Uma Infra-Estrutura para Coordenação de Atividades em Cadeias Produtivas Baseada em Coreografia de Serviços Web

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Alan Massaru Nakai e aprovada pela Banca Examinadora.

Campinas, 13 de setembro de 2007.

Prof. Dr. Edmundo Roberto Mauro Madeira
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Júlia Milani Rodrigues – CRB8a / 2116

Nakai, Alan Massaru

N145i Uma infra-estrutura para coordenação de atividades em cadeias produtivas baseada em coreografia de serviços web / Alan Massaru Nakai -- Campinas, [S.P. :s.n.], 2007.

Orientador : Edmundo Roberto Mauro Madeira

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Gerencia de cadeias produtivas. 2. Coreografia de serviços web. 3. Orquestração de serviços web. I. Madeira, Edmundo Roberto Mauro. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

(mjmr/imecc)

Título em inglês: An infrastructure for coordination of supply chain activities based on web services choreography

Palavras-chave em inglês (Keywords): 1. Supply chain management. 2. Web service choreography. 3. Web service orchestration.

Área de concentração: Sistemas distribuídos

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Edmundo Roberto Mauro Madeira (IC-UNICAMP)
Prof. Dr. Eleri Cardozo (FEEC-UNICAMP)
Prof. Dr. Luiz Eduardo Buzato (IC-UNICAMP)

Data da defesa: 23/03/2007

Programa de Pós-Graduação: Mestrado em Ciência da Computação

Substitua pela folha com as assinaturas da banca

Instituto de Computação
Universidade Estadual de Campinas

Uma Infra-Estrutura para Coordenação de Atividades em Cadeias Produtivas Baseada em Coreografia de Serviços Web

Alan Massaru Nakai[1]

Março de 2007

Banca Examinadora:

- Prof. Dr. Edmundo Roberto Mauro Madeira (Orientador)
- Prof. Dr. Eleri Cardozo
  Faculdade de Engenharia Elétrica e de Computação - UNICAMP
- Prof. Dr. Luiz Eduardo Buzato
  Instituto de Computação - UNICAMP
- Profa. Dra. Islene Calciolari Garcia (Suplente)
  Instituto de Computação - UNICAMP

[1]Suporte financeiro de: Bolsa Capes.

Resumo

Uma cadeia produtiva é definida como o conjunto de atividades envolvidas na criação, transformação e distribuição de um produto, da matéria-prima ao consumidor final. Os participantes da cadeia produtiva podem trabalhar de forma integrada, com o objetivo de otimizar o desempenho coletivo, aumentando sua competitividade comercial. Do ponto de vista tecnológico, a natureza distribuída, autônoma e heterogênea dos participantes da cadeia produtiva dificultam a automação dos seus processos de negócio interorganizacionais. Este trabalho propõe uma infra-estrutura baseada em coreografias de serviços Web para coordenar as atividades que compõem os processos de negócio interorganizacionais das cadeias produtivas. Esta infra-estrutura implementa um modelo de coordenação que visa facilitar o projeto e a implantação dos processos de negócio interorganizacionais. Neste modelo, os processos são representados por coreografias WS-CDL, que são mapeadas para planos de coordenação executáveis descritos em BPEL. O trabalho também apresenta um protótipo da infra-estrutura, com o objetivo de validá-la.

Abstract

A supply chain is the set of activities involved in the creation, transformation and distribution of a product, from raw material to the consumer. Supply chain's participants can work in an integrated way to optimize their performance and increase their commercial competitiveness. From the technological point of view, the distributed, autonomous and heterogeneous nature of supply chain's participants raises difficulties when we consider the automation of interorganizational processes. This work proposes an infrastructure based on Web services choreographies for coordination of the activities that compose the interorganizational business processes of the supply chains. This infrastructure implements a coordination model that aims to make easier the design and the deployment of the interorganizational business processes. In this model, processes are represented by WS-CDL choreographies, which are mapped to executable BPEL coordination plans. The work also presents a prototype of the infrastructure to validate it.

Agradecimentos

Ao meu orientador, Prof. Dr. Edmundo Madeira, pela paciência e dedicação.

À minha querida esposa, Andréia, pelo amor e carinho em todos os momentos.

Aos meus pais e à minha irmã, pela confiança e apoio.

Ao Evandro, pelo incentivo e ajuda.

Aos amigos do LIS, pelo companheirismo e pela experiência compartilhada.

Este trabalho foi financiado pela CAPES.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Uma cadeia produtiva pode ser definida como o conjunto de atividades envolvidas na distribuição de um produto, da matéria-prima ao consumidor final. Inclui fornecimento de matéria-prima, manufatura, armazenamento, distribuição, transporte e todos os sistemas de informação necessários para monitorar e gerenciar estas atividades [15].

Os participantes de uma cadeia produtiva podem trabalhar de forma integrada, usando técnicas de gerência, a fim de otimizar o desempenho coletivo na criação, distribuição e suporte ao produto final [16]. A integração entre parceiros de negócio tem reflexos importantes na competitividade comercial dos envolvidos, por exemplo, agilizando a movimentação dos produtos pela cadeia, diminuindo os estoques e, conseqüentemente, baixando os custos do produto final.

Do ponto de vista tecnológico, a natureza distribuída, autônoma e heterogênea dos participantes de uma cadeia produtiva pode tornar custosa a integração dos seus sistemas de informação e *workflows* internos, dificultando a automatização dos processos de negócio interorganizacionais. Além disso, um sistema para integração de cadeias produtivas deve prover flexibilidade suficiente para acompanhar a sua dinamicidade. Alterações na demanda do consumidor e na capacidade de produção dos fornecedores fazem com que os relacionamentos entre os parceiros de negócio evoluam com o tempo.

A tecnologia de serviços Web vem sendo apontada por muitos autores como uma alternativa viável para prover interoperabilidade entre parceiros de negócio. Serviços Web são aplicações de *software* disponibilizadas pela Web que apresentam mecanismos de comunicação padronizados baseados em mensagens XML (*Extensible Markup Language*) e protocolos bastante difundidos, como o HTTP (*Hypertext Transfer Protocol*). Esta tecnologia pode complementar (ou substituir) os sistemas legados, facilitando a integração interorganizacional de parceiros de negócio.

Com o advento da tecnologia de serviços Web, surge um novo paradigma de computação. Neste paradigma, os serviços tornam-se blocos básicos por meio dos quais são

criadas novas aplicações. É a chamada computação orientada a serviços (SOC - *Service-oriented computing*) [6]. O desenvolvimento de aplicações baseadas neste paradigma depende fundamentalmente da composição de serviços. Uma composição combina vários serviços, de acordo com um padrão de composição, visando alcançar um objetivo de negócio, solucionar um problema específico ou prover um novo serviço.

Composições de serviços Web podem ser descritas como coreografias ou orquestrações [17]. Ambas definem as possíveis interações que podem ocorrer em uma colaboração que envolve múltiplos serviços, porém as definições destes conceitos diferem ligeiramente. A coreografia está relacionada à troca pública de mensagens que ocorre na colaboração, de um ponto de vista global. Já a orquestração descreve, de um ponto de vista individual, as interações com outros serviços, no nível da troca de mensagens, incluindo uma lógica de negócios e a ordem da execução destas interações.

Diversas linguagens para representação de coreografias e orquestrações já foram propostas. Exemplos destas linguagens são WS-CDL (*Web Services Choreography Description Language*), para coreografia, e BPEL (*Business Process Execution Language*), para orquestração. As linguagens para orquestração definem processos executáveis e normalmente possuem implementações que interpretam e executam estes processos. Já as linguagens para coreografia têm uma função descritiva, e não podem ser diretamente executadas.

Este trabalho de mestrado está inserido no contexto de um projeto que está sendo desenvolvido no IC-UNICAMP. Tal projeto, que envolve outros dois alunos de pós-graduação (Evandro Bacarin e Andréia Kondo), tem o objetivo de desenvolver uma arquitetura, baseada em serviços Web, para integração de cadeias produtivas. A arquitetura visa cobrir diversos aspectos da gerência das cadeias, incluindo a execução de processos de negócio, a rastreabilidade de produtos e a negociação de contratos. O escopo deste trabalho de mestrado abrange a especificação e o desenvolvimento dos mecanismos para execução dos processos de negócio interorganizacionais para a arquitetura. Para tanto, propõe-se uma infra-estrutura para coordenação das atividades que compõem estes processos. Esta infra-estrutura baseia-se em um modelo de coordenação que tem o objetivo de facilitar o projeto e a implantação de processos de negócio interorganizacionais. Neste modelo, as atividades da cadeia produtiva são encapsuladas por serviços Web e os processos de negócio são representados por coreografias WS-CDL.

O modelo de coordenação define uma camada de gerência básica que fornece suporte para a execução das coreografias. Esta camada trata da conexão entre os participantes do processo de forma transparente aos projetistas. Isto permite que as coreografias sejam projetadas no nível dos papéis a serem executados. A ligação entre estes papéis e as entidades que os desempenham é feita em tempo de execução, permitindo a configuração dinâmica dos participantes da coreografia. Esta característica, chamada de ligação dinâmica, representa uma das contribuições deste trabalho.

A execução das coreografias é realizada por elementos da infra-estrutura chamados de *gerentes de coordenação*, que são responsáveis por interpretar e executar os *planos de coordenação*. Cada plano é composto por um conjunto de instruções que correspondem ao comportamento de um papel definido por uma coreografia WS-CDL. Utilizou-se a linguagem BPEL para representação destes planos. A realização deste trabalho incluiu o mapeamento entre coreografias WS-CDL e planos de coordenação BPEL.

Em resumo, as principais contribuições deste trabalho são:

- Um modelo de coordenação, baseado em coreografias WS-CDL, que visa a implantação eficiente de processos de negócio interorganizacionais e permite a ligação dinâmica entre os participantes da cadeia.

- Uma infra-estrutura que fornece suporte ao modelo, possibilitando a execução das coreografias WS-CDL referentes aos processos de negócio interorganizacionais da cadeia produtiva por meio de planos de coordenação representados em BPEL.

- A implementação de um protótipo que valida a infra-estrutura proposta.

Esta dissertação é organizada da seguinte forma. O Capítulo 2 introduz a fundamentação teórica necessária para o entendimento do restante do texto. São abordados os conceitos de serviços Web, coreografia e orquestração de serviços Web e cadeias produtivas. O modelo de coordenação e a infra-estrutura propostos são apresentados nos Capítulos 3 e 4, respectivamente. O Capítulo 5 descreve o protótipo implementado e o Capítulo 6 aborda alguns trabalhos que enfocam questões relevantes ao trabalho realizado. Finalmente, no Capítulo 7, são apresentadas as considerações finais.

Capítulo 2

Conceitos e Tecnologias

Este capítulo apresenta os conceitos de serviços Web, de coreografia e orquestração de serviços Web e de cadeias produtivas, que são essenciais para o entendimento do restante do trabalho.

2.1 Serviços Web

De forma genérica, um serviço Web pode ser definido como uma aplicação acessível por meio da Web. Apesar de correta, esta definição é bastante abrangente, incluindo por exemplo, *scripts* CGI (*Common Gateway Interface*) ou qualquer outro programa acessível por uma URL (*Universal Resource Locator*) [18]. Uma definição mais refinada, que é adotada por este trabalho, é aquela fornecida pela W3C (*World Wide Web Consortium*) [19]. Segundo esta definição, um serviço Web é uma aplicação de *software* identificada por uma URI (*Uniform Resource Identifier*) cujas interfaces e implementações são definidas e descritas usando XML [20]. A descrição de um serviço Web pode ser descoberta por outros sistemas de *software*. Tais sistemas podem interagir com o serviço Web, de acordo com a descrição do mesmo, usando mensagens XML e protocolos da Internet.

A Figura 2.1 mostra a arquitetura [12] que contempla a definição adotada. Nesta arquitetura, o provedor de serviços disponibiliza aplicações em forma de serviços ao cliente. O provedor também fornece uma descrição da interface dos serviços, onde são detalhadas as operações disponíveis, os tipos de mensagens de entrada e saída para cada operação e a forma como as mensagens devem ser enviadas. A descrição da interface pode ser publicada em uma ou mais agências de descoberta que permitem que clientes encontrem os serviços de que necessitam a partir de uma ampla variedade de critérios. O cliente utiliza a interface do serviço - fornecida pelo próprio provedor ou eventualmente encontrado em uma agência de descoberta - para desenvolver ou configurar uma aplicação que interaja com o serviço desejado.

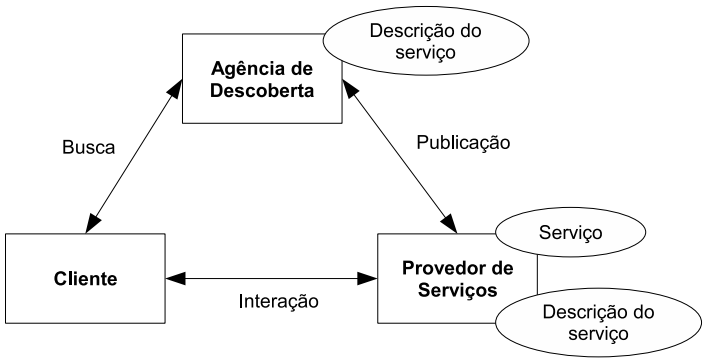

Figura 2.1: Arquitetura da tecnologia de serviços Web. Adaptado de [12].

As seguintes especificações suportam a arquitetura da Figura 2.1 [18, 21]:

- SOAP (*Simple Object Access Protocol*) [22, 18, 21]: protocolo utilizado para interação com serviços Web;
- WSDL (*Web Services Description Language*) [23, 18, 21]: linguagem para descrição de serviços Web;
- UDDI (*Universal Description, Discovery, and Integration*) [24, 18, 21]: registro para descrição de serviços Web.

