadas internacionais e nacionais utilizando ciclos, selecções condicionas e operações matemáticas.

Conforme podem verificar, intencionalmente trocamos a ordem dos elementos no esquema de origem, por forma a verificarmos com mais facilidade como os mapas de BizTalk funcionam.

Desta forma obtivemos o seguinte mapa final:



Nota: A ordem pelo qual efectuamos as ligações entre os elementos da origem para os elementos de destino, neste cenário, não tem qualquer importância para o compilador, no entanto o mesmo já não se pode dizer para as *functoids*. As *functoids* necessitam de determinados parâmetros de entrada que podem variar sendo a ordem de entrada importante.

A imagem anterior do mapa de BizTalk é na realidade a seguinte representação do documento XSLT:

- https://gist.github.com/1597897

Com base neste documento vamos seguir o comportamento do compilador. O que ele efectua é, traduzir as ligações existentes no mapa à medida que os encontra enquanto percorre o esquema de destino:

- O primeiro elemento encontrado é “Morada”, como tem uma ligação associada, a mesma é traduzido por expressão XPath (”Morada/text()”) que define o elemento a extrair da origem:

```
<Morada>
   <xsl:value-of select="Morada/text()" />
</Morada>
```

- O segundo elemento encontrado é o “CodigoPostal” que também ele tem uma ligação associada. Trata-se de uma selecção condicionada que será traduzida para uma condição XSLT (xsl:if):

```
<xsl:variable name="var:v1" select="userCSharp:LogicalIsString(string(CodigoPostal/text()))" />
<xsl:if test="string($var:v1)='true'">
  <xsl:variable name="var:v2"
        select="CodigoPostal/text()" />
  <CodigoPostal>
    <xsl:value-of select="$var:v2" />
  </CodigoPostal>
</xsl:if>
```

- O terceiro elemento é “EstadoCivil”, uma vez que não tem nenhuma ligação associada, o mesmo é ignorado no mapeamento.
- O quarto elemento processado é “NomeCompleto”, este elemento tem uma ligação associada, que corresponde ao valor $var:v3 que é construído a partir da concatenação dos vários *inputs* e a execução da função userCSharp:StringConcat que visualmente era o ***String Concatenate Functoid***:

```
<xsl:variable name="var:v3" select="userCSharp:StringConcat(string(Nome/text()) , &quot; &quot; ,
string(Apelido/text()))" />
<NomeCompleto>
 <xsl:value-of select="$var:v3" />
</NomeCompleto>
```

- O quinto elemento é o "Idade", uma ligação é encontrada o que significa que irá ser efectuado o mapeamento do script customizado que estava dentro do ***CSharp Inline Scripting Functoid***:

```
<xsl:variable name="var:v4" select="userCSharp:CalcularIdade(string(DataNascimento/text()))" />
<Idade>
 <xsl:value-of select="$var:v4" />
</Idade>
```

- Por fim é encontrado o nó (record) "Facturacao" com elementos: "TotalInternacional" e "TotalNacional". De realçar que apesar de não termos definido nenhum ciclo através da functoid "*Loop*" o compilador e suficientemente inteligente para perceber que estamos a tratar de um ciclo e traduzi-lo correctamente, no entanto irá criar para cada um dos elementos um ciclo próprio. Para uma melhor optimização seria necessário utilizarmos um script customizado.

```
<Facturacao>
  <xsl:variable name="var:v5"
       select="userCSharp:InitCumulativeSum(0)" />
  <xsl:for-each
       select="/s0:PessoaOrigem/Chamada">
    <xsl:variable name="var:v6" select="userCSharp:StringLeft(string(@Destino) ,
                                &quot;4&quot;)" />
    <xsl:variable name="var:v7"
                 select="userCSharp:LogicalEq(string($var:v6) , &quot;+351&quot;)" />
    <xsl:if test="string($var:v7)='true'">
      <xsl:variable name="var:v8" select="@Custo" />
      <xsl:variable name="var:v9"
             select="userCSharp:AddToCumulativeSum(0,string($var:v8),&quot;1000&quot;)" />
    </xsl:if>
  </xsl:for-each>
  <xsl:variable name="var:v10" select="userCSharp:GetCumulativeSum(0)" />
  <TotalInternacional>
    <xsl:value-of select="$var:v10" />
  </TotalInternacional>
  …
</Facturacao>
```

A sequência das ligações

"*A ordem com que as ligações são associadas no destino tem um grande impacto no resultado final.*"… Esta afirmação é verdadeira e ao mesmo tempo falsa!

