4
Prova de conceito: sistema para requisição de material

Este capítulo apresenta o projeto detalhado de um protótipo de um sistema para requisição de material baseado em mediação flexível de Web services, como ilustração da arquitetura apresentada no capítulo anterior.

A fim de reduzirem seus custos, as empresas realizam contratos de fornecimento com seus parceiros, comprometendo-se a faturar por volumes mensais em troca de preços diferenciados. As requisições de material geradas pelos vários solicitantes são processadas pelo setor que administra o contrato, e encaminhadas aos respectivos fornecedores. Com o aumento do número de parceiros, surge a necessidade de automação do processo, a fim de não criar gargalos na cadeia de suprimentos. A solução é integrar o sistema de *e-Commerce* de cada um ao sistema de requisição de material corporativo, que deve ser único a fim de centralizar os pedidos.

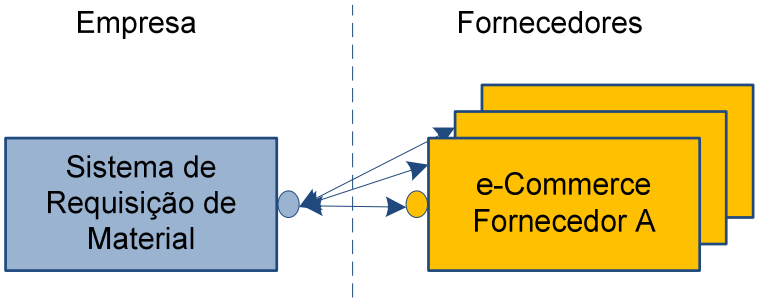

Figura 7 – Sistema de requisição de material corporativo

No sistema proposto, cada fornecedor possui um Web service com as funções de listar materiais, efetuar e acompanhar pedidos. Os Web services são cadastrados em um único ponto, onde os solicitantes acessam os catálogos de produtos, efetuam e acompanham suas requisições de maneira centralizada. Uma requisição pode conter um ou mais pedidos, em função dos fornecedores envolvidos. Cada material solicitado possui um prazo de fornecimento e, para diminuir custos com transporte, o pedido de uma requisição só é despachado para o solicitante quando o último item estiver disponível para envio.

Conforme observamos no capítulo anterior, devido à arquitetura adotada o sistema de requisição de material corporativo é visto pelos solicitantes como um fornecedor único virtual.

4.1. Arquitetura Geral

A fim de efetuarmos os experimentos que serão apresentados no próximo capítulo, implementamos cada um dos módulos do sistema:

Módulo de fornecedor: contém o catálogo e o banco de dados de pedidos e itens de um fornecedor. Há um destes módulos para cada fornecedor parceiro. Suas funcionalidades estão expostas em um Web service. Possui uma interface que permite o acompanhamento dos pedidos pelos fornecedores.

Mediador: consome os Web services dos fornecedores e realiza a composição técnica. Atua como um roteador de mensagens.

Módulo dos solicitantes: consome os Web services dos fornecedores por meio do mediador. Possui uma interface Web com os usuários que permite a composição de catálogos, montagem, envio e acompanhamento de requisições.

Módulo de materiais: contém o repositório com os cadastros de solicitantes e fornecedores, bem como dados transacionais e composições. Neste módulo são efetuadas as composições funcionais. Destina-se a simular algumas funções do módulo de materiais de um ERP corporativo.

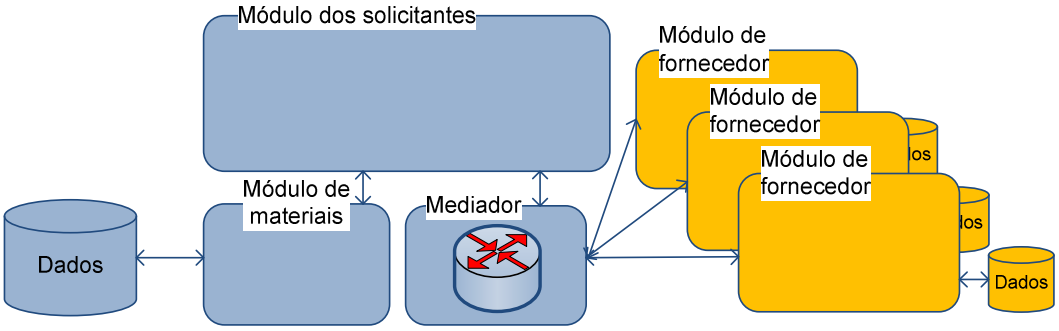

Figura 8 – Arquitetura geral do sistema

Detalhamos o funcionamento destes módulos a seguir.

4.1.1. Módulo de fornecedor

A fim de experimentarmos a composição de serviços, definimos este módulo de *e-Commerce* destinado aos fornecedores. Suas funções de envio de

catálogo, criação, consulta e cancelamento de pedidos são expostas por meio de um Web service.

Internamente, os pedidos são processados automaticamente, de acordo com o prazo estabelecido no catálogo. Para fins de simulação, os prazos dos itens são definidos em segundos. Conforme os requisitos do sistema, um pedido só é enviado quando todos os seus itens atingem o prazo estabelecido. Cada fornecedor possui seu módulo isolado dos demais, com seu próprio catálogo de materiais e banco de dados de pedidos.

Além do Web service, há um console de acompanhamento de pedidos para cada fornecedor.

Replicamos este módulo 3 vezes em um servidor de aplicações Web, simulando três fornecedores distintos, mas homogêneos, a serem acessados pelo mediador

4.1.2. Mediador

O mediador é baseado em uma plataforma de EAI com suporte a WS-BPEL. Possui as seguintes funcionalidades:

- **Interação com os fornecedores**: no repositório do mediador estão os WSDs dos fornecedores parceiros. O mediador atua como um fornecedor virtual em relação ao módulo de requisição, uma vez que expõe Web services com funções análogas aos fornecedores.
- **Composição técnica:** processa os atributos técnicos definidos para a composição – parceiros envolvidos, número de respostas desejadas e disponibilidade dos Web services dos fornecedores.
- **Agregação de catálogos:** as respostas dos fornecedores são agrupadas e enviadas para a camada de aplicação como uma única mensagem.

4.1.3. Módulo dos solicitantes

Compreende o controlador e a interface Web com os usuários. Possui as seguintes funcionalidades:

- **Interface dos solicitantes**: os solicitantes podem gerar e acompanhar suas requisições, de acordo com a composição selecionada.

- **Interface de administração**: o administrador pode gerar e disponibilizar novas composições para os usuários, conforme as políticas de compras da empresa.
- **Controlador**: quando o usuário encomenda sua requisição, esta camada verifica os fornecedores envolvidos por meio do módulo de materiais e dispara os pedidos correspondentes, por meio do mediador. Os fornecedores, por sua vez, retornam o status inicial de cada item, que pode ser 'Pendente' ou 'Cancelado', dependendo da disponibilidade dos itens. Decorridos os prazos de cada item, estes ficam aguardando os demais até que o pedido esteja completo, quando então o pedido passa ao status 'Enviado'.

Além das funcionalidades básicas, o controlador presente no módulo dos solicitantes realiza duas operações importantes no processo de composição: a orquestração de pedidos e o acompanhamento das requisições.

Quando uma requisição é montada, seus itens podem pertencer a mais de um fornecedor. Ao pressionar o botão "Encomendar", o módulo dos solicitantes determina os fornecedores envolvidos e cria pedidos em cada um deles por meio do mediador, informando os respectivos itens. O módulo de materiais registra a requisição e o código dos pedidos gerados.

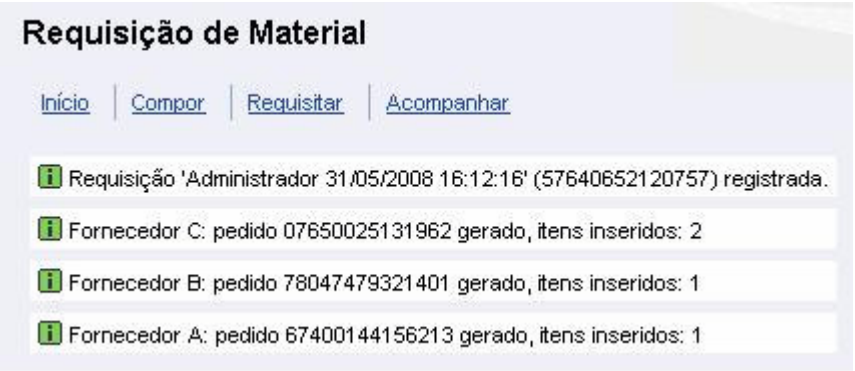

Figura 9 – Orquestração de pedidos no módulo dos solicitantes

Na fase de acompanhamento, as requisições do solicitante são mostradas em uma lista. Quando este seleciona uma requisição, o sistema determina os fornecedores envolvidos e consulta o status de cada pedido no fornecedor correspondente, por meio do mediador. Quando um pedido é selecionado, uma nova mensagem é gerada e enviada para o fornecedor, a fim de obter os status de cada item.

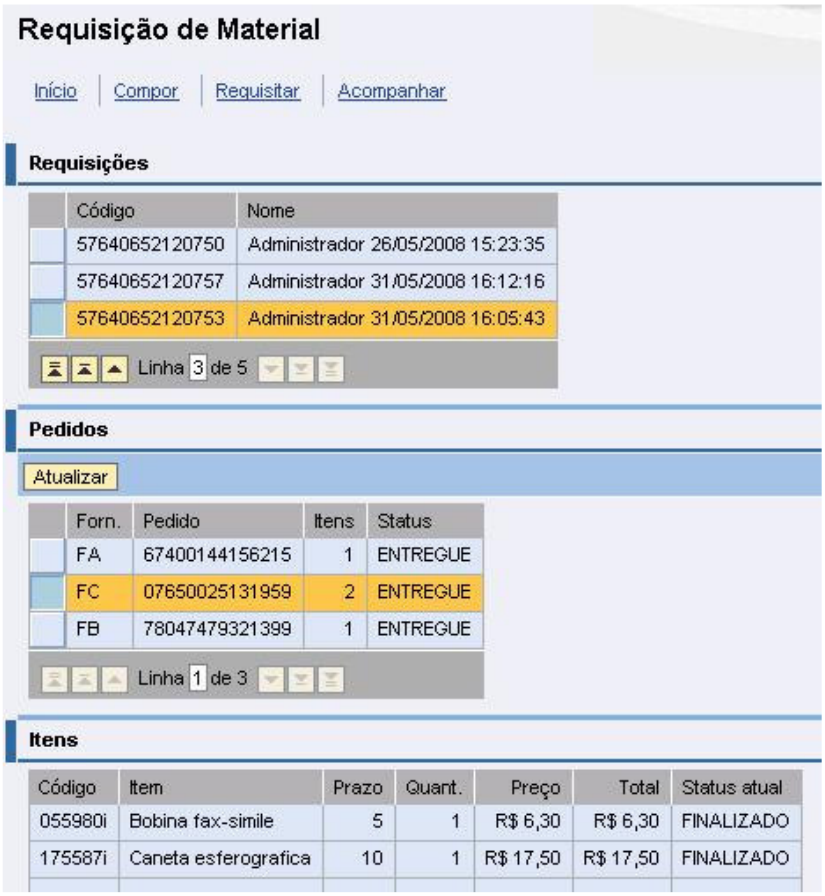

Figura 10 – Acompanhamento de requisições no módulo dos solicitantes

4.1.4. Módulo de materiais

Realiza a seleção dos fornecedores de acordo com os critérios indicados na composição selecionada, segundo a lógica de composição funcional que apresentamos no capítulo anterior. A fim de desempenhar suas funções, o módulo de materiais possui um repositório com os seguintes dados:

- Registros de fornecedores
  - Conceito
  - Localização
  - Certificações
- Registros de solicitantes
  - Dados de *logon*
  - Localização
- Registros de requisições
  - Solicitante
  - Data

- Pedidos
- Registros de composições
  - Atributos obrigatórios (interseção)
  - Atributos opcionais (união)
  - Mecanismo
  - Quantidade de catálogos

4.2. Descrição do sistema

Iniciamos nosso projeto de implementação definindo o módulo de fornecedor e o módulo dos solicitantes. Nosso objetivo inicial foi experimentar o sistema de requisição ainda sem realizar as composições, para então expandí-lo incluindo a arquitetura de mediação flexível apresentada neste trabalho, que contém um mediador e um repositório.