Estas especificações são apresentadas nas Seções 2.1.1, 2.1.2 e 2.1.3, respectivamente.

2.1.1 SOAP

SOAP é um protocolo de comunicação, baseado em XML, que permite troca de informações por meio de mensagens [18]. A estrutura de uma mensagem SOAP é composta por um elemento principal, chamado `envelope`, com dois elementos filhos, um `header` e um `body`. O elemento `header` é opcional e encapsula informações de controle que não fazem parte da carga útil da mensagem. Estas informações de controle incluem, por exemplo, diretivas de roteamento ou informações de contexto relacionadas ao processamento da mensagem [22]. O elemento `body` encapsula a carga útil da mensagem, ou seja, a informação que o emissor deseja transmitir ao receptor.

SOAP pode ser utilizado para trocas simples de documentos XML ou para chamadas a procedimentos remotos [18]. No envio de um documento XML, emissor e receptor concordam com a estrutura do documento a ser enviado, que é encapsulado no `body` da mensagem. Nas chamadas a procedimentos remotos, uma mensagem carrega a chamada ao procedimento e outra mensagem carrega a resposta. A chamada inclui o nome da

operação invocada e os parâmetros de entrada. A resposta contém o resultado e os parâmetros de saída.

O Código Fonte 2.1 ilustra um exemplo de chamada a procedimento remoto. Neste exemplo, o emissor da mensagem invoca a operação "Compra", denotada pelo elemento `Compra` (linhas 7 a 12). O elemento complexo `Pedido` (linhas 8 a 11) carrega os parâmetros de entrada da operação. No `header` da mensagem, o elementos `Cliente` (linha 3) e `Senha` (linha 4) carregam informações referentes à identificação do emissor da mensagem.

Código Fonte 2.1: Exemplo

```
1  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope
       /">
2     <soap:Header>
3        <Cliente>Comprador</Cliente>
4        <Senha>12345</Senha>
5     </soap:Header>
6     <soap:Body>
7        <Compra>
8           <Pedido>
9              <Produto>Pizza</Produto>
10             <Quantidade>3</Quantidade>
11          </Pedido>
12       </Compra>
13    </soap:Body>
14 </soap:Envelope>
```

Ao invés de definir um novo protocolo de transporte, SOAP é transportado por protocolos já existentes, como HTTP e SMTP [21]. Note que, neste contexto, o termo transporte não se refere à camada de transporte da rede, mas sim à forma como uma mensagem SOAP é transportada. O fato de utilizar protocolos bastante difundidos como forma de transporte faz do SOAP uma alternativa atraente para se alcançar interoperabilidade em ambientes heterogêneos.

2.1.2 WSDL

WSDL define uma gramática XML para descrever serviços Web como coleções de *endpoints* de comunicação capazes de trocar mensagens [23]. Um documento WSDL apresenta duas partes [21]. A primeira delas fornece uma descrição abstrata da interface do serviço. Esta descrição inclui as operações disponibilizadas pelo serviço e os tipos de mensagens de entrada e saída para cada uma delas. Os elementos que compõem a descrição abstrata do serviço são:

- `types`: encapsula um conjunto de definições de tipos de dados utilizados nas trocas de mensagens. Apesar de WSDL aceitar outros sistemas de definição de tipos, o

esquema XML [25] é o mais utilizado e é tratado como o sistema de definição de tipos intrínseco à linguagem [23];

- `message`: representa uma definição abstrata de dados que são enviados ou recebidos por um serviço. Uma mensagem é composta por uma ou mais partes lógicas (elementos *part*), cada qual associada a uma definição de tipo;
- `portType`: encapsula um conjunto de operações lógicas. Cada operação define uma ação suportada por um serviço Web e relaciona suas mensagens de entrada e saída.

A segunda parte de um documento WSDL fornece informações concretas a respeito do acesso ao serviço. Estas informações são compostas pelos seguintes elementos:

- `binding`: fornece detalhes a respeito do protocolo de comunicação e do formato das mensagens para um `portType` específico.
- `port`: descreve um *endpoint* combinando as informações de um `binding` e um endereço de rede.
- `service`: agrupa um conjunto de `ports`.

O Código Fonte 2.2 mostra um documento WSDL que descreve um serviço Web para realizar compras *online*. Tal serviço disponibiliza uma única operação ao usuário, a operação "Compra". O elemento raiz de um documento WSDL é o elemento `definitions` (linhas 1 a 8). Este elemento é identificado por um atributo `name` e define o espaço de nomes relativo ao documento WSDL por meio do atributo `targetNamespace`. Além disso, o elemento `definitions` declara os espaços de nomes que serão visíveis por todo documento.

A parte abstrata do documento encontra-se entre as linhas 10 e 37. O elemento `types` (linhas 10 a 22) encapsula um esquema XML que define um elemento complexo `Pedido`. O elemento `Pedido` é utilizado na definição da mensagem de entrada da operação "Compra". Embora um único esquema XML tenha sido definido neste exemplo, o elemento `types` pode conter um ou mais esquemas XML.

As mensagens de entrada e saída da operação "Compra" são definidas nas linhas 24 a 30. A primeira mensagem (linhas 24 a 26), identificada pelo nome "PedidoMsg", define uma única parte lógica, correspondente a um elemento `Pedido`. A segunda (linhas 28 a 30), identificada pelo nome "RespostaMsg", também define uma única parte lógica, do tipo `string`.

O exemplo apresenta um único `portType` (linhas 32 a 37), identificado pelo nome "CompraPT". Cada `portType` pode definir uma ou mais operações. "CompraPT" define uma única operação, a operação "Compra" (linhas 33 a 36). O elemento `input` (linha 34),

define a mensagem de entrada da operação como sendo do tipo "PedidoMsg". O elemento `output` (linha 35), define a mensagem de saída como sendo do tipo "RespostaMsg".

Código Fonte 2.2: Exemplo

```
1  <wsdl:definitions
2     name="wsdl-exemplo.wsdl"
3     targetNamespace="any:wsdl-exemplo.wsdl"
4     xmlns:ns="any:appResponse.wsdl"
5     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
6     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7     xmlns:sens="any:schema-exemplo"
8     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
9
10    <wsdl:types>
11       <schema xmlns="http://www.w3.org/2001/XMLSchema"
12          targetNamespace="any:schema-exemplo"
13          xmlns:sens="any:schema-exemplo">
14          <complexType name="TipoPedido">
15             <sequence>
16                <element name="Produto" type="string"/>
17                <element name="Quantidade" type="string"/>
18             </sequence>
19          </complexType>
20          <element name="Pedido" type="sens:TipoPedido"/>
21       </schema>
22    </wsdl:types>
23
24    <wsdl:message name="PedidoMsg">
25       <wsdl:part name="partePedido" element="sens:Pedido"/>
26    </wsdl:message>
27
28    <wsdl:message name="RespostaMsg">
29       <wsdl:part name="parteResposta" type="xsd:string"/>
30    </wsdl:message>
31
32    <wsdl:portType name="CompraPT">
33       <wsdl:operation name="Compra">
34             <wsdl:input message="ns:PedidoMsg"/>
35             <wsdl:output message="ns:RespostaMsg"/>
36       </wsdl:operation>
37    </wsdl:portType>
38
39    <wsdl:binding name="CompraPTBinding" type="ns:CompraPT">
40       <soap:binding style="rpc" transport="http://schemas.xmlsoap.
             org/soap/http"/>
41       <wsdl:operation name="Compra">
```

```
42          <soap:operation soapAction="" style="rpc"/>
43          <wsdl:input>
44             <soap:body encodingStyle="" namespace="any:appResponse.
                  wsdl" use="literal"/>
45          </wsdl:input>
46          <wsdl:output>
47             <soap:body encodingStyle="" namespace="any:appResponse.
                  wsdl" use="literal"/>
48          </wsdl:output>
49       </wsdl:operation>
50    </wsdl:binding>
51
52    <wsdl:service name="CompraServ">
53       <wsdl:port binding="ns:CompraPTBinding" name="CompraServPort"
             >
54          <soap:address location="http://exemplo/CompraServ"/>
55       </wsdl:port>
56    </wsdl:service>
57
58 </wsdl:definitions>
```

As informações concretas de acesso ao serviço são definidas entre as linhas 39 e 56. O elemento `binding` (linhas 39 a 50) descreve detalhes relacionados ao `portType` "CompraPT", como protocolo (SOAP sobre HTTP), estilo de interação com o serviço (chamada a procedimento remoto) e codificação das mensagens. WSDL utiliza elementos de extensão para especificar estes detalhes. Existem extensões para SOAP, HTTP e MIME (*Multipurpose Internet Mail Extensions*). O exemplo utiliza a extensão para SOAP. Detalhes do elemento `binding` não são importantes no contexto desta dissertação, logo, serão omitidos.

O elemento `service` (linhas 52 a 56) define um serviço chamado "CompraServ". Este serviço apresenta um único *endpoint*, definido pelo elemento `port` (linhas 53 a 55). O atributo `binding` (linha 53) indica que as informações da interface do serviço são definidas no `binding` "CompraPTBinding". O elemento `address` especifica o endereço de rede do serviço.

2.1.3 UDDI

UDDI oferece aos usuários uma forma unificada e sistemática para encontrar provedores de serviços por meio de um registro centralizado de serviços [21]. A definição dos registros de serviços e sua operação são suportados por um conjunto de estruturas de dados e um conjunto de APIs (*Application Programming Interface*) que provêem mecanismos para buscar e atualizar os registros de serviços.

As principais estruturas de dados da UDDI são:

- `businessEntity`: descreve a organização que provê os serviços Web. Lista o nome, endereço e outras informações de contato da companhia;
- `businessService`: descreve um grupo de serviços Web relacionados que são oferecidos por um `businessEntity`. Um `businessEntity` pode possuir vários `businessServices`;
- `bindingTemplate`: descreve informações técnicas necessárias para usar um determinado serviço Web. Basicamente define o *endpoint* do serviço e referencia documentos do tipo `tModel` (próximo item). Um `businessService` pode possuir múltiplos `bindingTemplates`;
- `tModel`: é um conteiner genérico para qualquer tipo de especificação, como por exemplo uma interface WSDL. Diferente das outras estruturas, `tModels` não pertencem à mesma hierarquia de elementos, podendo ser referenciados por qualquer outro elemento.

O Código Fonte 2.3 mostra um exemplo de `businessEntity` que apresenta um único `businessService` (linhas 7 a 29). Cada elemento da UDDI é identificado unicamente em um registro de serviços por um UUID (*Universally Unique Identifier*), que é formado por uma seqüência longa de números hexadecimais. Note por exemplo, que o UUID do elemento `businessService` é especificado pelo atributo `serviceKey` (linha 7) e que o `businessService` referencia o UUID do elemento `businessEntity` ao qual pertence (atributo `businessKey` - linha 8). As linhas 9 a 10 descrevem o serviço correspondente ao `businessService` (detalhes foram omitidos). O `bindingTemplate` das linhas 12 a 22 descreve as informações técnicas para invocação do serviço. O elemento `accessPoint` (linhas 14 a 16) indicam o *endpoint* do serviço e o elemento `tModelInstanceInfo` (linhas 18 a 20) referencia um `tModel` que contém especificações mais precisas a respeito da invocação, como por exemplo a interface WSDL correspondente.

Código Fonte 2.3: Exemplo de `businessEntity`.

```
1 <businessEntity businessKey="A687FG00-56NM-EFT1-3456-098765432124">
2    <name>Nome da empresa</name>
3    <contacts>
4       <!--Contatos da empresa-->
5       ...
6    </contacts>
7    <businessService serviceKey="894B5100-3AAF-11D5-80DC-002035229
        C64"
8      businessKey="D2033110-3AAF-11D5-80DC-002035229C64">
```

```
9       <name>Nome do Servico</name>
10      <description>...</description>
11      <bindingTemplates>
12         <bindingTemplate bindingKey="6D665B10-3AAF-11D5-80DC
               -002035229C64"
13           serviceKey="89470B40-3AAF-11D5-80DC-002035229C64">
14            <accesssPoint URLType="http">
15               <!--endpoint do serviço-->
16            </accessPoint>
17            <tModelInstanceDetails>
18               <tModelInstanceInfo tModelKey="D2033110-3BGF-1KJH
                     -234C-09873909802">
19                  ...
20               </tModelInstanceInfo>
21            </tModelInstanceDetails>
22         </bindingTemplate>
23      </bindingTemplates>
24      <categoryBag>
25         <keyedReference tModelKey="..."
26           keyName="Compra de Livros"
27           keyValue="123456"/>
28      </categoryBag>
29   </businessService>
30   <categoryBag> ...
31   </categoryBag>
32 </businessEntity>
```

O Código Fonte 2.4 ilustra um `tModel` que referencia um documento WSDL. Cada elemento `overviewDoc`, dentro de um `tModel`, pode representar qualquer documento escrito em qualquer linguagem. No exemplo, o `overviewDoc` entre as linhas 4 e 9 representa um documento WSDL. A URL deste documento é indicada pelo elemento `overviewURL` (linhas 6 a 8).

Código Fonte 2.4: Exemplo de `tModel`.

```
1  <tModel tModelKey="...">
2     <name>http://www.travel.org/e-checkin-interface</name>
3     <description>Descrição do serviço</description>
4     <overviewDoc>
5        <description>Documento WSDL</description>
6        <overviewURL>
7           <!--URL da especificação-->
8        </overviewURL>
9     </overviewDoc>
10 <categoryBag> ... </categoryBag>
11 </tModel>
```

Uma característica importante das estruturas de dados da UDDI é a possibilidade de categorização. Elementos `businessEntity`, `businessService` e `tModel` podem ser categorizados para facilitar a descoberta de serviços que contemplem determinados requisitos. As categorias são definidas por elementos `keyedReference` dentro de elementos `categoryBag`. Por exemplo, no Código Fonte 2.3, o elemento `keyedReference` entre as linhas 25 e 27 categoriza o `businessService` correspondente como um serviço de "Compra de Livros" (atributo `keyName` - linha 26). O atributo `keyValue` (linha 27) define um identificador para a categoria e o atributo `tModelKey` (linha 25) referencia um `tModel` que contém uma lista de categorias.