Na realidade a ordem com que associamos as ligações (*Drag&Drop*) dos elementos de origem para diferentes elementos de destino é irrelevante, uma vez que o compilador, conforme explicado anteriormente, irá processar pela ordem correcta… Excepto se tivermos de associar diversas ligações para o mesmo elemento de destino ou *functoid*. Nestes dois últimos casos a ordem com que se efectua a associação é extremamente importante, podendo originar resultados inesperados.

Impacto da ordem das ligações nas functoids

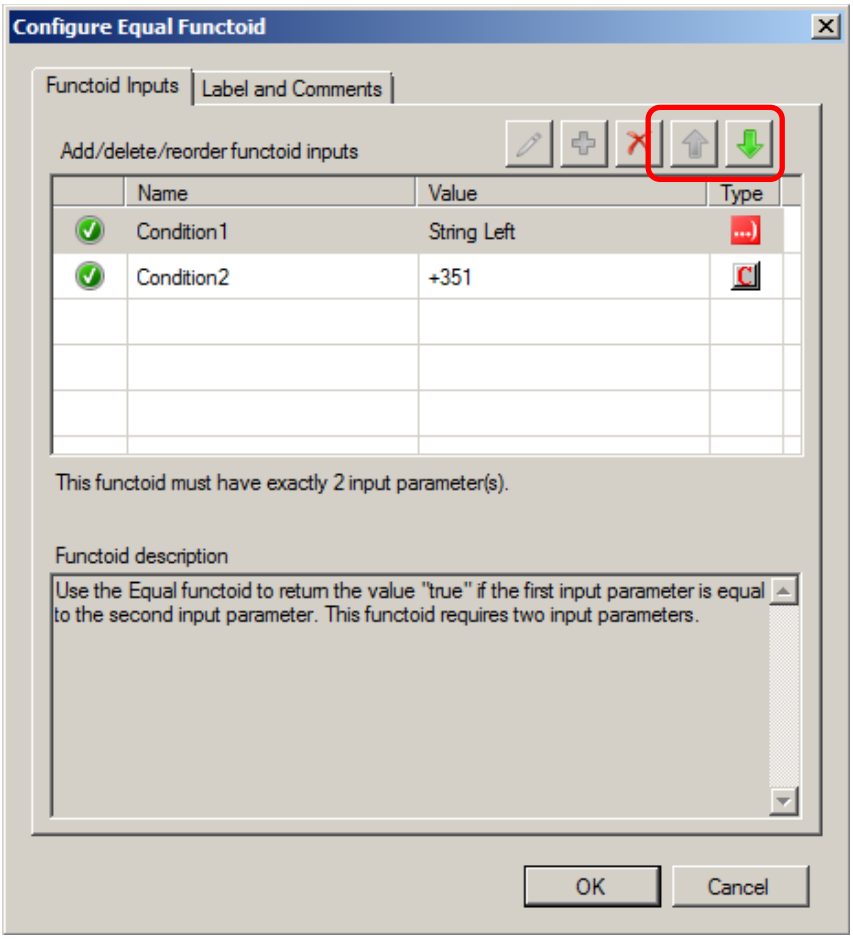
Uma grande parte das *functoids* existentes na *Toolbox* espera vários parâmetros de entrada, um exemplo prático é a *functoid* "*Value Mapping Functoid*".

Esta *functoid* retorna o valor do segundo parâmetro se o valor do primeiro for igual a "*True*", caso contrário não é retornado nada. Desta forma é necessário respeitar a ordem com que associamos as ligações:

- A primeira ligação a associar a esta *functoid* terá de enviar um valor booleano (*true/false*)
- O segundo será o valor que queremos retornar na *functoid*.

A troca na associação das ligações irá originar erros de mapeamento ou em resultados inesperados, conforme a *functoid* utilizada.

Reorganização das ligações (*links*) nas functoids é muito fácil, para isso basta abrir o detalhe (duplo clique) e usar os botões de ordenação.