4.3. Casos de uso

O sistema de requisição de material possui *interface* Web. Seus atores são o solicitante corporativo e o fornecedor de materiais. O diagrama abaixo apresenta as funcionalidades básicas do sistema, sem levar em consideração a composição de fornecedores:

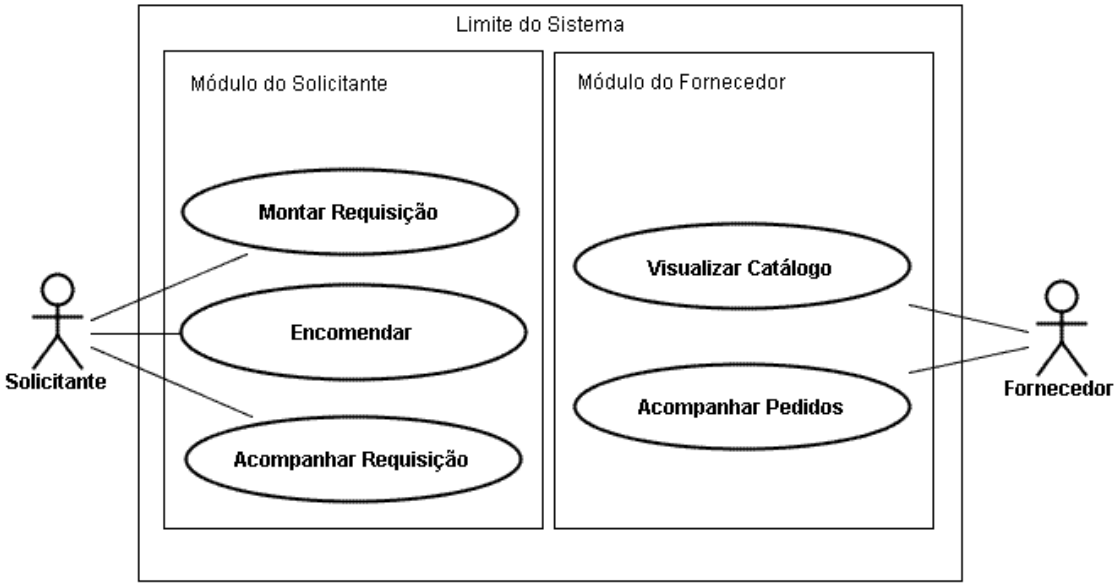

Figura 11 - Diagrama de casos de uso

Especificamos cada um dos casos de uso do diagrama:

Montar Requisição

Ator primário: Solicitante

Pré-condições: Nenhuma

Fluxo normal:

1. Usuário abre a página de Requisições
2. Sistema carrega catálogo
3. Adiciona itens do catálogo, informando as respectivas quantidades
4. Remove os itens indesejados ou reinicia removendo todos

Fluxo alternativo: Tentativa de remover itens sem antes selecioná-los

1. Sistema exibe mensagem de erro

Encomendar

Ator primário: Solicitante

Pré-condições: Há itens na nova requisição, nome da requisição definido

Fluxo normal:

1. Usuário clica no botão encomendar
2. Sistema solicita a geração de um pedido, e adiciona os itens da requisição a este pedido

Fluxo alternativo: Tentativa de encomendar sem antes adicionar itens à requisição

1. Sistema exibe mensagem de erro

Acompanhar Requisição

Ator primário: Solicitante

Pré-condições: Há requisições feitas

Fluxo normal:

1. Usuário abre página de acompanhamento
2. Sistema carrega as requisições feitas pelo usuário
3. Usuário seleciona uma requisição
4. Sistema carrega itens desta requisição

Fluxo alternativo: Não há requisições anteriores

1. Sistema exibe mensagem de erro

Visualizar catálogo

Ator primário: Fornecedor

Pré-condições: nenhuma

Fluxo normal:

1. Usuário abre página do catálogo
2. Sistema carrega os itens do catálogo e os apresenta

Fluxo alternativo: Não há itens no catálogo

1. Sistema exibe mensagem de erro

Acompanhar pedidos

Ator primário: Fornecedor

Pré-condições: nenhuma

Fluxo normal:

1. Usuário abre página de pedidos
2. Sistema carrega os pedidos e os apresenta
3. Usuário clica em um pedido
4. Sistema carrega os itens do pedido

Fluxo alternativo: Não há pedidos

1. Sistema exibe mensagem de erro

Fluxo alternativo 2: Não itens no pedido

1. Sistema exibe mensagem de erro

Para maior clareza do projeto inicial, provemos os diagramas de seqüência para os principais casos de uso:

Gerar uma requisição

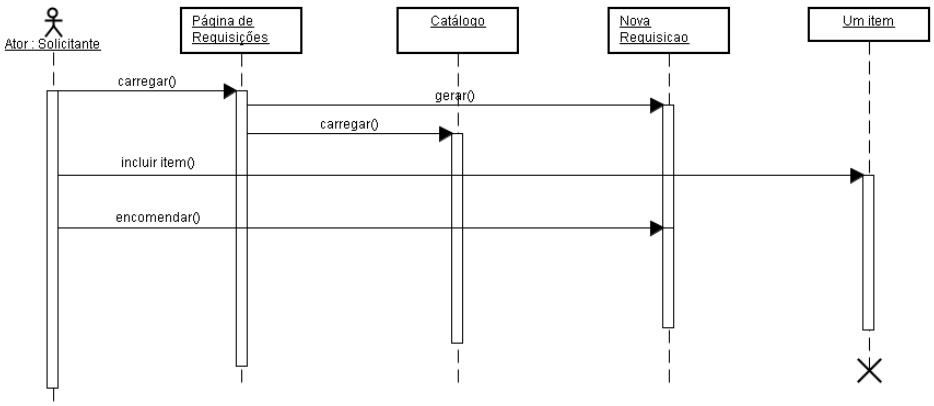

Figura 12 – Diagrama de seqüência: gerar uma requisição

Acompanhar uma requisição

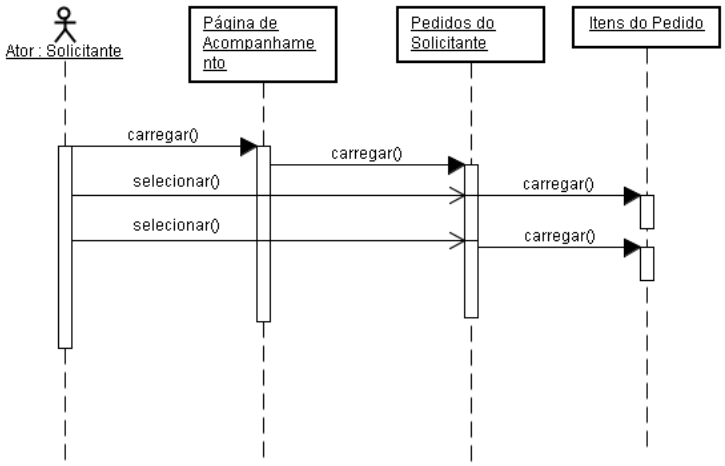

Figura 13 – Diagrama de seqüência: acompanhar uma requisição

Apresentamos a seguir o diagrama de classes para implementar as funções previstas nestes casos de uso:

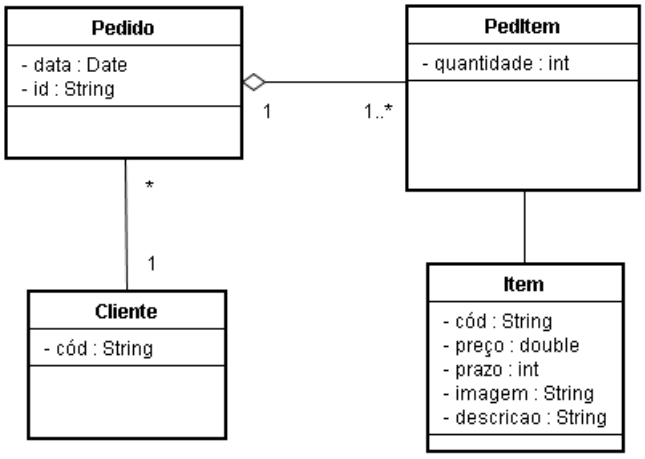

Figura 14 – Diagrama de classes do sistema básico de requisição

O modelo de classes apresentado define as relações entre os objetos do sistema: Pedido, Cliente e Item. Este último, quando está incluído em um Pedido, deriva um atributo quantidade, que indica quantas instâncias daquele item estão incluídas no pedido. Estas classes foram definidas como *entity beans* e utilizam o mecanismo *Container Managed Persistence,* para abstração da fonte de dados conforme o padrão DAO (*Data Access Objects*), apresentado mais adiante. Com o uso da tecnologia J2EE, construímos outras classes a fim de implementar a lógica de negócios e atender aos requisitos de abstração do projeto. Agrupamos estas classes em pacotes:

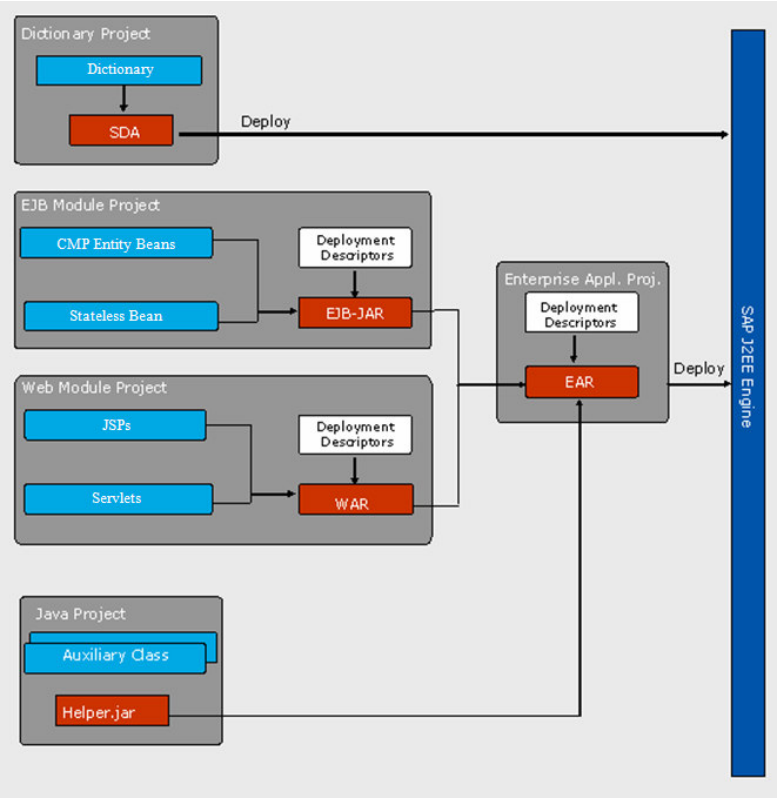

Figura 15 – Diagrama de pacotes J2EE

4.4. Padrões de Projeto

No desenvolvimento do sistema procuramos adotar as soluções de arquitetura indicadas nos padrões de projeto do *Sun Java Center*[1]. Apresentamos os padrões utilizados a seguir:

4.4.1. *Front Controller*

Objetivo: ter um ponto central de acesso para processamento de requisições.