A UDDI fornece seis APIs para busca, atualização e gerência de registros. No contexto deste trabalho, duas APIs são importantes, a API de busca (*UDDI Inquiry API*) e a API de publicação (*UDDI Publishers API*). Estas duas APIs são descritas a seguir.

- API de busca: fornece operações que permitem encontrar registros que satisfaçam certos critérios, como uma expressão regular, um UUID, uma categoria ou um `tModel`. Exemplos de operações:
  - `find_business`: retorna informações de elementos `businessEntity` que satisfazem os critérios de entrada;
  - `find_service`: retorna informações de elementos `businessService` que satisfazem os critérios de entrada;
  - `find_binding`: retorna informações de elementos `bindingTemplate` que satisfazem os critérios de entrada.
- API de publicação: fornece operações que permitem adicionar, modificar e remover registros. Exemplos de operações:
  - `save_business`: adiciona um elemento `businessEntity`;
  - `save_service`: adiciona um elemento `businessService` em um elemento `businessEntity`;
  - `delete_business`: remove um elemento `businessEntity`;
  - `delete_service`: remove um elemento `businessService`.

2.2 Coreografia e Orquestração de Serviços Web

Esta seção apresenta os conceitos de coreografia e orquestração de serviços Web. Além disso, introduz as linguagens WS-CDL e BPEL, que foram adotadas como as linguagens de coreografia e de orquestração neste trabalho.

2.2.1 Coreografia

A estrutura básica da arquitetura de serviços Web permite interações de aplicações de *software* por meio de mensagens XML e protocolos bastante difundidos, como o HTTP. Embora serviços Web, por natureza, proporcionem facilidades para integração de aplicações em ambientes distribuídos e heterogêneos, aplicações reais são tipicamente mais complexas do que invocações simples e independentes [18]. Utilizar um serviço em particular, normalmente envolve a execução de uma seqüência de operações em uma ordem particular. A integração de processos interorganizacionais, por exemplo, requer interações de longa duração que são dirigidas por modelos de processos explícitos, conforme afirma Aalst [26].

A seqüência de operações (trocas de mensagens) que ocorre entre um cliente e um serviço Web, como parte de uma invocação, é chamada de *conversação* de serviços Web [18]. Uma conversação pode envolver múltiplos serviços Web, cada um deles podendo ser cliente dos outros. Peltz [17] nomeia conversações entre múltiplos serviços Web como *coreografias* de serviços Web. Em uma coreografia, cada serviço desempenha o comportamento de um parceiro de negócios. A interação entre os serviços ocorre de forma colaborativa, a fim de executar um processo de negócios específico.

Coreografias de serviços Web podem ser descritas de várias formas [18], como por exemplo, por meio de máquinas de estado, diagramas de seqüência, diagramas de atividades ou linguagens específicas. As linguagens WS-CDL (*Web Services Choreography Description Language*) [27] e WSCI (*Web Service Choreography Interface*) [28] são exemplos de linguagens para descrição de coreografias. Essas linguagens apresentam duas vantagens em relação às outras abordagens: (i) são baseadas em XML e, portanto, diretamente processáveis por computador e (ii) aproveitam a abstração fornecida pela linguagem WSDL, ou seja, definem coreografias de serviços descritos em WSDL.

A descrição de uma coreografia deve expressar de forma precisa e padronizada, sob uma perspectiva global, as interações entre os serviços, as condições e restrições sob as quais as interações devem acontecer e detalhes a respeito das informações trocadas. Esta descrição pode ser usada como um acordo entre parceiros de negócio, permitindo que cada um desenvolva seu serviço Web de forma consistente com a coreografia. A implementação dos serviços Web que executam uma coreografia pode ser feita por linguagens de programação convencionais ou por linguagens de *orquestração* de serviços Web (veja Seção 2.2.2).

2.2.2 Orquestração

A *orquestração* de serviços Web permite compor um serviço por meio da combinação das funcionalidades de outros serviços. Muitos autores preferem o uso do termo *composição* de serviços Web. Neste trabalho, os dois termos serão usados de forma intercambiável.

Uma orquestração é um processo executável, ele próprio um serviço Web, que pode

interagir com outros serviços [17]. A interação com tais serviços é descrita no nível da troca de mensagens. O processo também inclui uma lógica de negócios e a ordem com que as interações acontecem. Diversas linguagens para orquestração de serviços Web vêm sendo propostas. Como exemplos, pode-se citar BPEL4WS (*Business Process Execution Languange for Web Services*) [29], BPML (*Business Process Modeling Language*) [30] e XPDL (*XML Process Definition Language*) [31].

2.2.3 WS-CDL

Esta seção apresenta uma visão geral da linguagem WS-CDL, que, neste trabalho, foi adotada como linguagem para representação de coreografias. Um documento WS-CDL consiste de um conjunto de elementos que definem, basicamente, os tipos de informação trocados durante a coreografia, os participantes envolvidos e as trocas de mensagens entre eles. Todas estas definições são encapsuladas por um elemento raiz, chamado `package`.

Os tipos de informação são definidos por elementos `informationType`, que referenciam tipos de mensagem definidos em documentos WSDL ou tipos/elementos definidos em esquemas XML. Outros elementos relacionados aos tipos de informação são `token` e `tokenLocator`. `Tokens` são elementos que definem um nome para uma propriedade específica representada por uma porção de uma informação contida em uma mensagem. São usados, por exemplo, no controle de contexto das mensagens. `TokenLocators` relacionam `tokens` com tipos de informação.

Os possíveis participantes de uma coreografia e seus relacionamentos são definidos pelos elementos `roleType`, `relationshipType`, `participantType` e `channelType`. Segue uma breve descrição de cada um destes elementos:

- `roleType` define um papel a ser desempenhado por um participante em uma coreografia e enumera `portTypes` WSDL que o participante deve apresentar;
- `relationshipType` indica que existe um relacionamento entre dois participantes de uma coreografia;
- `participantType` agrupa os papéis (`roleTypes`) que devem ser implementados por um participante em uma coreografia;
- `channelType` especifica características do canal de comunicação de um participante da coreografia, incluindo o *endpoint*, os dados utilizados no controle de contexto (`tokens`) e restrições ao uso do canal.

O Código Fonte 2.5 mostra um trecho WS-CDL especificando dois papéis, "Produtor-de-Café" (linhas 1 a 3) e "Fornecedor-de-Insumos" (linhas 5 a 7). O "Produtor-de-Café" deve implementar o `portType` "CompradorPortType", conforme especificado pelo

`behavior` "comprador" (linha 2). De forma análoga, o "Fornecedor-de-Insumos" deve implementar o `portType` "VendedorPortType". Nas linhas 9 a 12, é definido um `relationshipType`, chamado "Produtor-Fornecedor", envolvendo os dois papéis. O `channelType` "Canal-de-Compra" (linhas 14 a 24) define um canal de comunicação para interação com o "Fornecedor-de-Insumos". Este canal pode ser usado para operações do tipo *request-response*, conforme especificado pelo atributo `action` (linha 15). O elemento `reference` (linhas 18 a 20) referencia um `token` que especifica o *endpoint* do "Fornecedor-de-Insumos". Já o elemento `identity` (linhas 21 a 23) referencia um `token` que especifica os dados utilizados no controle do contexto. Neste exemplo, cada participante desempenha um único papel, portanto, não foi necessário o uso do elemento `participantType`.

Código Fonte 2.5: Exemplo de definição de participantes.

```
1  <roleType name="Produtor-de-Café">
2     <behavior name="comprador" interface="CompradorPortType"/>
3  </roleType>
4
5  <roleType name="Fornecedor-de-Insumos">
6     <behavior name="vendedor" interface="VendedorPortType"/>
7  </roleType>
8
9  <relationshipType name="Produtor-Fornecedor">
10    <roleType typeRef="Produtor-de-Café" behavior="comprador"/>
11    <roleType typeRef="Fornecedor-de-Insumos" behavior="vendedor"/>
12 </relationshipType>
13
14 <channelType  name="Canal-de-Compra"
15               action="request-respond" ...>
16    ...
17    <roleType typeRef="Fornecedor-de-Insumos"  behavior="vendedor"/>
18    <reference>
19       <token name="EndpointToken"/>
20    </reference>
21    <identity ...
22       <token name="ContextToken"/>
23    <identity>
24 </channelType>
```

As trocas de mensagens são suportadas por variáveis de dados, atividades de controle de fluxo e atividades básicas. Todos estes elementos são definidos dentro de elementos `choreography`. As variáveis de dados armazenam a informação trocada durante a coreografia. Elas são definidas por elementos `variable`, sendo declaradas no início de cada coreografia.

As atividades de controle de fluxo são de quatro tipos:

- `sequence`: permite a execução seqüencial de um conjunto de atividades, na ordem em que são definidas;
- `parallel`: permite a execução paralela de um conjunto de atividades;
- `choice`: especifica a existência de escolha entre duas ou mais atividades para que uma seja executada;
- `workunit`: permite controle condicional e repetitivo de um conjunto de atividades.

A principal atividade básica é a interação entre participantes. Esta atividade é especificada pelo elemento `interaction`. O Código Fonte 2.6 ilustra uma interação entre o "Produtor-de-Café" e o "Fornecedor-de-Insumos". Os atributos `channelVariable` e `operation` (linhas 2 e 3, respectivamente) especificam o canal utilizado na interação ("Canal-de-Compra-Var") e a operação a que a interação se refere ("Compra"). O elemento `participate` (linha 4 a 6) define os participantes envolvidos. O elemento `exchange` (linhas 7 a 12) indica uma troca de mensagens do tipo "Pedido-Compra", conforme especificado pelo atributo `informationType` (linha 8). O atributo `action` (linha 9) indica que a troca de mensagem se refere à requisição da operação. As variáveis "Var1" e "Var2", definidas pelos elementos `send` e `receive` (linhas 10 e 11), armazenam a informação trocada dos lados do emissor e do receptor da mensagem, respectivamente.

Código Fonte 2.6: Exemplo de interação entre participantes.

```
1  <interaction  name="Interação-Compra"
2                channelVariable="Canal-de-Compra-Var"
3                operation="Compra" ...>
4     <participate  relationshipType="Produtor-Insumos"
5                   fromRoleTypeRef="Produtor-de-Café"
6                   toRoleTypeRef="Fornecedor-de-Insumos" />
7     <exchange  name="Requisição"
8                informationType="Pedido-de-Compra"
9                action="request" ...>
10      <send      variable="Var1" .../>
11      <receive   variable="Var2" .../>
12    </exchange>
13    ...
14 </interaction>
```

Outras atividades básicas são:

- `assign`: permite criar e modificar o valor de uma variável;
- `perform`: permite executar uma coreografia aninhada;

- `silentAction`: sinaliza a existência de uma atividade, pertinente a determinado participante, mas que não interessa aos demais;
- `noAction`: sinaliza um ponto da coreografia no qual determinado participante não desempenha atividade alguma;
- `finalize`: finaliza uma coreografia.

Detalhes da linguagem foram omitidos em busca de simplificar a explicação. A especificação completa pode ser encontrada em [27].

2.2.4 BPEL

Esta seção introduz a linguagem BPEL, que, neste trabalho, foi adotada como linguagem de composição de serviços Web. Assim como WS-CDL, BPEL permite especificar colaborações entre serviços Web baseando-se nos tipos de informações trocadas, nos parceiros de negócio e nas trocas de mensagens que ocorrem entre eles. A diferença chave entre as duas linguagens é que, diferente de WS-CDL que especifica as interações sob uma perspectiva global, BPEL apresenta a perspectiva de um único participante, o executor do processo.

Em BPEL, os tipos de informações são especificados usando WSDL ou esquemas XML. A especificação da linguagem define extensões para WSDL, que equivalem aos elementos `token` e `tokenLocator` da linguagem WS-CDL, os elementos `property` e `propertyAlias`. O primeiro cria um nome globalmente único utilizado para referenciar os campos de uma mensangem pertinentes ao controle de contexto. O último relaciona um `property` com os campos das mensagens. O Código Fonte 2.7 ilustra a declaração de um elemento `property`. Neste exemplo, é declarada uma propriedade com nome "Propriedade-NotaFiscal" (linha 1). Esta propriedade possui um `propertyAlias` (linhas 3 a 6) que a relaciona com o campo `/NotaFiscal/Número` da mensagem "Pedido".

Código Fonte 2.7: Exemplo de definição de definição de propriedades em BPEL.

```
1 <property name="Propriedade-NotaFiscal" type="xsd:string"/>
2
3 <propertyAlias propertyName="Propriedade-NotaFiscal"
4    messageType="Pedido"
5    part="NotaFiscal"
6    query="/NotaFiscal/Número"/>
```

Os parceiros de negócio são definidos por elementos `partnerLinkType` e `partnerLink`. `PartnerLinkTypes` estendem WSDL e são usados para denotar a ligação entre o executor do processo e um parceiro de negócios. Um `partnerLinkType` relaciona os `portTypes`

que cada participante desta ligação deve apresentar. `PartnerLinks`, por sua vez, são instâncias de `partnerLinkTypes`, ou seja, o executor do processo pode participar de mais de uma ligação do mesmo tipo, cada qual com um parceiro diferente. Dependendo da implementação da máquina de execução BPEL utilizada, `partnerLinks` são associados aos *endpoints* dos parceiros de formas diferentes.