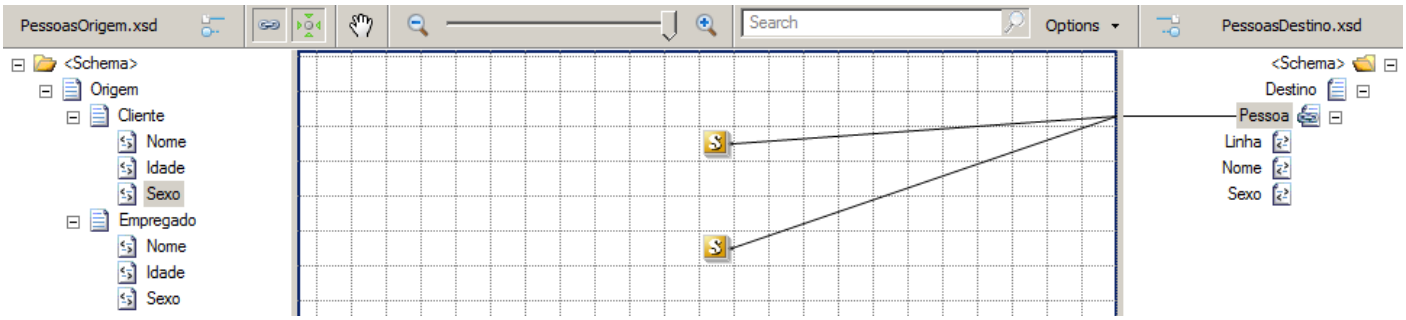


Impacto da ordem das ligações nos elementos no esquema de destino

Alterar a ordem da sequência na associação das ligações no mesmo elemento do esquema de destino poderá também ter impacto no resultado final pretendido.

Infelizmente, quando associamos diferentes ligações no mesmo elemento, não existe nenhuma forma ou local no editor gráfico onde podemos verificar a ordem da associação, à semelhança do que acontece com as *functoids*. A única forma de verificarmos a ordem nestes casos é inspeccionar o código XSLT gerado ou testando o mapa.

Um bom exemplo deste cenário é quando associamos duas *Scripting functoid* com diferentes *inline XSLT scripts* ao mesmo destino, uma vez mais a troca na associação poderá ter resultados inesperados.



Neste exemplo a 1ª "*Scripting functoid*" contêm o seguinte código XSLT:

```
<xsl:for-each select="Cliente">
  <Pessoa>
    <Nome><xsl:value-of select="Nome/text()" /></Nome>
    <Sexo><xsl:value-of select="Sexo/text()" /></Sexo>
  </Pessoa>
</xsl:for-each>
```

Este código efectua o mapeamento dos todos os elementos existentes no nó "Clientes" no esquema de origem, para os elementos no nó "Pessoa" no esquema de destino.

A segunda "*Scripting functoid*" contêm um código XSLT idêntico (https://gist.github.com/1598161) mas desta vez irá mapear todos os elementos existentes no nó "Empregado" no esquema de origem, para os elementos no nó "Pessoa" no esquema de destino.

O resultado expectável no documento final é aparecerem todos os clientes no nó "Pessoa" e de seguida todos os empregados. Se trocarmos a ordem da associação das ligações no esquema final, iremos verificar que o resultado também ele irá aparecer trocado.

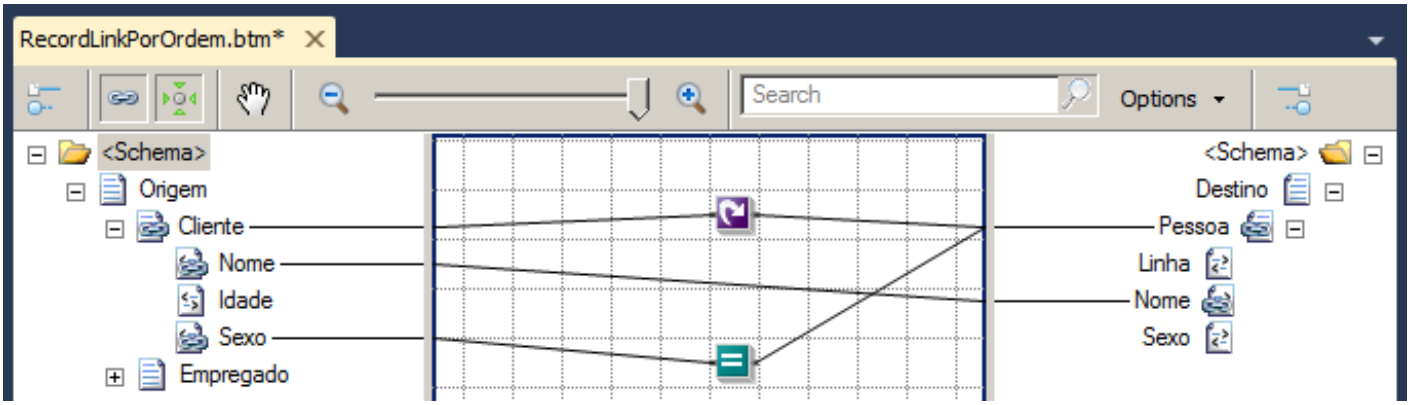
Podemos validar o resultado deste cenário nos seguintes *links*:

- Mensagem original:
  - https://gist.github.com/1598182
- Resultado expectado:
  - https://gist.github.com/1598172
- Resultado se trocarmos a ordem da associação:
  - https://gist.github.com/1598188

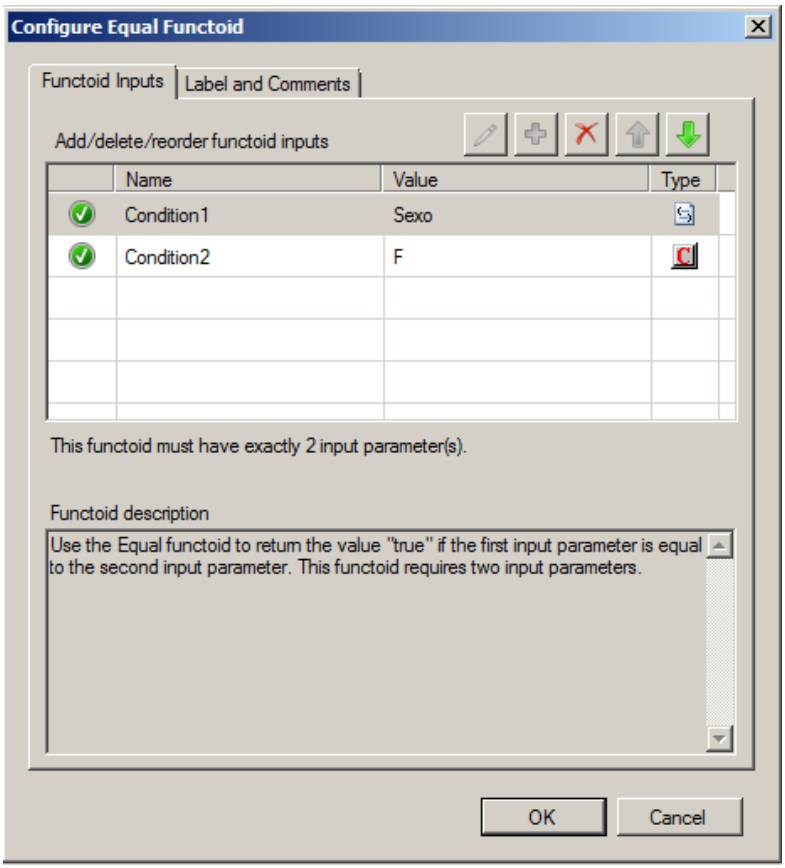
A excepção à regra da sequência das ligações

Em resumo, o motor de mapas processa as "regras" percorrendo o esquema de destino do início ao fim, processando as ligações pela ordem que os encontra e em caso de múltiplas ligações num determinado elemento ou *functoid*, as mesmas são processados pela ordem de associação. Isto significa que as ligações associadas aos nós pais são processadas antes das ligações associadas aos filhos.

Um exemplo deste cenário é o uso de condições no nó pai quando pretendemos condicionar o resultado final segundo uma determinada condição. Vamos então procurar todos os nomes dos clientes do sexo feminino. Para isso iremos criar um mapa com as seguintes configurações:



- Adicionar a "*Looping Functoid*", associando o nó de origem "Cliente" ao nó de destino "Pessoa"
- Adicionar uma "*Equal Functoid*" e associar o elemento de origem "Sexo" do nó "Cliente" à *functoid* e de seguida a *functoid* ao nó de destino "Pessoa"
- Editar a "*Equal Functoid*" e editar a segunda condição com o valor "F" para construirmos uma equação equivalente a Sexo="F":



- Ligar o elemento de origem "Nome" do nó "Cliente" ao elemento de destino "Nome" do nó Pessoa

A "*Equal Functoid*" vai retornar o valor "*True*" se o elemento "Sexo" for igual a "F", caso contrário será retornado o valor "*False*". O que origina que o nó "Pessoa" só é mapeado se o valor retornado for igual a "*True*", obtendo assim a condição que pretendíamos.