[1] http://java.sun.com/developer/technicalArticles/J2EE/patterns/

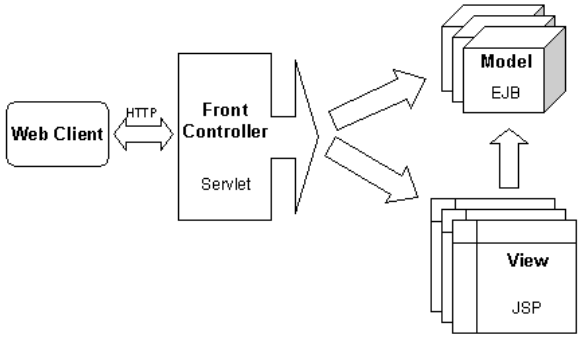

Figura 16 – Padrão *front controller*

Adotamos este padrão por meio do modelo *Model-View-Controller*, conforme ilustrado acima. Neste modelo a lógica da aplicação fica no controlador, enquanto as camadas do modelo e visão tratam dos dados e da interface com o usuário, respectivamente. O diagrama abaixo corresponde às camadas do módulo de fornecedor:

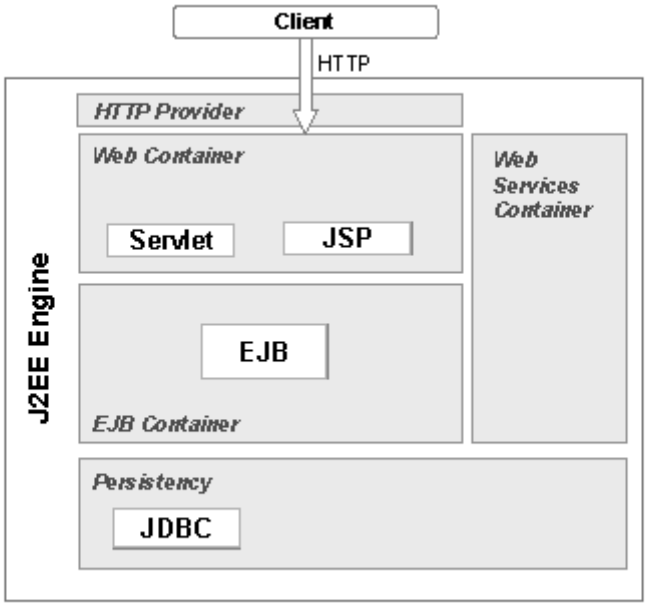

Figura 17 – Uso de MVC no módulo de fornecedor

Observar que neste módulo estão definidos *EJBs, JSPs e servlets,* que correspondem ao Modelo, Visão e Controlador do padrão, respectivamente.

Temos a mesma estrutura no módulo de solicitante:

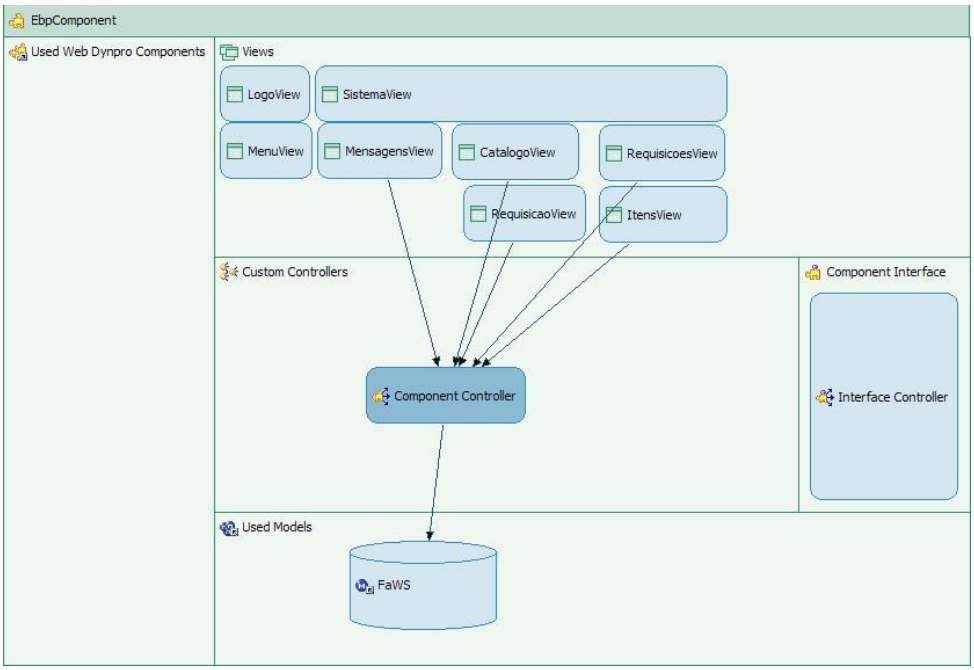

Figura 18 – Uso de MVC no módulo dos solicitantes

Neste módulo o uso do padrão é ainda mais evidente, pois a arquitetura *Web Dynpro* é baseada no padrão MVC. As visões correspondem às telas, o controlador contém a lógica da aplicação e o modelo é derivado a partir da WSD do provedor. Por projeto, a SAP limitou as funções de cada camada, de modo que o padrão fosse seguido estritamente.

4.4.2. *View Helper*

Objetivo: separar código e responsabilidades de formatação da *interface* do usuário do processamento de dados necessários à construção da *view*.

A aplicação desta técnica pode ser observada na tela de consulta aos pedidos/itens do módulo do fornecedor. A passagem de dados do *servlet* para a página JSP é feita por meio de um *array* de *strings* pré-formatadas, eliminando a necessidade de acoplamento das células da tabela com a classe correspondente.

4.4.3. *DAO (Data Access Object)*

Objetivo: Abstrair e encapsular todo o acesso à fonte de dados. O DAO gerencia a conexão com a fonte de dados para obter e armazenar os dados.

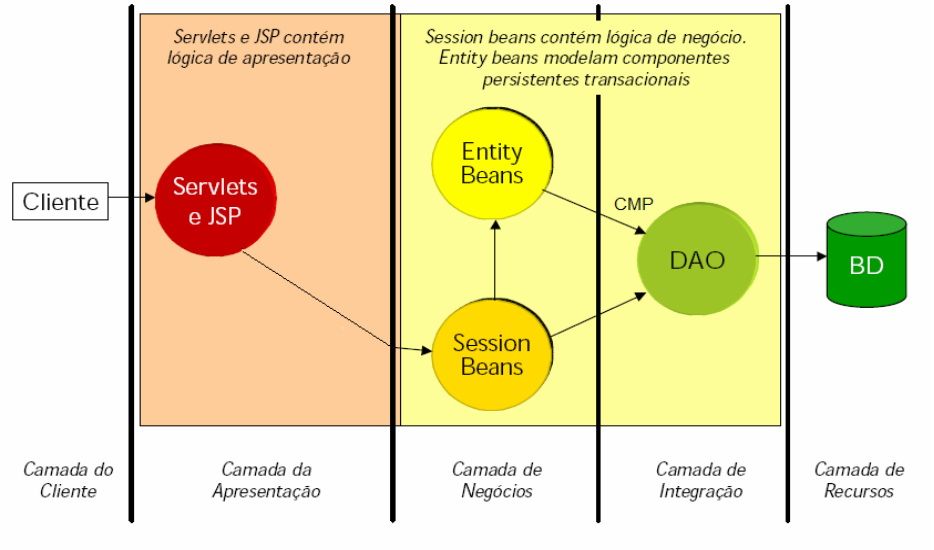

Figura 19 – Abstração por meio de *Data Acess Object*

As tabelas de Pedidos e Itens do fornecedor foram abstraídas por meio de *entity beans* com *Container-Managed Persistence* (CMP). Os *entity beans* são objetos de negócio armazenados de forma persistente, tipicamente através de banco de dados relacional. Os objetos de negócio são entidades do domínio do problema, tais como clientes, pedidos e itens. Cada instância de um *entity bean* corresponde a uma registro de uma tabela, e seus atributos são as colunas desta tabela. Com o uso de CMP, a persistência é tratada pelo *container*, que gera as instruções SQL adequadas ao banco de dados automaticamente. Desta forma, o tipo de banco de dados utilizado passou a ser apenas um parâmetro de configuração do sistema.

4.5. Implementação do protótipo

Assumimos as seguintes premissas de desenvolvimento:

1. Toda comunicação remota feita por meio de Web services, utilizando o protocolo HTTP.
2. O desenvolvimento adota Java e BPEL em plataforma SAP, de acordo com os requisitos do patrocinador.
3. A arquitetura proposta considera a interação com o módulo de materiais do ERP corporativo. Esta interação é simulada por meio de um módulo J2EE com o mesmo nome.
4. O módulo dos fornecedores é homogêneo. A heterogeneidade de dados e aplicações não é tratada pelos *Web services.* Cada fornecedor

que desejar participar do sistema deve compatibilizar seu programa ou utilizar o módulo fornecido.

O sistema foi implementado e testado no ambiente *Java(TM) 2 SDK build 1.4.2.09*.

O versionamento foi feito a partir de arquivos ZIP com *snapshots* do *workspace* do Eclipse a cada versão. O critério de novas versões foi determinado por evoluções representativas do código, por conclusão de codificação de um determinado requisito ou por conclusão de resultados de testes.