BPEL apresenta cinco tipos de atividade de controle de fluxo:

- `sequence`: permite a execução seqüencial de um conjunto de atividades, na ordem em que são definidas;
- `flow`: permite a execução paralela de um conjunto de atividades;
- `switch`: permite controle condicional de atividades;
- `while`: permite execução repetitiva de um conjunto de atividades;
- `pick`: aguarda a ocorrência de um evento, que pode ser o recebimento de uma mensagem ou alarme de tempo.

As atividades para envio e recebimento de mensagens são `invoke`, `receive` e `reply`. `Invoke` invoca operações de serviços Web. `Receive` suspende a execução do processo e aguarda o recebimento de uma mensagem. `Reply` é utilizado para responder a uma mensagem recebida, provendo o comportamento de uma operação síncrona. Qualquer destas três atividades pode referenciar elementos `correlationSet`, que definem as propriedades (elementos `property`) que devem ser consideradas no controle do contexto durante as trocas de mensagens. O Código Fonte 2.8 mostra um exemplo de invocação de operação. O elemento `correlationSets` (linhas 1 a 4) define o `correlationSet` referenciado pela atividade de invocação (linhas 6 a 14). O atributo `partnerLink` identifica o parceiro ao qual será feita a invocação da operação. Os atributo `portType` e `operation` (linhas 7 e 8) indicam a operação invocada. Os atributos `inputVariable` e `outputVariable` (linhas 9 e 10) referenciam as variáveis que armazenam os dados de entrada e de saída, respectivamente.

Código Fonte 2.8: Exemplo de invocação de operação em BPEL.

```
1 <correlationSets>
2    <correlationSet name="CorrelationSet-Compra"
3                    properties="Propriedade-NotaFiscal"/>
4 </correlationSets>
5 ...
6 <invoke partnerLink="Fornecedor"
7         portType="VendedorPortType"
8         operation="Compra"
```

```
9          inputVariable="Var1"
10         outputVariable="Var4">
11    <correlations>
12       <correlation set="CorrelationSet-Compra" ...>
13    </correlations>
14 </invoke>
```

O Código Fonte 2.9 apresenta o código para recebimento da invocação e emissão da resposta, referentes à invocação ilustrada no Código Fonte 2.8. Note que a sintaxe do elementos `receive` e `reply` são semelhantes à do elemento `invoke`, exceto pelo fato de especificarem uma única variável (de entrada, no caso do `receive`, e de saída, no caso do `reply`).

Código Fonte 2.9: Exemplo de recebimento de mensagem em BPEL.

```
1  <receive partnerLink="Produtor"
2          portType="VendedorPortType"
3          operation="Compra"
4          variable="Var2">
5     <correlations>
6        <correlation set="CorrelationSet-Compra" ...>
7     </correlations>
8  </receive>
9  ...
10 <reply partnerLink="Produtor"
11         portType="VendedorPortType"
12         operation="Compra"
13         variable="Var3">
14    <correlations>
15       <correlation set="CorrelationSet-Compra" ...>
16    </correlations>
17 </reply>
```

A manipulação de dados é realizada por meio dos elementos `variable` e `assign`, assim como na linguagem WS-CDL. Além disso, BPEL apresenta um elemento `empty`, que representa uma atividade nula. Detalhes e alguns elementos da linguagem foram omitidos com o intuito de apresentar o que é essencial para o entendimento do restante do texto. Conhecimento aprofundado pode ser adquirido por meio da especificação da linguagem [29].

2.3 Cadeias Produtivas

Uma cadeia produtiva pode ser definida como o conjunto de atividades envolvidas na distribuição de um produto, da matéria-prima ao consumidor final. Inclui fornecimento de

matéria-prima, manufatura, armazenamento, distribuição, transporte e todos os sistemas de informação necessários para monitorar e gerenciar estas atividades [15].

Beamon [13] agrupa as atividades de uma cadeia produtiva em dois processos básicos: (i) processo de planejamento de produção e controle de estoque e (ii) processo de distribuição e logística. O primeiro abrange os sub-processos de fabricação e de armazenamento. O planejamento de produção inclui a aquisição de matéria-prima, o projeto e o agendamento de processos de fabricação e gerência de materiais. O controle de estoque descreve o projeto e a gerência de políticas e procedimentos de armazenamento de matéria-prima e de produtos finais. O processo de distribuição e logística determina como os produtos são transportados de armazéns para varejistas. Inclui a gerência de estoque, transporte e distribuição do produto final. A Figura 2.2 mostra o diagrama de uma cadeia produtiva genérica e destaca os integrantes da cadeia produtiva que participam de cada processo básico. As setas indicam o fluxo de produtos e matérias-primas.

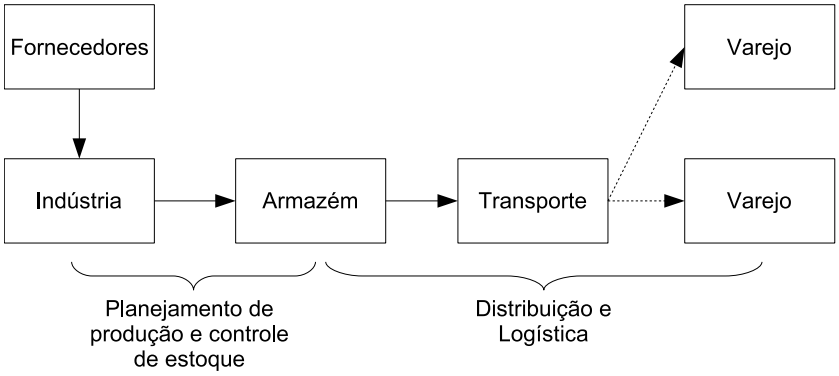

Figura 2.2: Processos básicos da cadeia produtiva. Adaptado de [13].

A Figura 2.3 mostra um exemplo simplificado da cadeia produtiva do café. O "Fornecedor de Insumos" fornece insumos para o "Produtor de Café". O café produzido é transportado para a "Cooperativa", que encaminha o produto para o "Torrefador". O café torrado é levado para o "Varejo", de onde finalmente alcança o consumidor final.

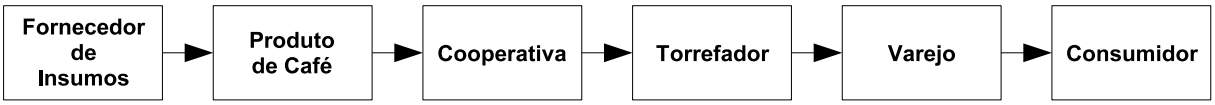

Figura 2.3: Exemplo simplificado da cadeia produtiva do café.

2.3.1 Integração de Cadeias Produtivas

Os integrantes de uma cadeia produtiva podem trabalhar de forma integrada, usando técnicas de gerência, a fim de otimizar o desempenho coletivo na criação, distribuição e

suporte ao produto final [16]. A gerência de cadeias produtivas agrupa conjuntos de abordagens utilizadas para que mercadorias sejam produzidas e distribuídas nas quantidades corretas, para os lugares corretos, com o objetivo de diminuir custos e satisfazer o nível de serviço requerido [32].

A gerência de cadeias produtivas é complexa. Os integrantes da cadeia podem apresentar objetivos conflitantes, dificultando a definição de estratégias para uma organização em particular. Outra dificuldade está relacionada à dinamicidade da cadeia. A demanda do consumidor, a capacidade de produção dos fornecedores e os relacionamentos da cadeia evoluem com o tempo [32].

Apesar do custo gerado pela gerência da cadeia produtiva, parcerias estratégicas entre os integrantes da cadeia podem ter impacto significativo no desempenho da mesma. Nos últimos anos, tornou-se claro que as indústrias diminuíram os custos de produção tanto quanto possível. Muitas destas companhias têm descoberto que a gerência efetiva da cadeia produtiva é o próximo passo para aumentar o lucro e o mercado [32]. Dentre os benefícios alcançados pela gerência da cadeia produtiva, pode-se destacar a economia gerada pela redução de estoques, resultado do aumento da velocidade com que os produtos movem-se pela cadeia. Além disso, a integração permite a redução de redundância de fornecedores. Como conseqüência, o aumento da produção dos fornecedores restantes e a diminuição dos custos da gerência da cadeia contribuem para reduzir os custos do produto [16].

2.3.2 Arquitetura para Integração de Cadeias Produtivas Agropecuárias

A infra-estrutura proposta utiliza a arquitetura para integração de cadeias produtivas agropecuárias definida por Bacarin [14] como arcabouço. Esta arquitetura se baseia em um modelo composto por quatro elementos básicos:

- Elementos de produção: encapsulam processos produtivos que geram produtos que fluem pela cadeia produtiva;
- Elementos de armazenamento: armazenam produtos e matéria-prima;
- Elementos de transporte: são responsáveis por mover produtos e matéria-prima entre elementos de produção e armazenamento;
- Atores: são agentes de software ou humanos que podem estar direta ou indiretamente envolvidos na execução de atividades da cadeia.

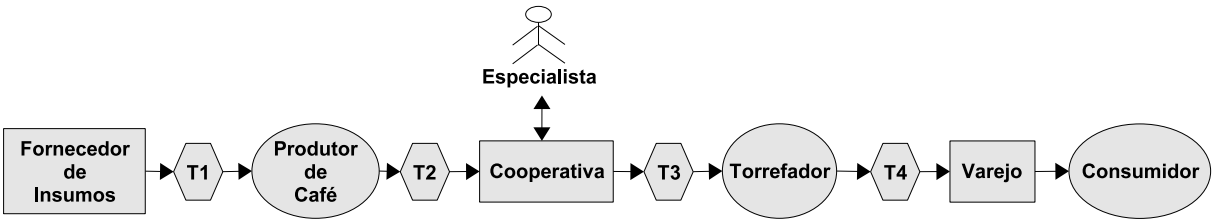

Figura 2.4: Trecho da cadeia produtiva do café representado pelo modelo de Bacarin.

A Figura 2.4 apresenta a cadeia produtiva do café ilustrada na Figura 2.3 representada graficamente pelo modelo proposto por Bacarin. Elementos de produção são representados por elipses, elementos de armazenamento por retângulos, elementos de transporte por hexágonos e atores por bonecos. Note que no modelo de Bacarin, elementos de transporte são explicitamente representados. Para exemplificar o uso de atores, um ator foi adicionado à cooperativa. Este ator poderia representar, por exemplo, um especialista que separa o café em função de suas diferentes qualidades.

Cada elemento básico do modelo contém um conjunto de componentes da arquitetura responsáveis pela sua gerência. São eles: gerente de coordenação, gerente de negociação, gerente de sumários e gerente de regulamentos. O gerente de coordenação é responsável por disparar e coordenar todos os processos da cadeia produtiva. O comportamento do gerente de coordenação é descrito por planos de coordenação. Tal plano é composto por um conjunto de instruções que disparam as atividades da cadeia por meio de requisições a serviços Web. O gerente de coordenação é capaz de interpretar e executar estas instruções. A definição de uma arquitetura para o gerente de coordenação e sua integração com o modelo de coordenação proposto é um dos principais objetivos deste trabalho.

O gerente de negociação é responsável pela negociação de contratos entre participantes da cadeia produtiva. Tais contratos representam acordos entre parceiros de negócio e descrevem suas obrigações e autorizações mútuas. Definem qualidade, forma de distribuição e custos. O gerente de sumários controla a rastreabilidade de produtos da cadeia [33]. É responsável por manter sumários dos eventos ocorridos ao longo da cadeia e permite a realização de consultas a respeito da origem, destino ou localização dos produtos. O gerente de regulamentos tem o objetivo de assegurar que os regulamentos da cadeia são mantidos. Os regulamentos são conjuntos de regras que especificam restrições impostas aos diferentes estágios de execução da cadeia produtiva de forma a regular a evolução do produto ao longo da mesma. Alguns exemplos de regulamentos são: critérios de qualidade, regulamentos governamentais e condições determinadas por aspectos sociais, culturais, econômicos e até mesmo religiosos de uma região.

Capítulo 3

Modelo de Coordenação

Este capítulo apresenta o modelo proposto para coordenação de atividades em cadeias produtivas. Este modelo foi concebido com dois objetivos principais, facilitar o projeto e a implantação dos processos de negócio interorganizacionais da cadeia e permitir a sua execução.

O modelo de coordenação divide-se em três camadas (Figura 3.1). A camada superior corresponde à modelagem do processo de negócio interorganizacional. A segunda trata da modelagem do comportamento individual de cada participante envolvido neste processo. A terceira camada trata da coordenação necessária para que os participantes possam executar o processo de negócio, incluindo o estabelecimento da conexão entre os participantes e o encaminhamento de mensagens entre eles. As três camadas são detalhadas nas Seções 3.1, 3.2 e 3.3.

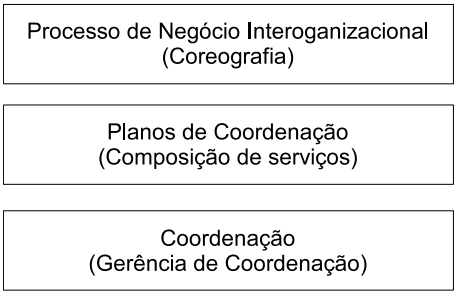

Figura 3.1: Camadas do modelo de coordenação.

3.1 Processos de Negócio Interorganizacionais

A primeira camada do modelo de coordenação está relacionada ao projeto do processo de negócio interorganizacional. Esta camada visa estabelecer o relacionamento interorga-

nizacional dos participantes da cadeia produtiva. Conforme apresentado na Seção 2.2.1, coreografias permitem especificar as interações entre diversos serviços Web, que colaboram entre si, a fim de alcançar um determinado objetivo. Neste trabalho, coreografias especificadas na linguagem WS-CDL são usadas para descrever as interações interorganizacionais entre os participantes da cadeia.