Se verificarmos o código gerado,

```
…
<xsl:template match="/s0:Origem">
   <ns0:Destino>
      <xsl:for-each select="Cliente">
         <xsl:variable name="var:v1" select="userCSharp:LogicalEq(string(Sexo/text()) , "F")" />
         <xsl:if test="$var:v1">
            <Pessoa>
               <Nome>
                  <xsl:value-of select="Nome/text()" />
               </Nome>
            </Pessoa>
         </xsl:if>
      </xsl:for-each>
   </ns0:Destino>
</xsl:template>
```

Iremos verificar que a primeira acção do mapa, após o ciclo que percorre os vários elementos do nó é: obter o valor gerado na "*Equal Functoid*", representada na variável "v1";

Sendo a segunda operação validar a condição (IF) do valor da primeira operação (v1), ou seja, vai testar se o valor de "v1" é "*True*". Se a condição for verdadeira o código dentro da condição é executado, caso contrário irá passar para o próximo elemento sem executar nada. Obtendo assim o output desejado:

```
<ns0:Destino xmlns:ns0="http://ComoFuncinamOsMapas.Schema2">
       <Pessoa>
              <Nome>Elsa Ligia</Nome>
       </Pessoa>
</ns0:Destino>
```

Excepção: Processar ligações fora de ordem

No entanto existe uma importante excepção a esta regra de sequência, especialmente quando utilizamos scripts customizados nos elementos recursivos.

Um bom exemplo deste cenário é a utilização de scripts de incremento de contadores. Podemos ilustrar este cenário, adicionando duas "*Scripting Functoids*" ao mapa:

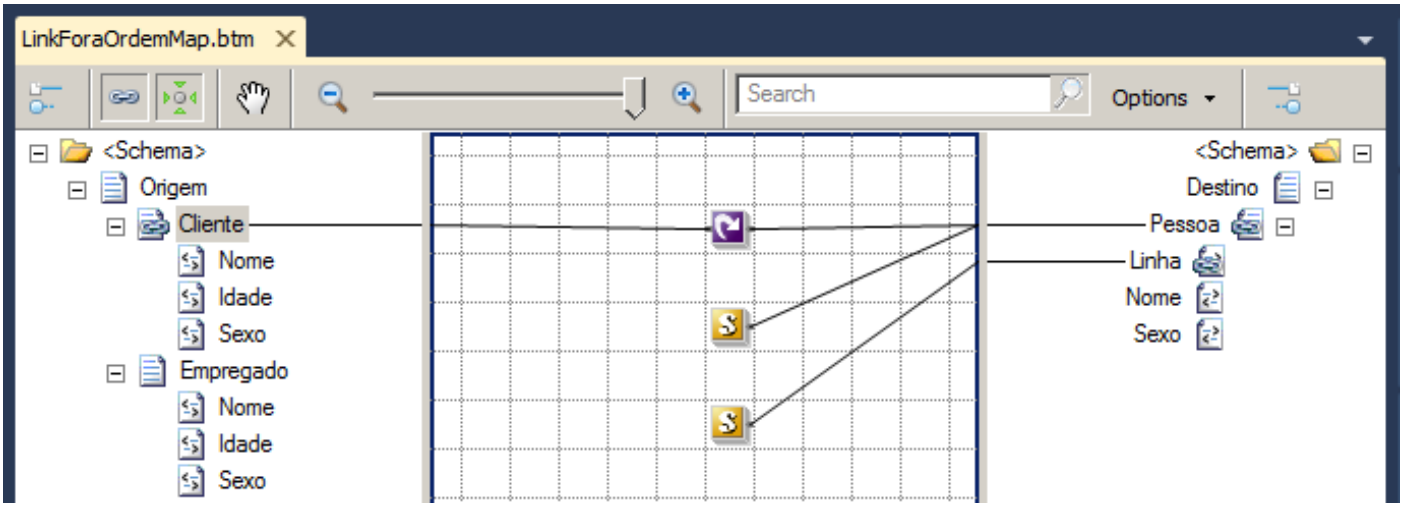
- A primeira contendo a inicialização e incremento do contador;

```
int contador = 0;
public void IncrementarContador()
{
   contador += 1;
}
```

- A segunda obtendo o valor do contador

```
public int RetornarContador()
{
     return contador;
}
```

Nota: Este exemplo estará associado a um ciclo (*Loop*), ou elemento recursivo.