Além do Java, utilizamos as seguintes tecnologias:

- PC com processador AMD *Athlon64x2* 6000 3 GHz com 2GB RAM e 400 GB de HD.
- Sistema operacional: *Windows XP Professional* SP2
- Navegador Web*: Internet Explorer* 6, *Mozilla Firefox* 2.0
- Servidor de aplicações Web: SAP *Netweaver 2004s* SP12
- Mediador: SAP *XI* 7.0
- Bancos de dados: *MaxDB* 7.6, *Oracle* 10.2
- IDE: SAP *Netweaver Developer Studio* versão 7.09 (*Eclipse* 2.12 + *plug-ins* SAP)
- Diagramas UML: *Jude Community* 5.0.2
- Testes: *JUnitEE* 1.11, *AppPerfect DevTest4J* 9.2.0, HP *QuickTest Pro* 9.2
- Ferramentas XML: Altova *XML Spy 2008*, *SOAPUI* 2.0.2.

4.5.1. Módulo do fornecedor

A implementação iniciou com a criação de um projeto de dicionário, a fim de implantar as tabelas de pedidos e itens de pedidos no banco de dados:

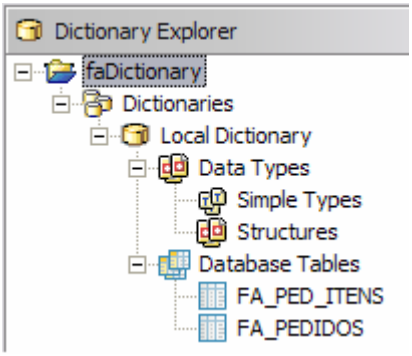

Figura 20 – Dicionário J2EE - módulo de fornecedor

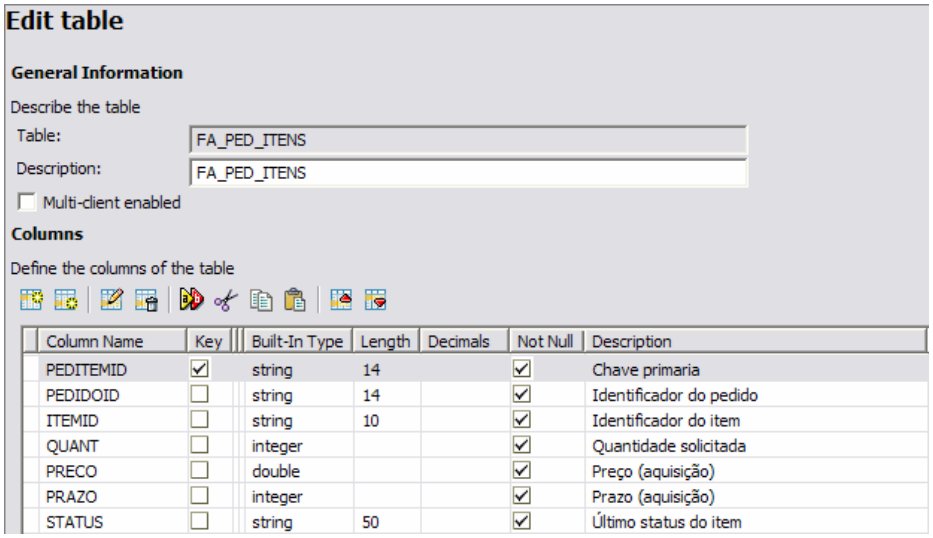

Figura 21 – Definição da tabela de itens de pedido

Uma vez definidos os atributos e chaves das tabelas, o arquivo SDA gerado foi implantado (*deployed*) no servidor. Esta implantação criou as tabelas no banco de dados *MaxDB*.

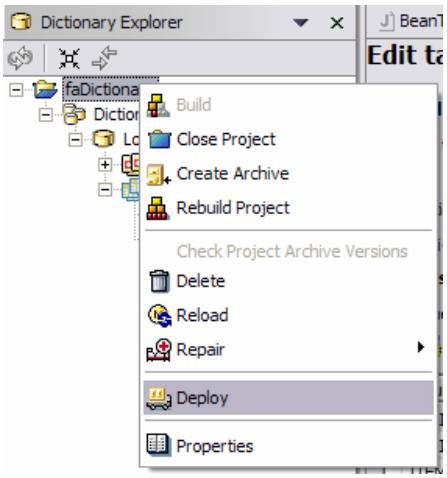

Figura 22 – *Deploy* do SDA do módulo de fornecedor

Em seguida construímos as classes de apoio (*Helper classes*) com as constantes do sistema, variáveis públicas, tratamento de exceções e as classes com os modelos de dados:

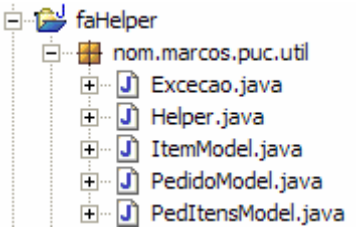

Figura 23 – *Helper Classes* do módulo de fornecedor

O *Eclipse* dispõe de um recurso de geração automática dos métodos get e set de cada classe, que facilita a codificação, além de torná-la menos sujeita a erros:

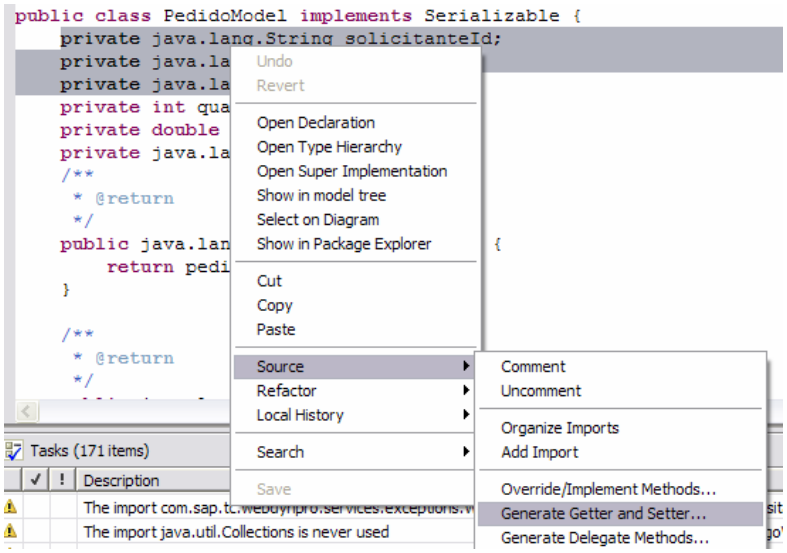

Figura 24 – Geração automática de código no Eclipse

Programamos também uma rotina para ler o catálogo de itens a partir de um arquivo-texto delimitado por tabulações. Definimos preços variados para os itens e prazos crescentes, como se vê na penúltima coluna. Desta forma pudemos validar o mecanismo de retenção de itens do pedido até que o último seja finalizado:

Código	Produto	Descrição	Preço	Prazo	Foto
055980i	Bobina fax-simile	215x30m termocopy VCP PT 1 UN	6.30	5	5055980i.gif
175587i	Caneta esferográfica	Cristal azul Bic CX 50 UN	17.5	10	175587I.GIF
229606i	Cartucho toner HP	Twin pack (2xq2612a) preto q2612at HP CX 1 UN	353	15	229606i.jpg
268306i	Etiqueta autoadesiva	Mod. op1342 Pimaco EN 210 UN	4.15	20	268306I.GIF
741021i	Tubo telescopico	Preto TS604 Rhamos & Brito PT 1 UN	29.99	100	741021i.jpg
783634i	Dvd c/100un -r	4.7gb 8x pino Elgin-	109	10000	783634i.jpg

```
canon PT 1 UN
```

Tabela 2 – Catálogo de itens de fornecedor

O próximo passo foi criar a página JSP para apresentar este catálogo no navegador. Para isso criamos um módulo Web (faWeb) e nele implementamos a página faCatalogo.jsp, onde inserimos as imagens dos produtos:

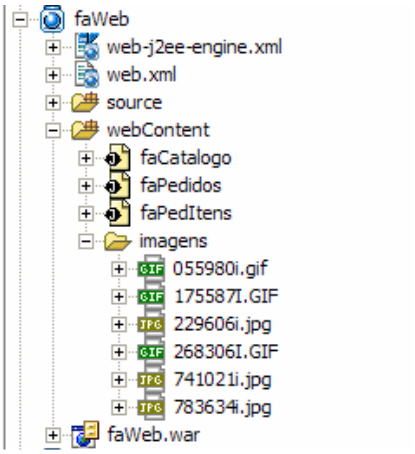

Figura 25 – Módulo de fornecedores – parte Web

Para realizar os testes foi necessária a criação de um módulo EAR (faEar). A partir deste ponto passamos a implantar as versões de nossa aplicação no WebAS.

Em seguida criamos o módulo EJB e passamos a implementar as classes com acesso ao banco de dados – *Entity Beans, Container Managed Persistence*. Com o auxílio de um *wizard*, o *Eclipse* gerou o automaticamente arquivo persistent.xml, que relaciona os atributos dos *beans* com as colunas das tabelas do banco de dados:

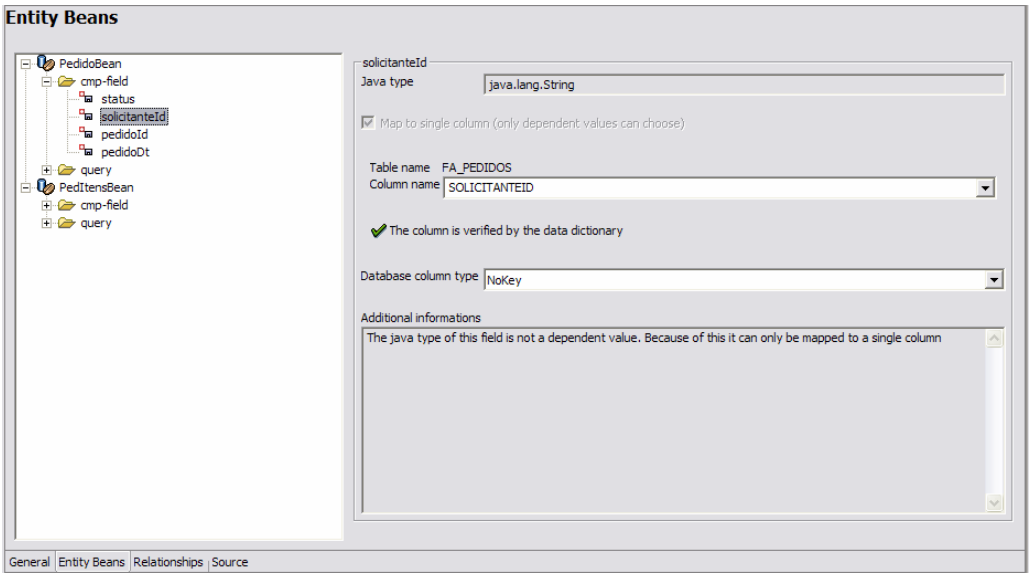

Figura 26 – Relacionamento dos *beans* com as tabelas

Para implementar a lógica de negócios e os métodos que serão expostos por meio de Web services criamos o *session bean* ProcessaPedido:

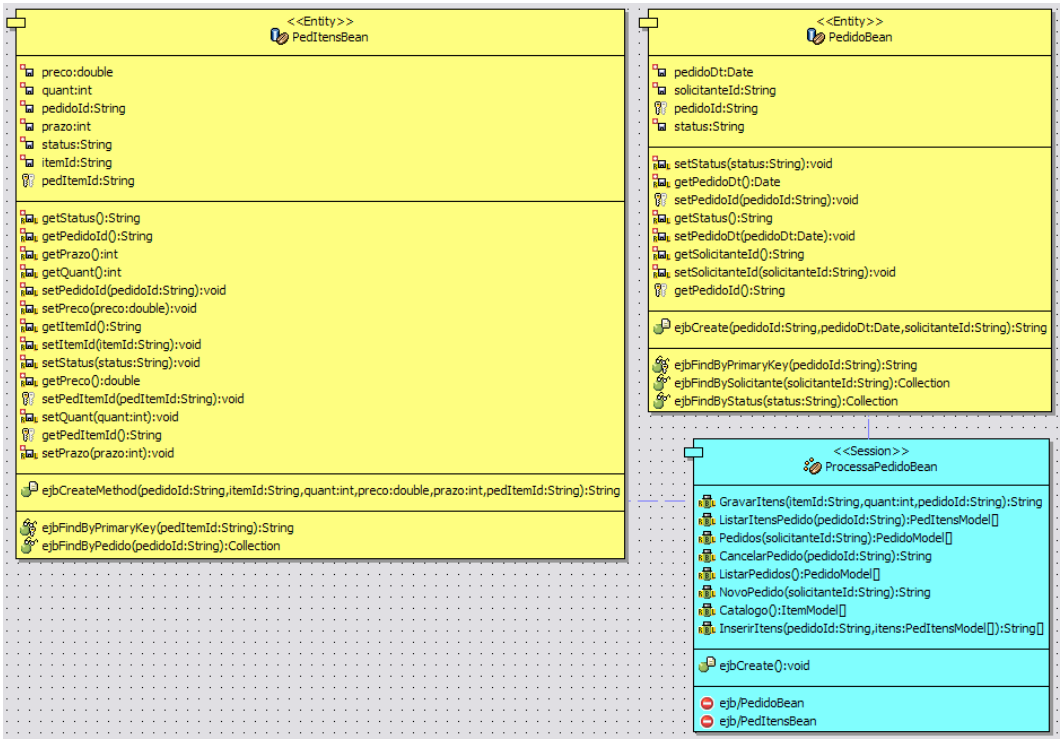

Figura 27 – *Entity Beans* e *Session Beans* do módulo de fornecedor

A implementação foi facilitada pela geração automática das interfaces da classe, que são acessadas pelos *servlets* e Web services. Nosso trabalho foi definir os métodos e implementar a lógica da aplicação:

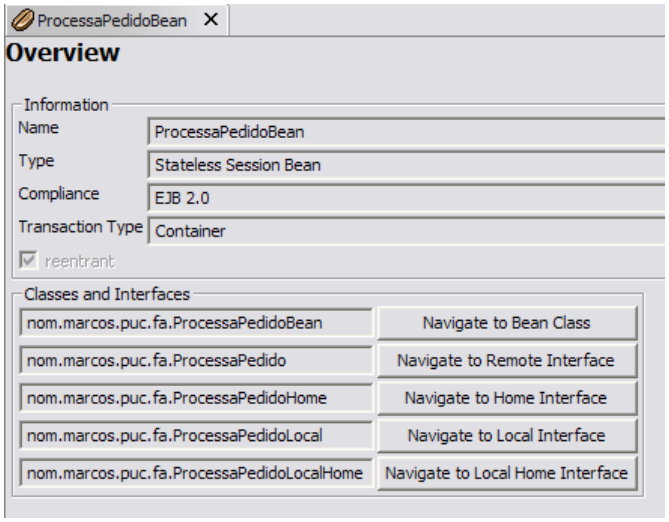

Figura 28 – Interfaces dos *beans* geradas automaticamente pelo Eclipse

Neste ponto retomamos o módulo Web e criamos os *servlets* e JSPs para exibição dos pedidos e seus itens, a partir do EJB ProcessaPedido. Estava concluído o módulo do fornecedor isolado. Como o objetivo era expor seus métodos em um Web service, geramos sua definição diretamente no *Eclipse*:

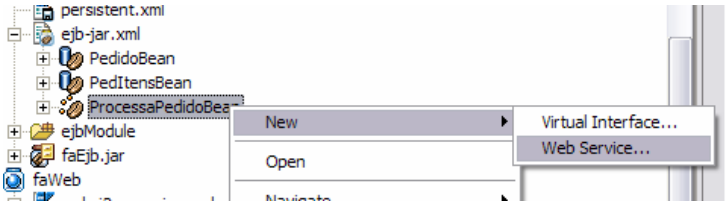

Figura 29 – Geração do Web service do módulo de fornecedor

A execução do *wizard* envolve a seleção dos métodos que serão expostos e a definição do nome da *interface*. O *Eclipse* gera o arquivo XML com a definição da *interface* virtual e referencia as classes necessárias:

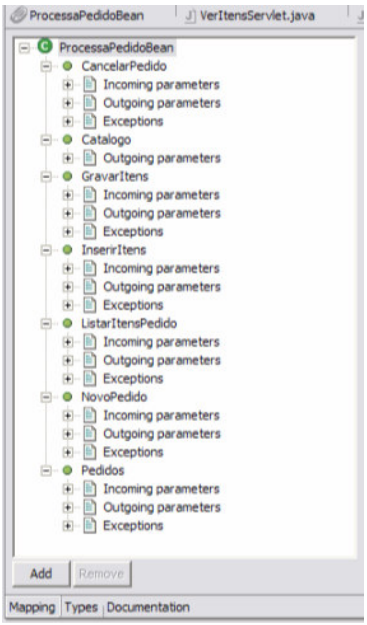

Figura 30 - Definições geradas para a *interface* do Web service

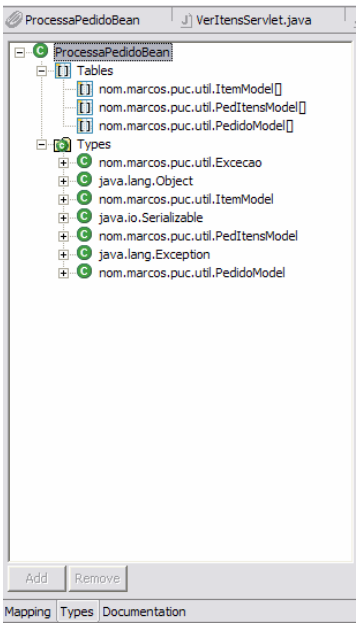

Figura 31 – Classes referenciadas pela *interface* do Web service

Uma vez implantados no WebAS, o SAP Netweaver disponibiliza uma *interface* Web para teste dos serviços, que foi muito útil durante o desenvolvimento. Mesmo antes de iniciar o módulo do solicitante, já podíamos consumir o Web service, experimentando diferentes valores de entrada para cada método. A figura a seguir mostra as operações do Web service do Fornecedor A:

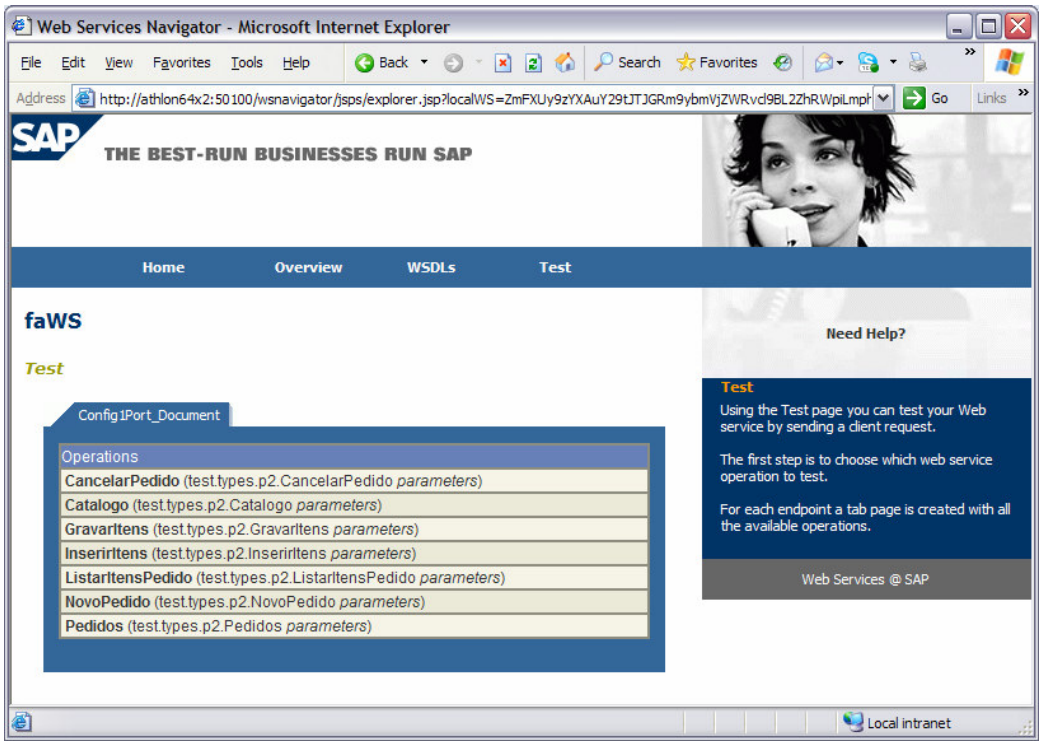

Figura 32 – Console para teste de Web services do SAP Netweaver

Este é o resultado da requisição SOA gerada no teste da operação “Pedidos” (Listar Pedidos), dado um código de solicitante. Este console é baseado na tecnologia SAP *Web* Dynpro:

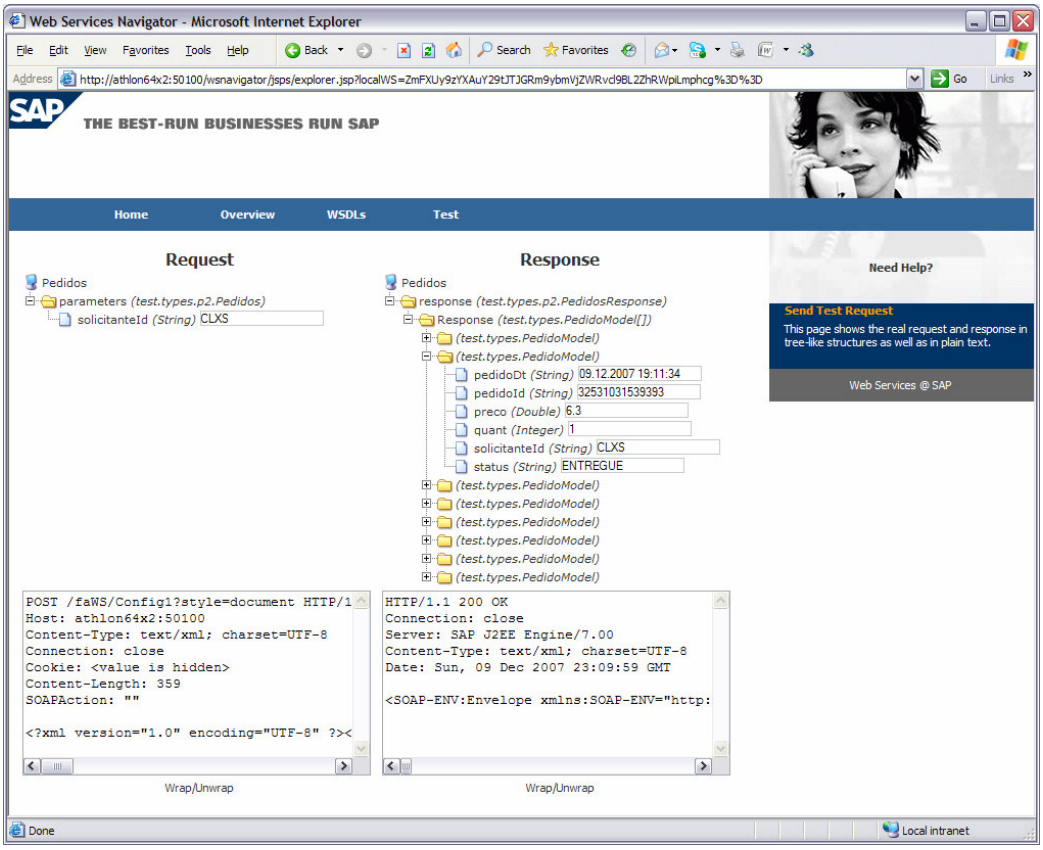

Figura 33 – Resultado de teste de WS no console do SAP Netweaver

4.5.2. Módulo dos solicitantes

A SAP desenvolveu uma API Java para construção de aplicações com *interface* de usuários Web denominada *Web Dynpro*. É possível desenvolver aplicações usando esta API por meio de um *plug-in* do *Eclipse*, que a SAP denominou *SAP Netweaver Developer Studio*. Iniciamos o desenvolvimento definindo um esquema de *views* e sua navegação, que compõe a *interface* do usuário. Isto é feito graficamente no *plug-in*, conforme mostra a figura a seguir:

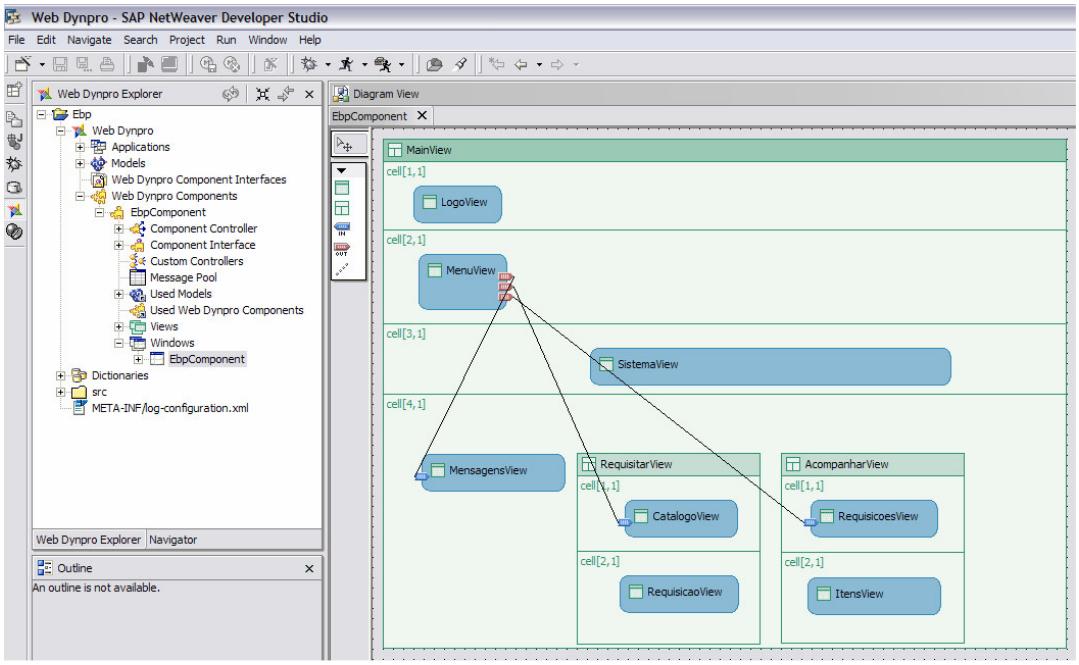

Figura 34 - *Plug-in Web Dynpro* no Eclipse

Cada *view* é montada como uma hierarquia de elementos da API, bastando definir sua posição na árvore e as respectivas propriedades:

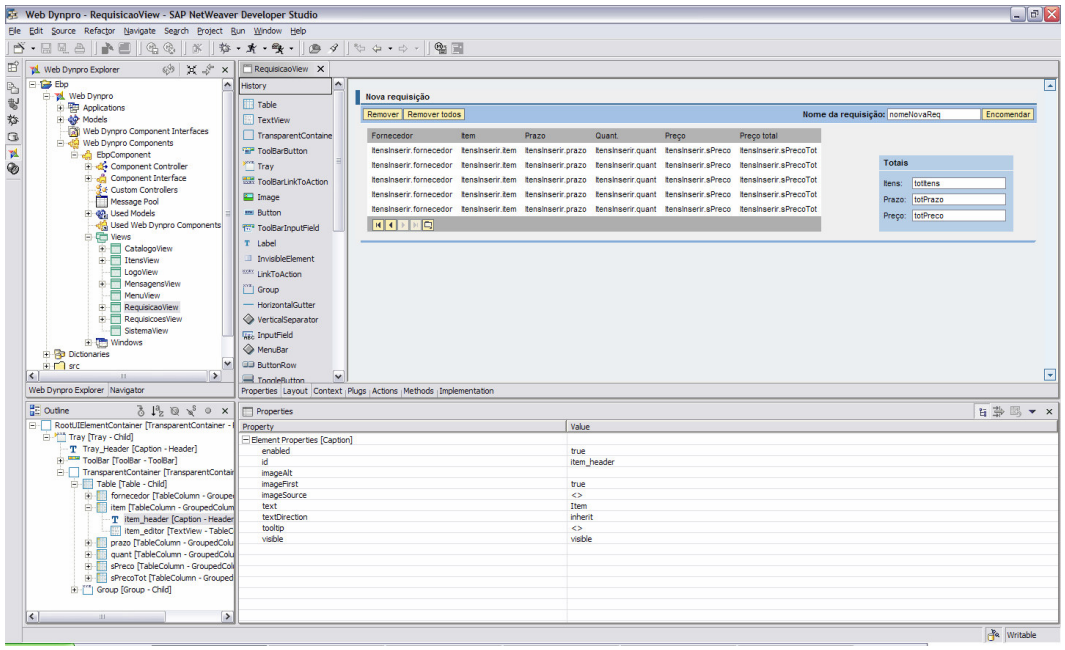

Figura 35 - Definição de *views Web Dynpro*

Para consumir o Web service é possível criar um modelo baseado no URL do WSDL. O *plug-in* importa a definição de todas as classes e métodos do Web service:

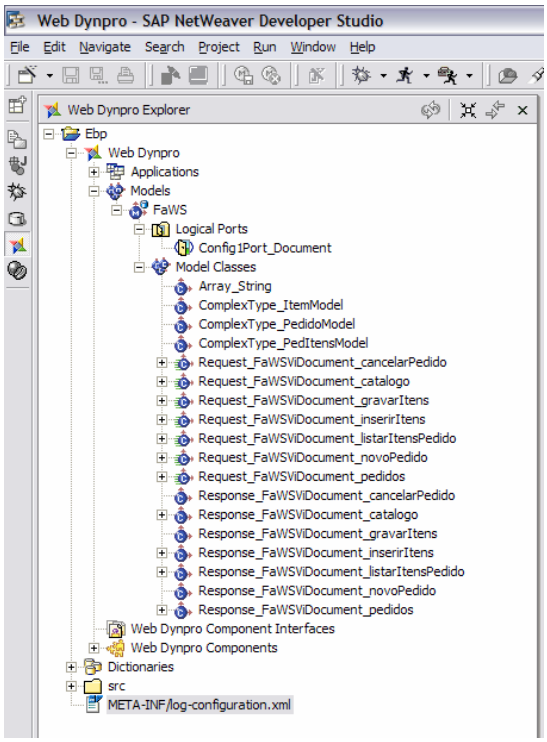

Figura 36 - Web service importado como modelo *Web Dynpro*

Uma vez associado o Web service ao modelo, este foi mapeado ao contexto do controlador e pode ser então consumido pelas *views*:

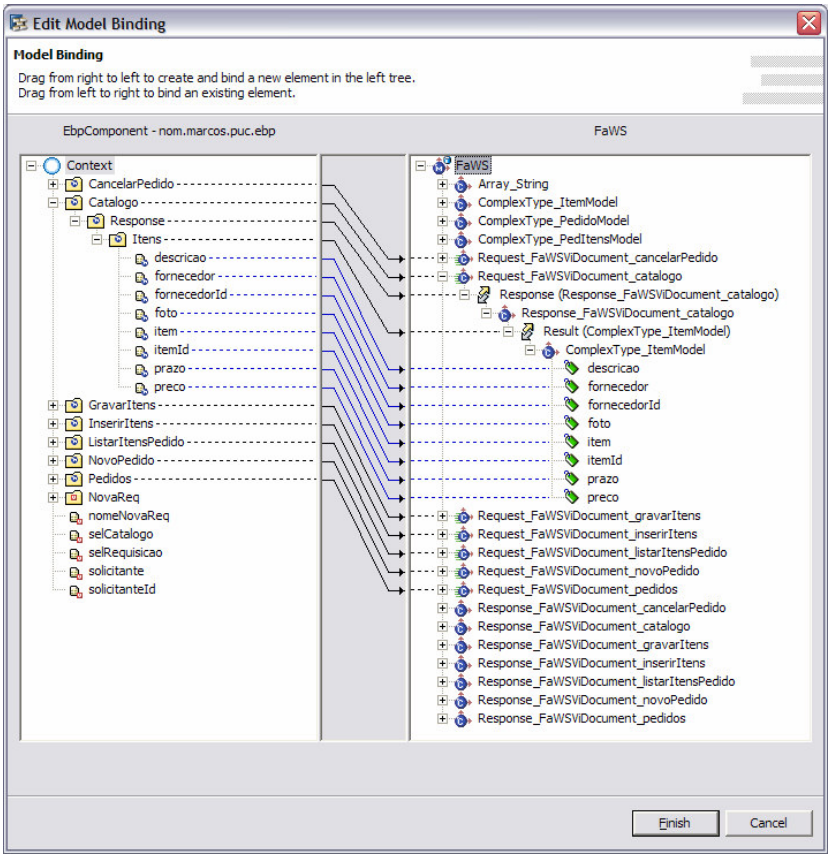

Figura 37 - Mapeamento do modelo ao contexto

Para ligar um elemento do tipo tabela ao contexto que por sua vez está associado ao Web service, basta definir uma associação (*binding*). Um *wizard* gera o código automaticamente. A figura a seguir mostra este *wizard* para associação de uma tabela ao contexto da *view*:

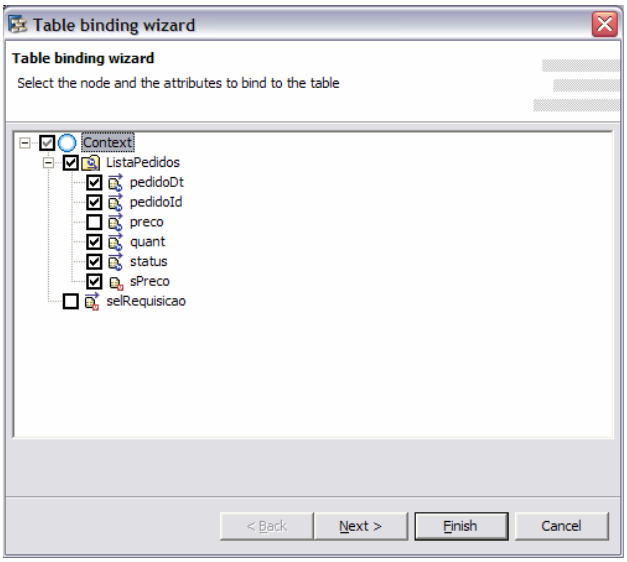

Figura 38 – *Wizard* para associação de uma tabela ao contexto da *view*

A implantação do módulo *Web Dynpro* é facilitada pelo *plug-in*. De fato, um arquivo EAR é gerado automaticamente neste processo.