A vantagem de se especificar os processos de negócio como coreografias WS-CDL é que, por ser baseada em XML, a descrição pode ser processada por computador, permitindo a derivação automática dos esqueletos dos planos de coordenação (veja Seção 3.2).

3.2 Planos de Coordenação

A segunda camada do modelo trata do comportamento individual que cada participante da cadeia produtiva deve desempenhar no processo de negócio. O comportamento de um participante é descrito por um plano de coordenação. Um plano de coordenação é um conjunto de instruções que podem ser interpretadas e executadas por um gerente de coordenação. Existem três tipos de instruções: instruções de requisição, instruções de entrada e instruções de controle de fluxo.

As instruções de requisição disparam atividades da cadeia. Instruções de entrada suspendem a execução do gerente de coordenação e aguardam uma requisição proveniente de outro gerente de coordenação. Instruções de controle de fluxo gerenciam o fluxo de execução do plano de coordenação. Exemplos de intruções de controle de fluxo são instruções condicionais e de repetição.

O conceito do plano de coordenação pode ser diretamente mapeado para linguagens de composição de serviços Web. Neste caso, as atividades da cadeia são encapsuladas por serviços Web. As instruções de requisição são mapeadas para instruções de invocação de serviços e as instruções de entrada são mapeadas para instruções de recebimento de mensagens.

Para a validação do modelo, decidiu-se utilizar a linguagem BPEL para representação de planos de coordenação. Dois motivos levaram a esta decisão: (i) BPEL vem sendo amplamente adotado pela literatura como padrão para composição de serviços Web e (ii) apresenta uma variedade de máquinas de execução e ferramentas de suporte. É importante notar que a implementação do modelo não se atém à linguagem de composição utilizada. Um plano descrito na linguagem BPEL poderia ser mapeado para outra linguagem, adequada a uma implementação específica do gerente de coordenação.

As regras e restrições fornecidas pela descrição da coreografia WS-CDL possibilitam que uma organização projete um plano de coordenação que reflita seu comportamento no processo de negócio e seja interoperável com qualquer outro plano gerado a partir da mesma descrição. Na verdade, a descrição da coreografia não fornece informação sufi-

ciente para geração de planos de coordenação completos. Ela trata apenas das interações interorganizacionais, ou seja, não contém a lógica interna de cada organização. Ao plano incompleto, gerado a partir das informações contidas em uma descrição de coreografia, dá-se o nome de *esqueleto de plano de coordenação*. A Figura 3.2 ilustra a relação conceitual entre uma descrição de coreografia e um esqueleto de plano de coordenação. Enquanto a descrição da coreografia abrange todas as interações, entre todos os envolvidos, um esqueleto de plano de coordenação abrange apenas a porção das interações relacionadas a um único participante.

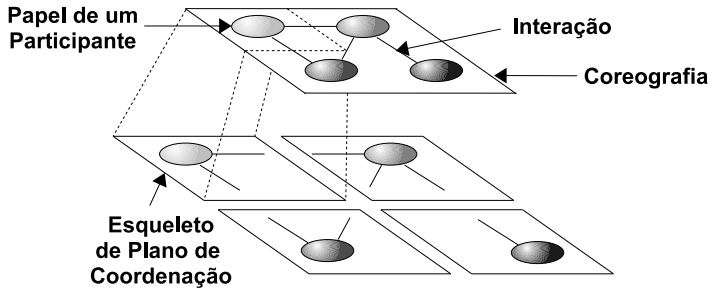

Figura 3.2: Relação entre a coreografia e esqueletos de planos de coordenação.

Qualquer participante da cadeia que tenha interesse em participar de uma coreografia, desempenhando um papel específico, pode utilizar o esqueleto adequado para gerar seu plano de coordenação, o qual será executado pelo seu gerente de coordenação. A geração dos planos de coordenação é feita por um programador que insere, em um esqueleto, as instruções necessárias para adicionar a lógica interna do participante da cadeia. A lógica interna consiste de chamadas a aplicações de *software* internas ou disparo de *workflows* internos que fornecem dados para as requisições externas ou que realizam atividades requisitadas por outros participantes de coreografia. A Figura 3.3 mostra a metodologia empregada para geração dos planos de coordenação.

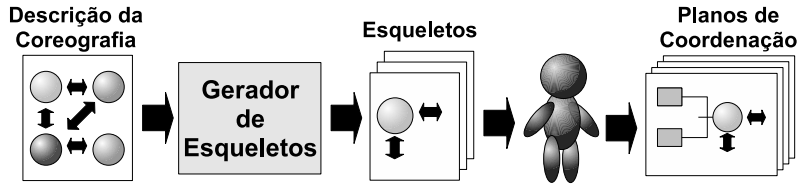

Figura 3.3: Metodologia para geração de planos de coordenação

Para ilustrar a geração de planos de coordenação, considere o exemplo de processo de negócio hipotético ilustrado na parte superior da Figura 3.4. Neste processo de negócio, um produtor de café "P" requisita a intermediação de uma cooperativa "C" na aquisição de insumos (mensagens M1 e M4). "C" encomenda os insumos (mensagens M2 e M3) a um fornecedor "F" enquanto "P" contrata para um transportador "T" (mensagens M5 e M6) para transportar os produtos. Para realizar o transporte de produto, "T" requisita os dados da entraga para "F" (mensagens M7 e M8). A parte inferior da Figura 3.4 mostra a aplicação do modelo de coordenação. Em um primeiro momento, o processo de negócio é modelado como uma coreografia WS-CDL. Em seguida, gera-se os planos de coordenação correspondentes a cada papel do processo (P, C, F e T). Cada plano pode ser executado pelo gerente de coordenação de um participante da cadeia.

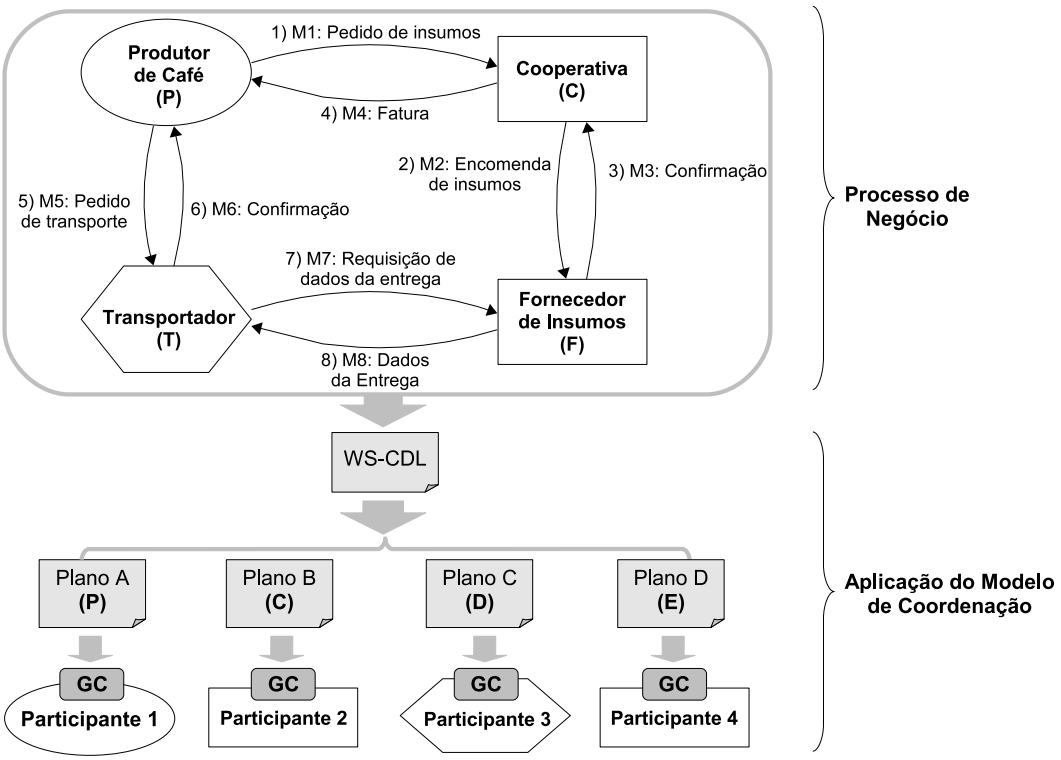

Figura 3.4: Cenário para exemplo de geração de plano de coordenação.

O Código Fonte 3.1 apresenta a descrição da coreografia, expressa em pseudo WS-CDL, para o cenário da Figura 3.4. As interações da descrição da coreografia que aparecem nas linhas 16, 17, 21 e 22 correspondem às interações nas quais o transportador participa. São estas instruções que compõem o seu esqueleto de plano de coordenação (expresso em pseudo BPEL) ilustrado no Código Fonte 3.2. Qualquer transportador que tenha interesse

em participar da coreografia de fornecimento ao produtor de café pode usar este esqueleto de plano de coordenação para gerar seu plano de coordenação específico.

Código Fonte 3.1: Exemplo de descrição de coreografia, descrito em pseudo WS-CDL, referente ao cenário da Figura 3.4

```
1  interaction(pedido-insumos)     %Interação referente a
2                                  %operação "pedido-insumos"
3     exchange(P, M1, C, M1)        %P envia M1, C recebe M1
4  end_interaction
5  interaction(encomenda-insumos) %Interação referente a
6                                  %operação "encomenda-insumos"
7     exchange(C, M2, F, M2)       %C envia M2, F recebe M2
8     exchange(F, M3, C, M3)       %F envia M3, C recebe M3
9  end_interaction
10 interaction(pedido-insumos)     %Interação referente a
11                                 %operação "pedido-insumos"
12    exchange(C, M4, P, M4)       %C envia M4, P recebe M4
13 end_interaction
14 interaction(trasporte)          %Interação referente a
15                                 %operação "transporte"
16    exchange(P, M5, T, m5)       %P envia M5, T recebe M5
17    exchange(T, M6, P, m6)       %T envia M6, P recebe M6
18 end_interaction
19 interaction(dados-entrega)      %Interação referente a
20                                 %operação "dados-entrega"
21    exchange(T, M7, F, M7)       %T envia M7, F recebe M7
22    exchange(F, M8, T, M8)       %F envia M8, T recebe M8
23 end_interaction
```

Código Fonte 3.2: Exemplo de esqueleto de plano de coordenação para "T", descrito em pseudo BPEL, referente ao cenário da Figura 3.4

```
1  receive(P, transporte, M5)     %recebe M5 de P
2                                 %referente a operação "transporte"
3  reply(P, transporte, M6)       %responde M6 para P
4                                 %referente a operação "transporte"
5  invoke(F, transporte, M7, M8) %envia M7 para F e recebe M8 de F
6                                 %referente a operação "dados-entrega"
```

O Código Fonte 3.3 mostra um plano de coordenação, expresso em pseudo BPEL, gerado a partir do esqueleto de plano de coordenação do transportador. As instruções que aparecem nas linhas 3 e 4 referem-se a lógica interna de um forncedor específico.

Código Fonte 3.3: Exemplo de plano de coordenação para "T", descrito em pseudo BPEL, referente ao cenário da Figura 3.4

```
1 receive(P, transporte, M5)     %recebe M5 de P
2                                %referente a operação "transporte"
3 invoke(...)                    %dispara atividade interna
4 invoke(...)                    %dispara atividade interna
5 reply(P, transporte, M6)       %responde M6 para P
6                                %referente a operação "transporte"
7 invoke(F, transporte, M7, M8) %envia M7 para F e recebe M8 de F
8                                %referente a operação "dados-entrega"
```

A metodologia para geração de planos de coordenação é uma das contribuições deste trabalho. A linguagem WS-CDL define de forma clara e formal as interações entre os participantes de um processo de negócio. Isto pemite a derivação automática dos esqueletos, minimizando erros e inconsistências. Outra vantagem da automatização é o aumento da eficiência da geração dos planos e conseqüentemente da implantação do processo de negócio. A Seção 4.3 apresenta a especificação de um gerador automático de esqueletos de planos.

3.3 Coordenação

A terceira camada provê um nível de coordenação básica que suporta as interações entre os participantes da cadeia produtiva. Esta coordenação é implementada pelo gerente de coordenação e é transparente para as outras camadas. O gerente de coordenação interpreta e executa planos de coordenação, sendo responsável por disparar e coordenar os processos de negócio da cadeia produtiva. Ele representa a interface de um participante da cadeia produtiva, na interação com outros participantes, em processos de negócio interorganizacionais. Isto envolve:

- o estabelecimento de conexões com gerentes de coordenação pertencentes a outros participantes da cadeia produtiva, possibilitando a execução de processos de negócio interorganizacionais;
- a troca de mensagens de negócio com gerentes de coordenação de outros participantes. Exemplos de tais mensagens são: pedidos de suprimento e de serviços, envio de faturas, etc;
- a coordenação das atividades que caracterizam a lógica interna de um participante durante sua atuação em um processo de negócio. Estas atividades disparam

aplicações e *workflows* internos e ações de outros gerentes, incluindo: gerentes de negociação, gerentes de sumários, gerentes de regulamentos e gerentes de coordenação de outros níveis hierárquicos.

A Figura 3.5 esquematiza a função de um gerente de coordenação. Todas as suas interações são realizadas via serviços Web. Portanto, *workflows* e sistemas internos necessitam de uma interface de serviços Web.