Seria esperado que no primeiro ciclo o resultado do segundo script fosse o valor "1", no segundo ciclo obtivéssemos o valor "2" e assim sucessivamente. No entanto, se testarmos o mapa vamos verificar que a realidade é diferente:

```
<ns0:Destino xmlns:ns0="http://ComoFuncinamOsMapas.Schema2">
     <Pessoa><Linha>0</Linha></Pessoa>
     <Pessoa><Linha>1</Linha></Pessoa>
     <Pessoa><Linha>2</Linha></Pessoa>
</ns0:Destino>
```

Conforme podemos verificar no resultado em cima, a sequência com que as ligações são executadas é:

- Criar o nó (elemento) "Pessoa";
- Criar os elementos filhos e executar as ligações associadas aos mesmos:
  - Executar a função "RetornarContador" que irá retornar o valor "0" na primeira iteração.
- Executar as ligações associadas ao nó pai:
  - Executar a função "IncrementarContador".

Como podemos atestar verificando o código XSL produzido pelo mapa:

```
…
<xsl:template match="/s0:Origem">
  <ns0:Destino>
    <xsl:for-each select="Cliente">
      <Pessoa>
        <xsl:variable name="var:v1" select="userCSharp:RetornarContador()" />
        <Linha>
          <xsl:value-of select="$var:v1" />
        </Linha>
        <xsl:variable name="var:v2" select="userCSharp:IncrementarContador()" />
        <xsl:value-of select="$var:v2" />
      </Pessoa>
    </xsl:for-each>
  </ns0:Destino>
</xsl:template>
```

Claro que podemos alterar o código existente nas "*Scripting Functoids*" por forma a contornamos este comportamento e obter assim o resultado pretendido. No entanto este exemplo serve para alertar que, em alguns

cenários, especialmente no uso de scripts customizados nos nós recursivos, é necessário verificar e validar a sequência em que estes são executados.

Código Fonte

Todo o código fonte utilizado neste artigo encontra-se disponível no MSDN Code Gallery para download:

- Funcionalidades básicas dos mapas de BizTalk

Conclusão

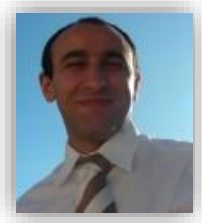
Com este artigo exploramos alguns mapeamentos comuns associados aos mapas, tentando desmontar as opções que a máquina tomou para cumprir com a intenção original do mapa visual.

Quando começarem a explorar o mundo dos mapas, existem duas questões que devem avaliar com maior atenção:

- **Qual a melhor forma para resolver um problema**: garantidamente existem várias abordagens para resolver um determinado problema. Muitas vezes decidir a melhor acaba por ser o mais difícil. Compilar e analisar o código gerado pode ser um bom princípio para começar a conhecer o impacto de determinadas opções.
- **Testes incrementais**: muita das vezes caímos na tentação de tentar resolver um determinado problema de transformação de início ao fim e só depois de finalizarmos é que vamos testar a solução. Deixar para o final pode tornar extremamente difícil detectar problemas em mapeamentos complexos. Limitar o âmbito dos testes deverá ser um processo contínuo e incremental durante a criação de mapas devendo ser efectuados logo que seja completado um significativo bloco de trabalho.

Espero que este tipo de *Hacking* ajude a entender o comportamento e as técnicas de *debugging* elementares para este tipo de problemas. Como todas as áreas em constante maturação, acompanhar os diferentes autores *online* é provavelmente a forma mais natural para ir descobrindo novos padrões e respectivas soluções.

Autor



Escrito por Sandro Pereira [MVP & MCTS BizTalk Server 2010]

Actualmente *Senior Software Developer* na empresa DevScope (www.devscope.net). É *Microsoft Most Valuable Professional* (MVP) em Microsoft BizTalk desde 2011 (https://mvp.support.microsoft.com/profile/Sandro.Pereira). O seu principal foco de interesse são as tecnologias e plataformas de Integração (EAI): BizTalk e SOAP / XML / XSLT e Net, que utiliza desde 2002.

É um participante bastante activo nos fóruns da Microsoft (*MSDN BizTalk Server Forums*), contribuidor no *MSDN Code Gallery* e na *Microsoft TechNet Wiki*, autor do **Blog**: http://sandroaspbiztalkblog.wordpress.com/, membro da comunidade BizTalk Brasil: http://www.biztalkbrasil.com.br/ e da comunidade BizTalk Administrators: http://www.biztalkadminsblogging.com.

Podes contacta-lo em: sandro-pereira@live.com.pt (Twitter: @sandro_asp).

Sandro Pereira | DevScope | MVP & MCTS BizTalk Server 2010