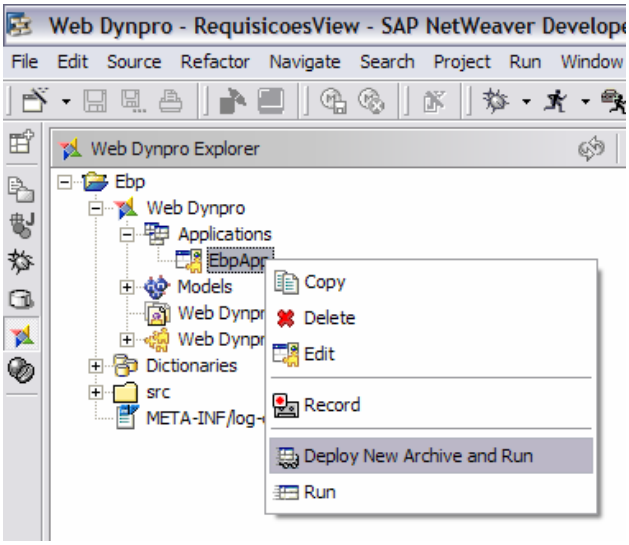

Figura 39 – *Deploy* do módulo *Web Dynpro*

4.5.3. Mediador

O mediador atua como um *proxy* dos Web services dos fornecedores, fazendo com que o módulo dos solicitantes trabalhe com todos eles como se

fossem um Web service único. O primeiro passo para acrescentar o mediador ao projeto foi importar as mensagens do Web service do fornecedor no repositório, por meio de seu WSD. Uma vez importadas, pudemos definir as respectivas *message interfaces* do mediador, que são classes para processamento síncrono ou assíncrono, de entrada ou saída, ou ainda abstratas (independentes). Observar que a definição destas classes depende apenas das mensagens utilizadas, não do número de fornecedores. Para que todos os fornecedores compartilhem do mesmo conjunto de mensagens, definimos seus Web Services utilizando um *namespace* comum: ***urn:fornWsd/fornWsVi/document*.**

Em seguida construímos mapeamentos diretos entre estas *message interfaces* de entrada e saída para testar a mediação. Definimos o *Business Service* EBPXI com o *Communication Channel* EBP_Client como Web service para requisição de catálogos no mediador, expondo a *interface* Catalogo_OutboundSync. Na etapa de determinação do destinatário, ligamos as mensagens desta *interface* diretamente ao Fornecedor A. A partir das definições do Business Service EBPXI o ambiente faz a geração de um WSD, que importamos no módulo dos solicitantes, substituindo o modelo do fornecedor A. Testamos o funcionamento da aplicação, que permaneceu o mesmo após a inclusão do mediador. Notamos, porém, uma demora maior na resposta, causada pelo processo de conversão de mensagens.

O próximo passo foi tornar o destinatário um parâmetro da mensagem de requisição de catálogo. Na fase de determinação do destinatário, o mediador pode selecionar uma *message interface* de acordo com o conteúdo da chamada. Este processo consiste em associar expressões condicionais em relação ao conteúdo da mensagem para cada *message interface*, como em uma cláusula *switch*. Nosso objetivo é verificar a presença do código do fornecedor correspondente a cada *message interface* associada. Para isto, modificamos a definição importada da mensagem de solicitação, incluindo os elementos id e num:

De:

```
<xs:element name="Catalogo">
     <xs:complexType>
        <xs:sequence />
     </xs:complexType>
</xs:element>
```

Para:

```
<xs:element name="CatalogoXI">
     <xs:complexType>
```

```
            <xs:sequence>
               <xs:element name="id" type="xs:string"
                 nillable="true" />
               <xs:element name="num" type="xs:int" />
            </xs:sequence>
         </xs:complexType>
      </xs:element>
```

O elemento id contém o código do fornecedor, e o elemento num contém o número de catálogos desejados. Este último parâmetro é processado pelo *BPM*, que será detalhado mais adiante. Feitos os ajustes no WSD para a nova mensagem, definimos um *Business Service* para cada fornecedor e referenciamos sua interface na fase de determinação do destinatário, com base no campo id da mensagem. Desta maneira habilitamos a seleção de um ou mais fornecedores durante a chamada do catálogo.

Neste ponto do projeto, ficou definido que cada fornecedor teria um *Business Service* e um id próprios, cadastrados diretamente no mediador. De fato, este possui um console de configuração independente do console de projeto, onde são definidas as mensagens, interfaces e os processos. Neste console de configuração é cadastrado o URL do fornecedor, bem como os dados de autenticação e outros atributos técnicos. Como os fornecedores são independentes, faz sentido manter estas características separadas por fornecedor. Além disso, o processo de adesão ocorreria em duas etapas distintas: o cadastramento dos atributos funcionais no módulo de materiais, e a configuração dos atributos técnicos no mediador. O atributo de associação entre estes cadastros é o código do fornecedor, informado no campo id da mensagem.

De maneira semelhante, acrescentamos o elemento id a todas as mensagens de requisição do mediador. Assim, quando a aplicação solicita o cancelamento de um pedido, por exemplo, informa por meio do elemento id na requisição qual fornecedor deve ser acionado. Por meio de uma transformação de mensagens, este elemento é removido da requisição, antes de ser passado para o fornecedor. A figura a seguir mostra o mapeamento responsável pela remoção do elemento id de uma requisição para cancelamento de pedido:

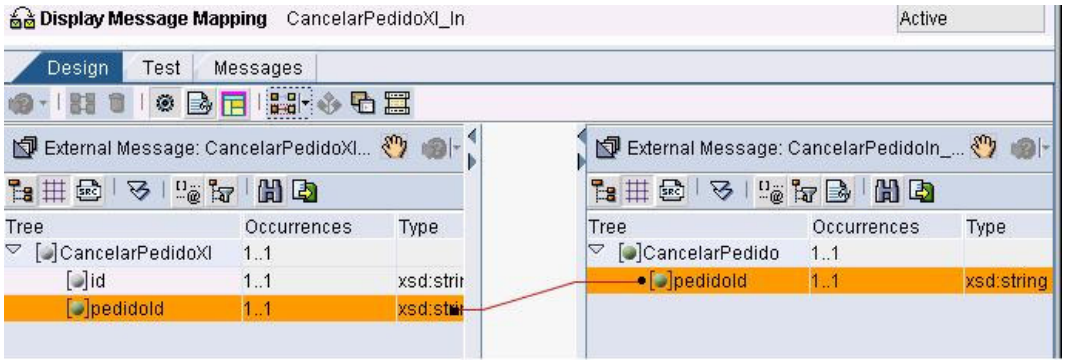

Figura 40 – Mapeamento para remoção do elemento id

Como definido na arquitetura de mediação flexível, passamos a implementar a agregação de catálogos. O objetivo é permitir que a aplicação indique mais de um fornecedor na solicitação de catálogo, e que as respostas sejam reunidas em uma única mensagem, como se viessem todas de um único fornecedor. Para atingir este objetivo, definimos um *integration process* no XI, que permite o processamento de mensagens utilizando a linguagem WS-BPEL em um módulo denominado *BPM* (*Business Process Module*). Os *integration processes* deste módulo são conhecidos como *bpm*s.

Modificamos a fase de determinação do destinatário de modo que todas as chamadas para a interface Catalogo_OutboundSync sejam encaminhadas para este *bpm*, que denominamos FlexComp_PAR.

O processamento WS-BPEL é feito por meio de mensagens assíncronas. Para processar mensagens de uma interface síncrona, é preciso definir uma ponte síncrona/assíncrona, por meio de um par *receiver/sender* no início e no final do processo, respectivamente. A mensagem recebida pelo *receiver* é respondida com a mensagem definida para o *sender*, a *interface* fica aguardando o processamento até que esta mensagem seja gerada. Conforme nosso objetivo, após o recebimento da mensagem pelo *receiver*, o sistema irá consultar os fornecedores envolvidos e agregar as suas mensagens de resposta em uma mensagem única, que será retornada por meio do *sender* à *interface* síncrona. Este processo é custoso e dependente do desempenho dos fornecedores em responder à requisição, o que causa um tempo de resposta elevado a cada consulta.

Após o passo de recepção, há o passo de determinação de destinatários. Este carrega o conteúdo do campo id da mensagem como critério, e aciona a determinação de destinatários definida na configuração do mediador para aquele *integration process*. O resultado é uma lista de destinatários, que fica

armazenada em uma variável própria que é acessada pelo elemento seguinte, um *sender*, que dispara uma solicitação para cada fornecedor. Cada solicitação é gerada com base numa transformação da solicitação original, sendo removidos os elementos id e num.

O SAP XI define um bloco *ParForEach*, que efetua processamentos paralelos baseado em uma lista de destinatários. Este bloco é análogo ao bloco *forEach* do WS-BPEL, em modo paralelo:

```
<sap-extn:parforeach name="Recebe Fornecedores"
select="Receivers" variable="Receiver" xmlns:sap-
extn="http://www.sap.com/xi/extensions">

<forEach counterName="BPELVariableName"
parallel="yes|no"
standard-attributes>
```

Assim, colocamos um *receiver* dentro deste bloco, sendo ele multiplicado pelo bloco em função do número de destinatários na lista *Receivers*.