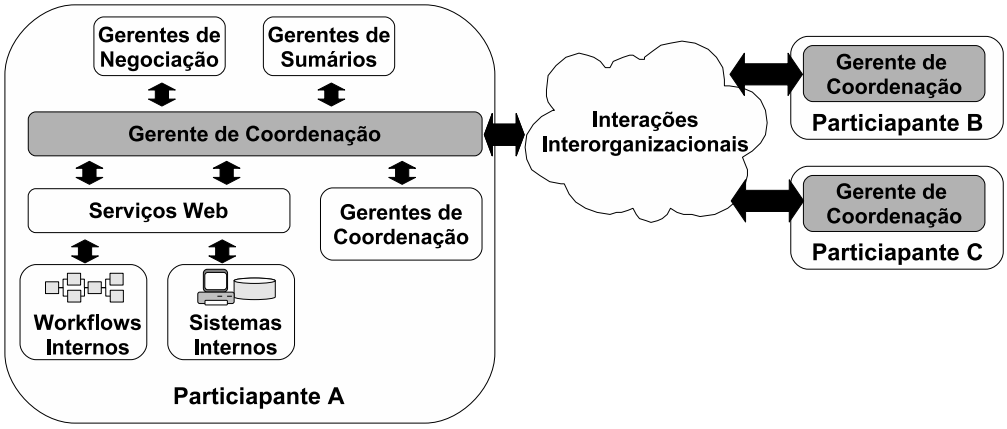

Figura 3.5: Função do gerente de coordenação.

3.3.1 Conexão entre Participantes de Cadeia Produtiva

Interações entre dois participantes da cadeia produtiva são precedidas por uma fase de conexão. Nesta fase, os gerentes de coordenação de cada participante acordam sobre a coreografia a ser seguida, o papel que deve ser desempenhado por cada um e o identificador de contexto que distingue a instância da coreografia. Uma conexão é disparada durante a execução da coreografia, por um gerente de coordenação, toda vez que uma mensagem de aplicação necessite ser enviada para outro gerente de coordenação, que não está conectado.

Para que uma conexão entre dois gerentes de coordenação seja estabelecida, o gerente que inicia a conexão deve conheçer o *endpoint* do outro. Existem duas formas para que o *endpoint* seja obtido. Na primeira, o *endpoint* do gerente de coordenação destino é definido de forma estática antes da execução do plano de coordenação. Na segunda, o *endpoint* é descoberto dinamicamente, por meio de um *repositório de participantes*. Todo gerente de coordenação, ao iniciar a execução de um plano de coordenação, se registra no repositótio de participantes. Este repositório funciona como um servidor de nomes e permite que outros gerentes de coordenação descubram, em tempo de execução, o *endpoint*

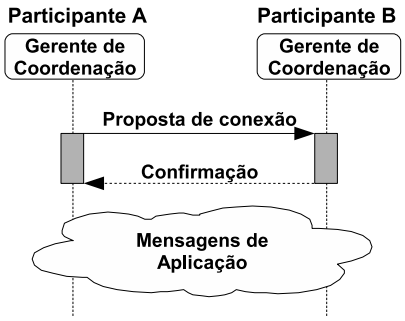

Figura 3.6: Diagrama de seqüência ilustrando a forma estática de conexão entre dois participantes da cadeia.

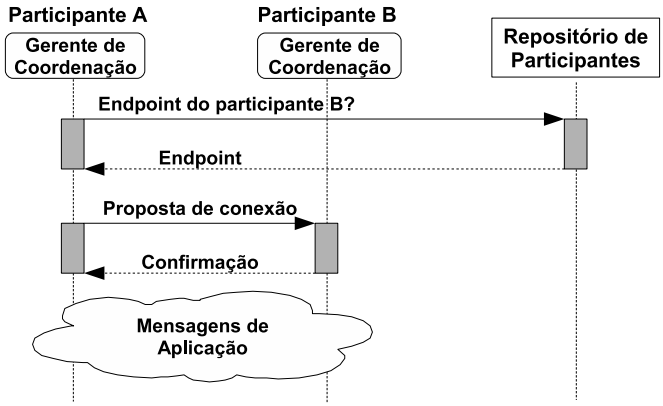

Figura 3.7: Diagrama de seqüência ilustrando a forma dinâmica de conexão entre dois participantes da cadeia.

daquele que está desempenhando um papel específico em uma instância de coreografia ou descubram automaticamente o *endpoint* de um gerente que pode desempenhar tal papel.

Depois de determinar o *endpoint* destino, o gerente de coordenação envia uma proposta de conexão e aguarda a sua confirmação. Ao receber a confirmação, a mensagem de aplicação que disparou a conexão é enviada. Se a conexão for recusada, uma exceção interna é gerada. Uma proposta de conexão pode ser recusada por três motivos. Primeiro, o gerente de coordenação destino pode não ser capaz de executar o plano de coordenação apropriado para desempenhar o papel esperado pelo proponente. Segundo, uma conexão idêntica pode já estar ativa. Terceiro, a proposta de conexão pode violar restrições de autenticação, autorização ou segurança. O diagrama de seqüência da Figura 3.6 ilustra uma conexão no qual o *endpoint* do participante "B" é conhecido estaticamente pelo participante "A". O diagrama de seqüência da Figura 3.7 mostra o caso em que o *endpoint* de "B" é obtido por meio do repositório de participantes.

A forma de conexão via repositório de participantes permite a formação dinâmica de

diferentes configurações de participantes da cadeia produtiva. Para exemplificar, suponha que, no cenário da Figura 3.4, o produtor de café ("P") tenha que descobrir em tempo de execução um parceiro que possa realizar o transporte dos insumos. A execução deste cenário é descrita no diagrama de seqüência ilustrado na Figura 3.8. Entre as mensagens 1 e 11, "P", "C" e "F" conectam-se (forma estática de conexão) e trocam mensagens referentes à encomenda de insumos. Em seguida, "P" realiza uma busca por transportadores no repositório de participantes (12). O repositório, por sua vez, retorna uma lista de participantes que podem desempenhar o papel "T" (13). Na seqüência, "P" escolhe um transportador, se conecta e realiza a contratação do serviço de transporte (14-18). "T", que não conhece o *endpoint* do participante que desempenha o papel de "F", faz uma consulta ao repositório de participantes (19-20). Após obter o *endpoint*, "T" se conecta a "F" e troca mensagens referentes à requisição de dados de entrega (23-24).

Note que diferentes configurações de participantes podem se formar neste cenário, dependendo do critério utilizado por "P" para determinar o participante que desempenha o papel "T". Isto só é possível porque o modelo de coordenação permite a ligação dinâmica entre os parceiros de negócio. Os planos de coordenação são projetados no nível dos papéis a serem desempenhados. A ligação entre os papéis e os participantes da cadeia é feito em tempo de execução, sendo transparente para o projetista dos planos de coordenação. A capacidade de prover a ligação dinâmica entre os parceiros de negócio é uma das contribuições do modelo proposto.

3.3.2 Encaminhamento de Mensagens

Todas as mensagens de aplicação carregam um cabeçalho com dados que são usados para o seu encaminhamento e para o controle de contexto. Estes dados incluem identificadores únicos para: o processo de negócio (BPID - *Business Process ID*), a instância do processo de negócio (CID - *Context ID*), o papel do emissor da mensagem (FRID - *From Role ID*) e o papel do receptor da mensagem (TRID - *To Role ID*).

A combinação destes dados identifica unicamente uma conexão com outro gerente de coordenação, tornando possível o encaminhamento das mensagens emitidas para seus destinos e a entrega das mensagens recebidas para as instâncias corretas do plano de coordenação. Uma vantagem desta abordagem é que utiliza identificadores padronizados, que não fazem parte da carga útil da mensagem, no controle de contexto, tornando-o transparente para o projetista do plano de coordenação. A separação entre a lógica necessária para o controle de contexto e a lógica de aplicação dos processos de negócio facilita o projeto das coreografias correspondentes.

A Figura 3.9 exemplifica o encaminhamento de mensagens. Neste exemplo, o gerente de coordenação "CMA", encaminha uma mensagem de aplicação referente à instância "1"

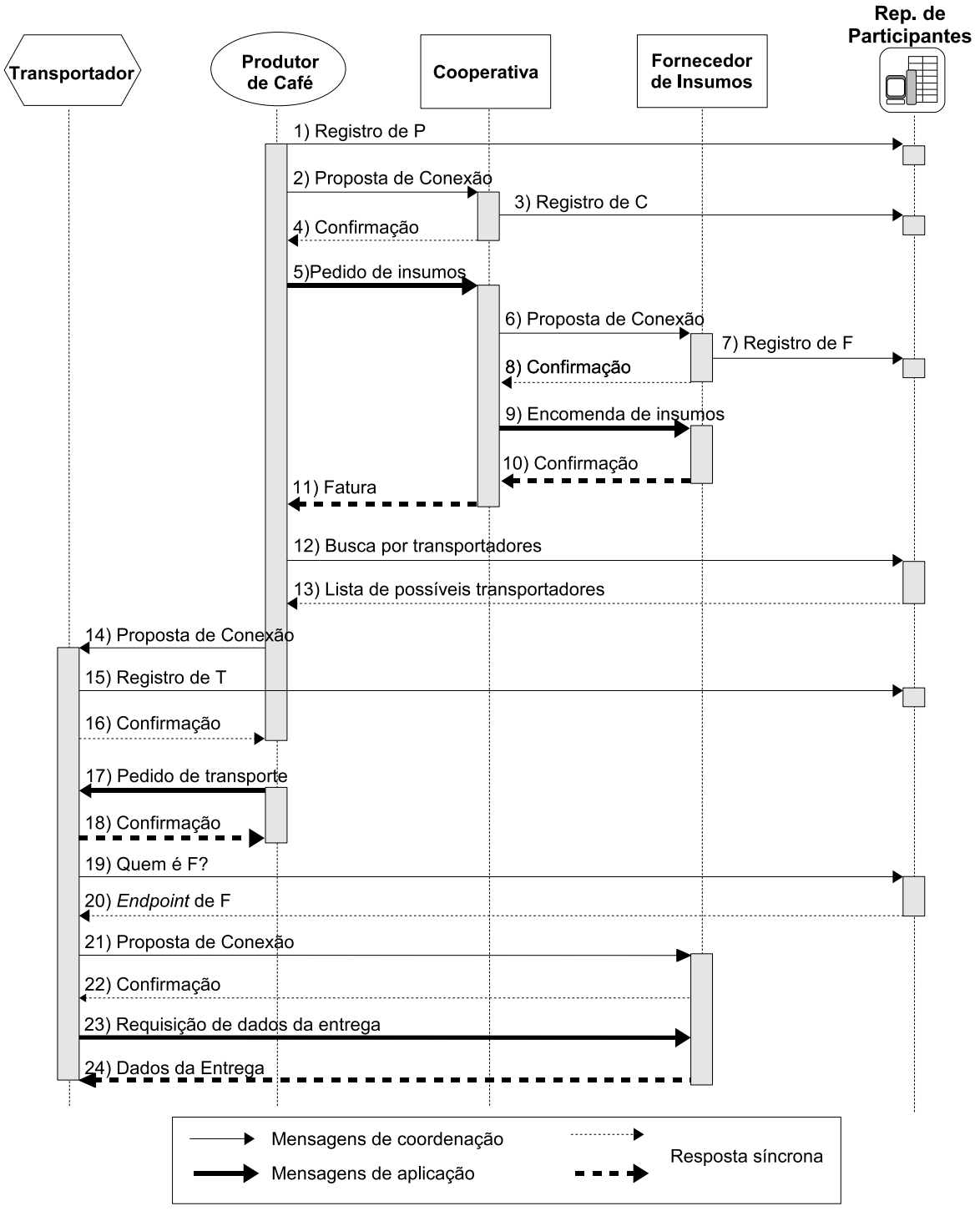

Figura 3.8: Diagrama de seqüência para cenário da Figura 3.4.

do processo de negócio “P1” para o gerente de coordenação “CMB”. “CMB”, por sua vez, entrega a mensagem para a instância “1” do plano de coordenação referente ao papel “B”.

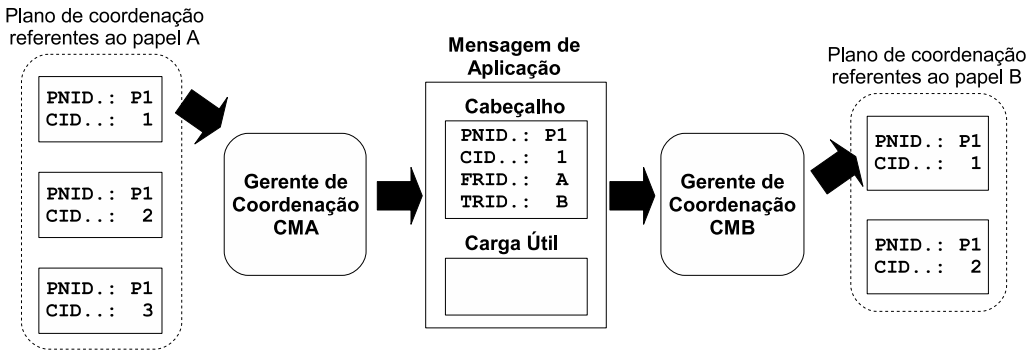

Figura 3.9: Esquema do encaminhamento de mensagens.

Capítulo 4

Infra-estrutura para o Modelo de Coordenação

Este capítulo apresenta a infra-estrutura que suporta o modelo de coordenação. Ela é composta por quatro elementos, o gerente de coordenação, o gerador de esqueletos, o repositório de participantes e o repositório de coreografias. O esquema da infra-estrutura é ilustrado na Figura 4.1.

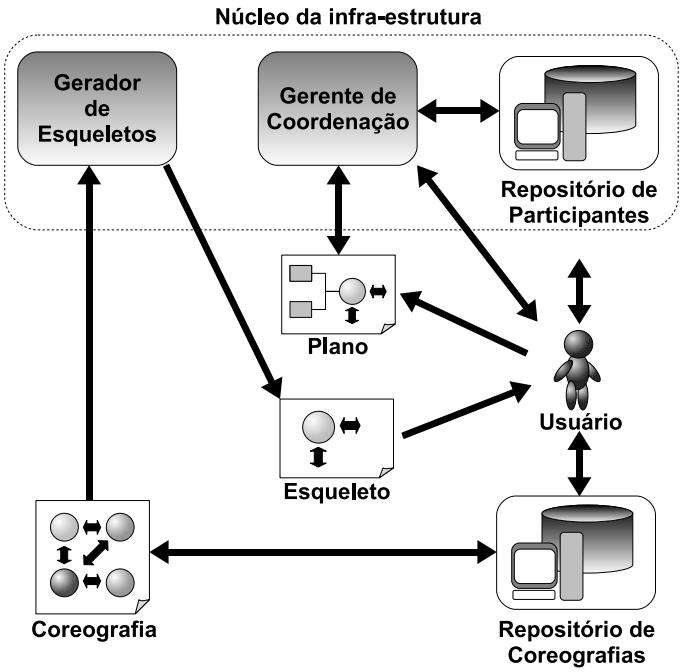

Figura 4.1: Infra-estrutura que suporta o modelo de coordenação.

O gerente de coordenação é responsável por interpretar e executar as instruções dos

planos de coordenação. Ele interage com o repositório de participantes para encontrar em tempo de execução outros gerentes de coordenação que participam da mesma coreografia. Outra função do repositório é permitir a descoberta automática de participantes da cadeia, ou seja, ele fornece um mecanismo para que os participantes sejam encontrados de acordo com os papéis que são capazes de desempenhar. Os planos de coordenação são criados por um programador humano que adequa esqueletos de planos de coordenação segundo as necessidades de um participante. Os esqueletos são gerados automaticamente pelo gerador de esqueletos, que os deriva a partir de coreografias descritas em WS-CDL. O gerente de coordenação, o repositório de participantes e o gerador de esqueleto formam o núcleo da infra-estrutura, sendo considerados essenciais para o suporte ao modelo. A Seções 4.1, 4.2 e 4.3, respectivamente, descrevem estes elementos.

O repositório de coreografia é usado para compartilhar especificações de coreografias com outros participantes da cadeia produtiva. Desta forma, parceiros de negócio podem gerar planos de coordenação adequados aos processos de que pretendem participar. A especificação do repositório de coreografias faz parte de trabalhos futuros.

4.1 Gerente de Coordenação

A especificação do gerente de coordenação é uma das principais contribuições deste trabalho. Este gerente é o elemento da infra-estrutura que executa as coreografias WS-CDL, interpretando e executando os planos de coordenação. A Figura 4.2 ilustra a arquitetura para o gerente de coordenação, que é composta por dois módulos: a *máquina de execução* e o *módulo gerente*. A máquina de execução é o módulo que efetivamente interpreta e executa a lógica descrita por planos de coordenação, enquanto o módulo gerente é responsável pela conexão com outros gerentes de coordenação e pelo encaminhamento de mensagens.

4.1.1 Máquina de Execução

A máquina de execução apresenta a capacidade de disparar a invocação de serviços Web e prover operações de serviços Web, de acordo com a lógica imposta pelo plano de coordenação. Além disso, pode executar concorrentemente diversas instâncias de planos de coordenação, sejam iguais ou diferentes.

Diferente de máquinas de execução BPEL convencionais, a máquina de execução do gerente de coordenação implementa um controle de contexto padronizado. BPEL originalmente utiliza campos específicos de cada mensagem recebida para garantir sua entrega à instância correta do processo. A máquina de execução do gerente de coordenação, assim como WS-*Coordination* [34], utiliza informações que não fazem parte da carga útil das

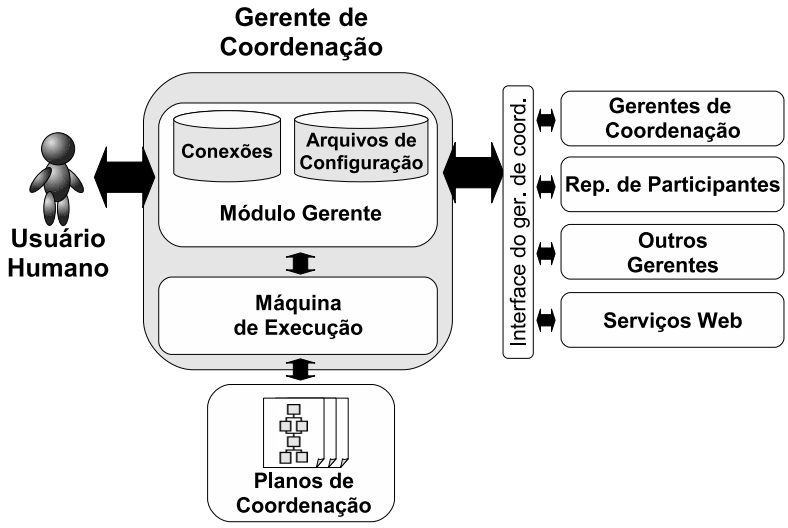

Figura 4.2: Arquitetura para o gerente de coordenação.

mensagens. A vantagem desta abordagem é que o controle de contexto é transparente ao projeto dos planos de coordenação. As informações utilizadas no controle de contexto incluem identificadores únicos para o processo de negócio, para a instância do processo de negócio e para o papel do parceiro que enviou ou a quem se destina a mensagem.

A máquina de execução provê uma interface para a interação com o módulo gerente chamada *EngineToMMIF*. Esta interface permite que o módulo gerente inicie a execução de um plano de coordenação (operação `ExecutePlan`) e requisite operações de controle que podem ser solicitadas por usuários humanos. Foram definidas apenas duas operações de controle, `GetStatus` e `Abort`. A primeira permite que o usuário consulte o estado de um plano de coordenação. A segunda permite que o usuário aborte a execução de um plano de coordenação. Além desta interface, cada plano de coordenação apresenta uma interface específica, que descreve as operações providas pela máquina de execução durante a execução do plano. A Figura 4.3 esquematiza as interfaces da máquina de execução. A Tabela 4.1.1 detalha as operações da interface *EngineToMMIF*.

4.1.2 Módulo Gerente

As funções do módulo gerente (Figura 4.2) são:

- gerenciar a conexão com outros gerentes de coordenação. Para isso necessita interagir com o repositório de participantes;

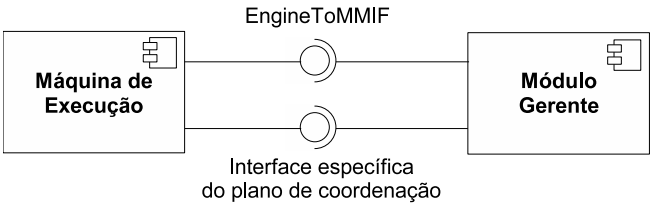

Figura 4.3: Interfaces providas pela máquina de execução para o módulo gerente.

EngineToMMIF		
Operação	Descrição	Parâmetros
`ExecutePlan`	Inicia a execução de um plano de coordenação.	**BusinessProcessID**: Identificador do processo de negócios. Distingue unicamente um processo de negócios.
`GetStatus`	Retorna o estado de um plano de execução	**ContextID**: Identificador de controle de contexto. Distingue unicamente uma instância de um processo de negócios.
`Abort`	Aborta a execução de um plano de execução.	**RoleID**: Identificador do papel que deve ser ou que está sendo desempenhado.

Tabela 4.1: Detalhes da interface *EngineToMMIF*.

- encaminhar mensagens de aplicação entre a máquina de execução e serviços Web. Tais serviços podem ser:
  - outros gerentes de coordenação que representam os parceiros de negócio;
  - aplicações e *workflows* internos;
  - gerentes de negociação, de sumários e de regulamentos;
- fornecer um mecanismo para interação com usuários humanos.

O módulo gerente mantém um arquivo de configuração para cada papel que é capaz de desempenhar. Estes arquivos são documentos XML que contêm informações necessárias para execução do plano de coordenação referente ao respectivo papel. O Código Fonte 4.1 mostra um exemplo de arquivo de configuração. O elemento `planoId` (linha 2) especifica o identificador do plano de coordenação ("plano1") com o qual o arquivo de configuração está relacionado. Os elementos `partner` (linhas 4 a 16) definem a forma de conexão com cada parceiro de negócio. Um parceiro de negócio é classificado como gerente de coordenação ou serviço Web, conforme o valor do atributo `partnerType` ("cm" ou "ws",

respectivamente). Caso o parceiro seja um gerente de coordenação, existem quatro formas de conexão:

- `static`: o *endpoint* do gerente de coordenação é definido no próprio arquivo de configuração. Exemplo: parceiro "r1" (linhas 4 a 8);
- `ns`: o *endpoint* do gerente que está executando um papel específico em uma instância do processo de negócio é obtido por meio do repositório de participantes. Exemplo: parceiro "r2" (linha 9);
- `automatic`: o *endpoint* de um gerente que seja apto a desempenhar um papel específico, é obtido por meio do repositório de participantes. Exemplo: parceiro "r3" (linha 10);
- `connection`: o *endpoint* do gerente é obtido por meio da proposta de conexão enviada por ele. Exemplo: parceiro "r4" (linha 11).

Caso o parceiro não seja um gerente de coordenação, sua forma de conexão é obrigatoriamente estática. Exemplo: parceiro "ws1" (linhas 12 a 16).

Código Fonte 4.1: Exemplo de arquivo de configuração do módulo gerente.

```
1  <planConfig ...>
2    <planId>plano1</planId>
3    <partners>
4      <partner role="r1" partnerType="cm" endpointType="static">
5        <EndpointReference>
6          <Address>http://www.r1.com.br</tns:Address>
7        </EndpointReference>
8      </partner>
9      <partner role="r2" partnerType="cm" endpointType="ns"/>
10     <partner role="r3" partnerType="cm" endpointType="automatic"/>
11     <partner role="r4" partnerType="cm" endpointType="connection"/>
12     <partner role="ws1" partnerType="ws">
13       <EndpointReference>
14         <Address>http://www.ws1.com.br</tns:Address>
15       </EndpointReference/>
16     </partner>
17   </partners>
18 </pcns:planConfig>
```

A partir do momento que uma conexão é estabelecida, o identificador do plano de coordenação e o *endpoint* do parceiro passam a ser armazenados em uma tabela de conexões, de onde podem ser recuperados mais rapidamente. Cada entrada desta tabela possui os seguintes campos:

- PBID: identificador do processo de negócios;
- CID: identificador da instância do processo;
- RID: identificador do papel executado pelo gerente de coordenação;
- planID: identificador do plano de coordenação;
- PRID: identificador do papel do parceiro;
- PEPR: *endpoint* do parceiro;
- status: estado da conexão, que pode ser "connected", quando a conexão já está estabelecida, e "waiting", quando o gerente de coordenação está esperando a resposta do parceiro.

Ao receber uma mensagem de aplicação emitida pela máquina de execução, o módulo gerente utiliza os dados de encaminhamento para recuperar o *endpoint* do gerente destino. De forma análoga, ao receber uma mensagem de um parceiro, os dados de encaminhamento são utilizados para recuperar o identificador do plano de coordenação destino.

O módulo gerente provê duas interfaces, uma para gerência de conexões e outra para suportar a interação com a máquina de execução. Estas duas interfaces, chamadas *ConnectionIF* e *MMToEngineIF* são ilustradas nas Figuras 4.4 e 4.5, respectivamente. Os detalhes das operações são detalhadas pelas Tabelas 4.2(i) e 4.2(ii).

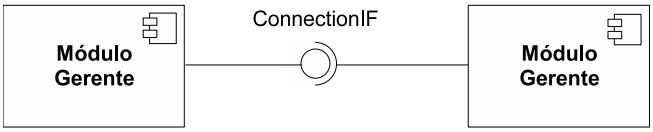

Figura 4.4: Interface provida pelo módulo gerente para conexões com outros módulos gerentes.

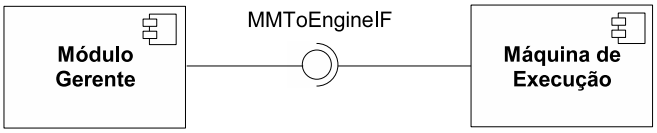

Figura 4.5: Interface provida pelo módulo gerente para a máquina de execução.

O módulo gerente também provê um mecanismo para interação com usuários humanos. Este mecanismo permite que a lógica do plano de coordenação contenha requisições para

ConnectionIF			
Operação	Descrição	Parâmetros Específicos	Parâmetros Comuns
`Connect`	Estabelece uma conexão entre dois gerentes de coordenação.	**CMID**: Identificador do gerente de coordenação que está propondo a conexão. **CMEndpoint**: *Endpoint* do gerente de coordenação que está propondo a conexão. **FromRoleID**: Identificador do papel do emissor da proposta de conexão. **ToRoleID**: Identificador do papel que o receptor da proposta de conexão está sendo convidado a desempenhar. **NSEndpoint**: *Endpoint* do repositório de participantes que está sendo utilizado no processo de negócio.	**BusinessProcessID**: Identificador do processo de negócios. Distingue unicamente um processo de negócios. **ContextID**: Identificador de controle de contexto. Distingue unicamente uma instância de um processo de negócios.
`Disconnect`	Reporta o fim da execução de um papel e finaliza as conexões relacionadas a ele.	**RoleID**: Identificador do papel que está sendo finalizado	

(i)

MMtoEngineIF			
Operação	Descrição	Parâmetros Específicos	Parâmetros Comuns
`PlanConclusion`	Reporta a conclusão de um plano de coordenação.		**BusinessProcessID**: Identificador do processo de negócios. Distingue unicamente um processo de negócios.
`ReportMessage`	Reporta uma mensagem ao usuário humano.	**Message**: Texto da mensagem.	**ContextID**: Identificador de controle de contexto. Distingue unicamente uma instância de um processo de negócios.
`InputRequest`	Solicita uma entrada de dados do usuário humano.	**Message**: Texto da mensagem.	
`OptionRequest`	Solicita uma entrada de dados do usuário humano, fornecendo as opções de entrada.	**Message**: Texto da mensagem. **Lista de opções.**	**RoleID**: Identificador do papel que está sendo desempenhado.