Neste ponto, esbarramos em uma forte restrição do mediador SAP XI: não é possível definir este bloco quando o processamento é síncrono. Esta restrição deriva do mecanismo de workflow SAP, que suporta a execução do WS-BPEL neste ambiente. A figura a seguir mostra o erro "*Multiple receivers are not permitted on synchronous calls"* ao executar o workflow:

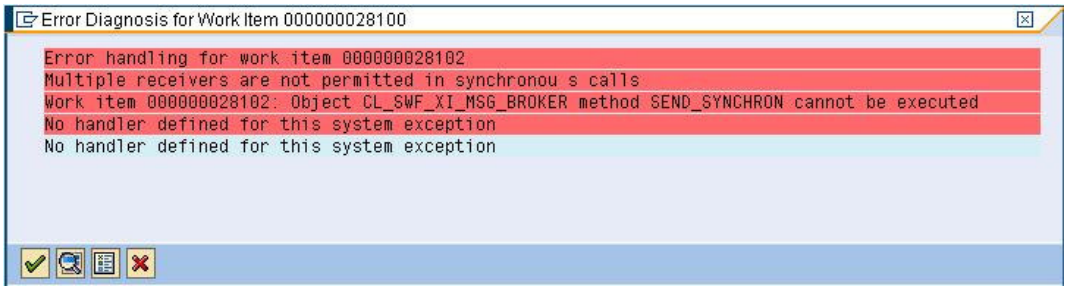

Figura 41 – Erro ao executar *workflow*

Estava claro que a chamada de catálogo tinha que ser síncrona, pois a resposta do fornecedor teria que vir com o menor custo possível. A princípio chegamos a abandonar o uso do bloco *ParForEach* e utilizamos um bloco *Fork*, mas desta forma acabamos acoplando o número de fornecedores ao código do *bpm*. Nos fóruns de desenvolvedores SAP acabamos encontrando uma saída interessante *[20]*: definir um outro *integration process*, que por definição é assíncrono, e nele realizar a comunicação síncrona com o fornecedor. Desta maneira, transferimos o custo de comunicação para dentro do mediador, e superamos a limitação do bloco *ParForEach* com relação ao *receiver* síncrono.

Este *integration process* de comunicação é igual para todos os fornecedores, sendo instanciado em função da lista de destinatários do *sender*, assim como ocorre no bloco *ParForEach*. O denominamos Forn_*BPM*:

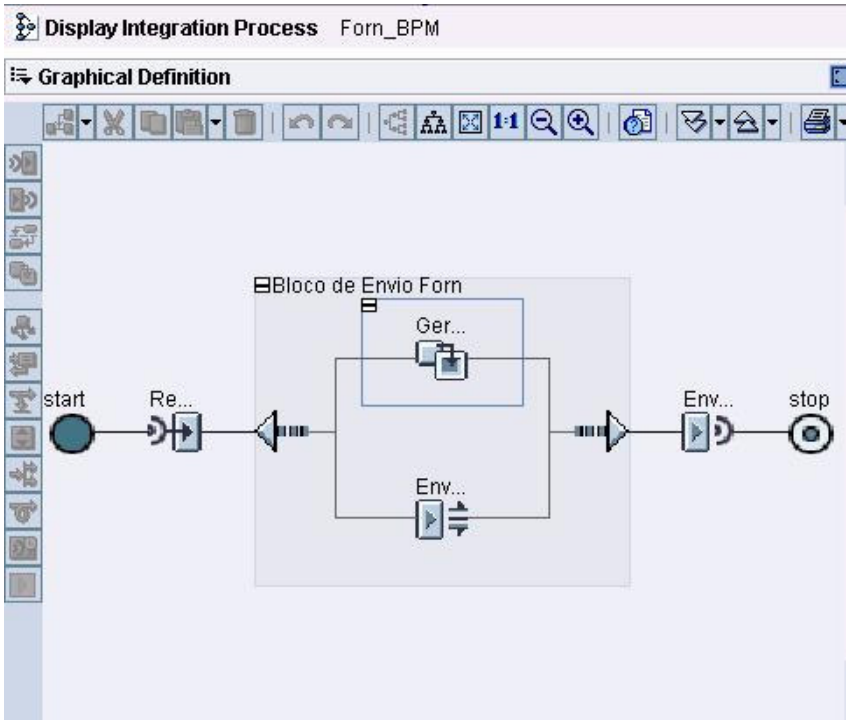

Figura 42 – Representação gráfica do *integration process* Forn_BPM

Na linha de execução superior do bloco de envio, temos uma cláusula de exceção, que é acionada caso o fornecedor não esteja disponível no momento da consulta. O sistema aguarda até 5 segundos pelo estabelecimento da conexão com o fornecedor, e até 60 segundos pelo retorno do processamento. Estes parâmetros são configuráveis. Caso o fornecedor exceda estes prazos, a cláusula de exceção é acionada, e uma transformação gera uma mensagem de retorno vazia, que é encaminhada para o *sender* como resposta. Desta maneira, o processo principal sempre obtém as respostas dentro de um tempo limitado, evitando que o solicitante aguarde demais por um fornecedor que não responde às requisições em um tempo adequado.

As mensagens coletadas pelo *receiver* no bloco *ParForEach* do *bpm* principal são coletadas uma a uma dentro de uma condição switch, que compara um contador ao elemento num informado na requisição. Caso o número de mensagens desejado seja atingido, o bloco *ParForEach* é finalizado e as mensagens coletadas são enviadas para o bloco seguinte. Desta maneira é possível acionar um grande número de fornecedores e coletar apenas as n

primeiras respostas, evitando-se inclusive aqueles fornecedores que demoram demais ou nem chegam a responder.

O bloco seguinte verifica se há respostas válidas coletadas, pois é possível que, numa chamada envolvendo um único fornecedor, este não responda. Havendo um ou mais elementos no *container* com as respostas, estes são transformados por um mapeamento de mensagens em uma mensagem única. Este mapeamento é denominado pela SAP de *CollectMerge*, e utiliza uma transformação no código XML da mensagem chamada *removeContexts*. Como o nome sugere, esta transformação retira os níveis superiores da hierarquia dos elementos de uma mensagem, permitindo que os elementos de uma lista sejam mesclados com outros elementos de outra lista. A imagem a seguir ilustra esta transformação:

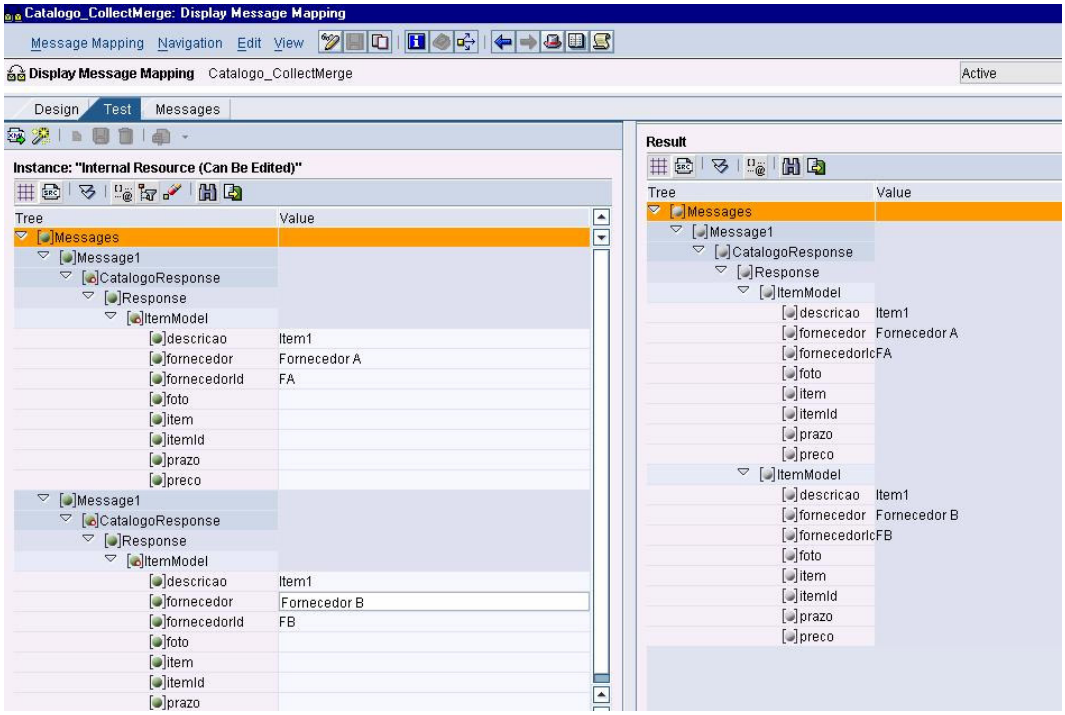

Figura 43 – Transformação *removeContexts*

Por fim, esta mensagem com as respostas mescladas é enviada para o *sender*, que retorna a chamada síncrona que deu origem ao processamento do *bpm*. Caso o sistema não tenha recebido respostas válidas, um bloco gera uma resposta vazia de retorno. Assim, realizamos a agregação de catálogos de diversos fornecedores, que para a aplicação aparecem com um. Os parâmetros técnicos da composição, como a disponibilidade e o tempo de resposta dos fornecedores foram levados em conta, bem como os fornecedores envolvidos e número de catálogos desejados.

4.5.4. Módulo de materiais

A composição funcional é baseada nos dados do repositório. Para isto definimos uma ligação entre o módulo dos solicitantes e o módulo de materiais do ERP, onde ficam armazenados os cadastros de fornecedores, solicitantes, requisições e pedidos. Este módulo não estava disponível para uso, por essa razão criamos um módulo J2EE *ebpEJb*, semelhante ao dos fornecedores, para simular suas funções.

Iniciamos definindo as tabelas em um projeto de dicionário de dados, e suas respectivas classes como *CMP Beans*. Além dos cadastros citados no parágrafo anterior, acrescentamos outros específicos para a definição das composições, que são as listas de fornecedores, listas de regiões e o cadastro de composições em si. Definimos também classes de apoio para o escalonamento e suspensão de fornecedores.

O session bean *EbpBean* expõe as funções do módulo de materiais como um Web service interno com autenticação no acesso, de acordo com as premissas do projeto.

Definimos os cadastros de solicitantes e fornecedores baseado em arquivo, assim como fizemos para os catálogos no módulo de fornecedores. O cadastro de solicitantes é lido durante a inicialização da camada de aplicação, e o cadastro de fornecedores é utilizado pelas composições no módulo de materiais. Passamos a detalhar o processo de composição funcional.

Cada fornecedor possui os atributos id, conceito e uf (unidade federativa), que são avaliados durante a composição. O atributo conceito é numérico, pode ser indicado um valor mínimo para este valor em determinada composição. Já o código do fornecedor (id ou fornecedorId) é o mesmo utilizado no mediador, pode ser associado a listas de fornecedores, como ‘ISO 9000’, ou ‘Entrega em 24hs’, indicando características do fornecedor que podem ser exigidas em uma composição. O atributo uf do fornecedor pode estar associado a uma ou mais listas de regiões, como ‘Sudeste’ ou ‘Rio de Janeiro’.

No módulo de materiais implementamos o mecanismo de suspensão, que armazena temporariamente em memória os fornecedores que deixaram de responder a uma chamada. Para fins de simulação, definimos este tempo como 20 segundos e cadastramos o fornecedor D, que de fato sequer foi implantado, para exercitar este mecanismo de suspensão.