(ii)

Tabela 4.2: (i)Detalhes da interface *ConnectionIF*. (ii) Detalhes da interface *MMToEngineIF*.

intervenção humana no fluxo da execução. Por exemplo: ao receber uma proposta de compra, o gerente de coordenação pode requisitar que o usuário humano decida se a proposta deve ser aceita ou não. Além disso, por meio deste mecanismo, o usuário pode iniciar um plano, acompanhar o seu estado ou abortar sua execução.

O diagrama de seqüência da Figura 4.6 retoma o cenário ilustrado na Figura 3.4, enfocando a interação entre o transportador ("T") e o fornecedor de insumos ("F"). Este novo exemplo, mostra as interações entre os módulos gerentes e as máquinas de execução de cada participante.

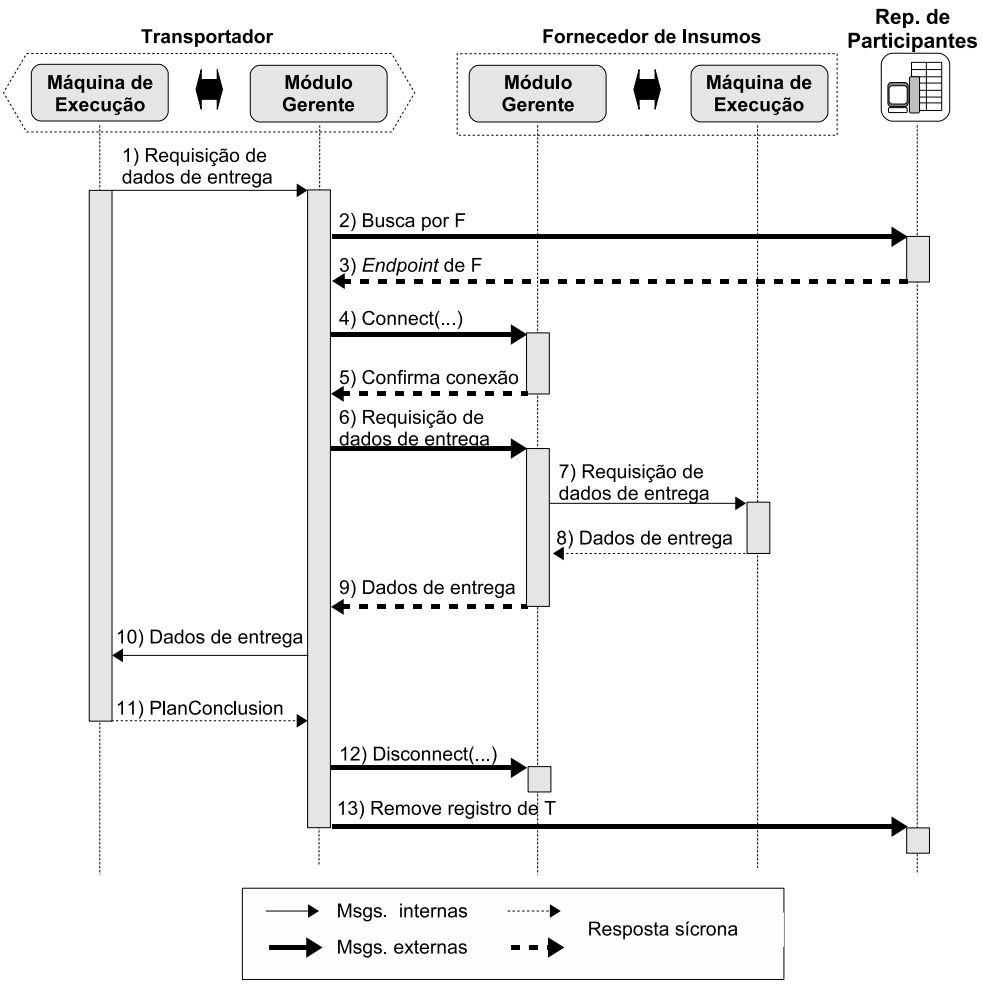

Figura 4.6: Trecho do cenário ilustrado na Figura 3.4 enfocando a interação entre os módulos gerentes e máquinas de execução do transportador e do fornecedor de insumos.

Para ilustrar a interação com o usuário humano, considere que, no mesmo cenário ilustrado pela Figura 4.6, "F" retorne uma exceção ao invés dos dados de entrega (mensagem 9) e que esta exceção tenha que ser reportada para o usuário. Neste caso, ao receber a

mensagem de exceção, a máquina de execução de "T" emite uma mensagem ao usuário que é encaminhada por meio do seu módulo gerente. Em seguida, a execução do plano de coordenção é finalizada. Esta situação é ilustrada no diagrama de seqüência da Figura 4.7.

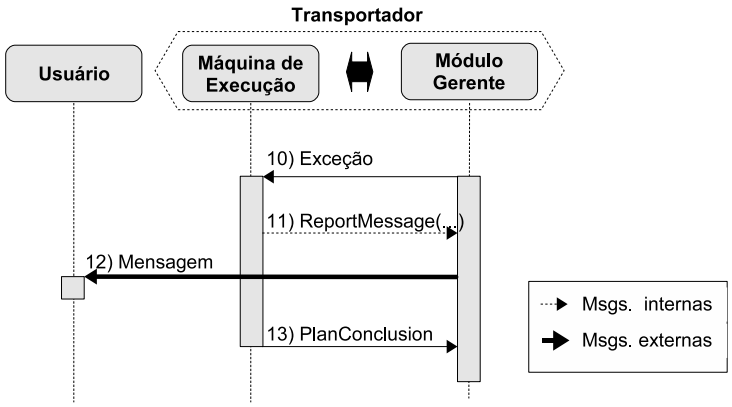

Figura 4.7: Exemplo de interação com o usuário.

4.2 Repositório de Participantes

O repositório de participantes tem duas funções. A primeira é possibilitar que participantes da cadeia (por meio de seus gerentes de coordenação) encontrem parceiros de negócios que desempenhem papéis específicos (descoberta automática). A segunda é servir como servidor de nomes, permitindo que gerentes de coordenação obtenham o *endpoint* de outros gerentes em tempo de execução (ligação dinâmica).

Este repositório pode ser implementado como um serviço de registro UDDI. Neste caso, cada participante cadastra um `uddi:businessEntity`[1] no serviço de registro. O `uddi:businessEntity` apresenta informações do participante da cadeia produtiva, como descrição e forma de contato. Além disso, o sistema de categorização da UDDI é utilizado para relacionar participates e papéis. A Figura 4.8 apresenta um exemplo de registro de participante. O elemento `uddi:keyedReference` (linhas 5 e 6) indica que o participante é capaz de desempenhar o papel R_A do processo de negócio C_1, conforme indicado pelo atributo `keyValue` (linha 6). O atributo `keyName` especifica o identificador do papel. Já o atributo `tModelKey` poderia referenciar um `uddi:tModel` específico que categorizasse todos os papéis de processos de negócio existentes em determinado domínio.

A busca por um participante capaz de desempenhar o papel R_A da coreografia C_1, deve ser realizada por meio da opearação `find_business` da API de busca da UDDI,

[1]Nesta seção, o prefixo `uddi` é usado para identificar elementos da UDDI.

usando a categoria adequada como parâmetro. A Figura 4.9 mostra a requisição SOAP para esta operação.

```
1 <businessEntity ...>
2   Nome, descrição, contato ...
3   businessServices...
4     <categoryBag>
5        <keyedReference keyName="RA"
6           keyValue="C1:RA" tModelKey="..."/>
7        ...
8     </categoryBag>
9 </businessEntity>
```

Figura 4.8: `uddi:businessEntity` de um participante capaz de desempenhar o papel R_A de C_1.

```
...
<soap:body>
   <find_business ...>
      <categoryBag>
         <keyedReference keyName="RA"
            keyValue="C1:RA" tModelKey="..."/>
         ...
      </categoryBag>
   </find_business>
<soap:body>
...
```

Figura 4.9: Código SOAP para encontrar participantes aptos a desempenhar o papel R_A da coreografia C_1.

Todo papel em execução deve ser representado no serviço de registro como um elemento `uddi:businessService`. O exemplo da Figura 4.10 mostra um elemento `uddi:businessService` representando a instância *CID* do papel R_A da coreografia C_1. O elemento `uddi:bindingTemplate` (linhas 6 a 13) fornece o *endpoint* do gerente de coordenação que está desempenhando o papel (linha 7) e referencia um `uddi:tModel` que contém ligações para a descrição da coreografia e o documento WSDL correspondentes ao processo de negócio. O elemento `uddi:categoryBag` (linhas 15 a 20) é utilizado para identificar a instância do processo de negócio. O elemento `uddi:keyedReference` das linhas 16 e 17 indica que o `uddi:businessService` se refere ao papel R_A do processo de negócio C_1. Já o `uddi:keyedReference` das linhas 18 e 19 indica o identificador da instância do processo.

Toda vez que um gerente de coordenação inicia a execução de um plano, ele deve adicionar um novo elemento `uddi:businessService` no registro do participante ao qual representa. Esta tarefa é realizada por meio da operação `save_service` da API de publicação da UDDI. A Figura 4.11 ilustra o código SOAP para esta operação.

```
<businessEntity ...>
   ...
   <businessService>
      ...
      <bindingTemplates>
         <bindingTemplate>
            <accessPoint>CM endpoint</accessPoint>
            <tModelDatails>
               <tModelInstanceInfo tModelKey="uuid">
               ...
               </tModelInstanceInfo>
            </tModelDatails>
         </bindingTemplate>
      </bindingTemplates>
      <categoryBag>
         <keyedReference keyName="RA"
            keyValue="C1:RA" tModelKey="..."/>
         <keyedReference keyName="CID"
            keyValue="C1:RA:CID" tModelKey="..."/>
      </categoryBag>
   </businessService>
</businessEntity>
```

Figura 4.10: `uddi:businessService` representando a instância *CID* do papel R_A da coreografia C_1.

De forma análoga, quando um gerente de coordenação termina a execução de um plano, ele deve excluir o elemento `uddi:businessService` correspondente do registro. Isto é feito por meio da operação `delete_service` da API de publicação da UDDI. O código SOAP para esta operação é apresentado na Figura 4.12.

```
...
<soap:body>
   <save_service...>
      <authInfo> ... <authInfo>
      <businessService>
         ...
      </businessService>
   </save_service>
</soap:body>
...
```

Figura 4.11: Código exemplo da operação `save_service`.

O *endpoint* do gerente de coordenação que está desempenhando um papel específico pode ser obtido da seguinte forma: utiliza-se a operação `find_service` da API de busca da UDDI para encontrar o identificador do `uddi:businessService` que representa a instância do papel. Esta operação é exemplificada na Figura 4.13. Em seguida, usa-se o identificador obtido para executar a operação `find_binding`, que retorna o elemento `uddi:bindingTemplate`, de onde pode ser extraído o *endpoint*. A Figura 4.14 mostra o

código SOAP para esta operação.

```
...
<soap:body>
   <delete_service...>
      <authInfo> ... <authInfo>
      <serviceKey>
         ...
      </serviceKey>
   </delete_service>
<soap:body>
...
```

Figura 4.12: Código exemplo da operação `delete_service`.

```
...
<soap:body>
<find_service...>
   <categoryBag>
      <keyedReference keyName="RA"
         keyValue="C1:RA" tModelKey="..."/>
      <keyedReference keyName="CID"
         keyValue="C1:RA:CID" tModelKey="..."/>
   </categoryBag>
</find_service>
</soap:body>
...
```

Figura 4.13: Código exemplo da operação `find_service`.

```
...
<soap:body>
   <find_binding serviceKey="..."/>
</soap:body>
...
```

Figura 4.14: Código exemplo da operação `find_binding`.

4.3 Gerador de Esqueletos de Planos de Coordenação

O gerador de esqueletos de planos de coordenação é um mecanismo que gera automaticamente esqueletos de planos de coordenação a partir de coreografias WS-CDL. Além de aumentar a rapidez da tarefa, esta automatização diminui a possibilidade de erros e inconsistências na geração dos planos. A entrada do gerador é uma coreografia descrita em WS-CDL, um documento WSDL contendo as definições de tipos de mensagens

e `portTypes` que são referenciados e o identificador do papel para o qual se deseja gerar o esqueleto. Como saída, são gerados um esqueleto descrito em BPEL e um documento WSDL específicos para o papel. Por convenção, o esqueleto do plano de coordenação referente ao papel R_A será identificado por EPC_A e o documento WSDL correspondente por $WSDL_A$. A Figura 4.15 mostra o esquema da geração do esqueleto para um papel R_A.

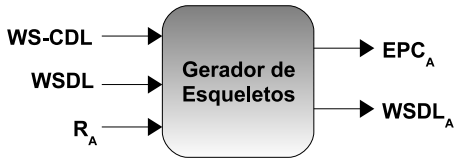

Figura 4.15: Geração de esqueleto referente ao papel R_A.

A especificação do gerador de esqueletos não tem o objetivo de apresentar uma tradução detalhada da linguagem WS-CDL para BPEL, e sim validar o modelo de coordenação. Neste sentido, foram adotadas algumas restrições à utilização da linguagem WS-CDL. Estas restrições impedem o uso de alguns elementos cujas funções não são aplicáveis ao modelo de coordenação ou que não são essenciais à sua validação. São elas[2]:

1. `cdl:tokens` e `cdl:tokensLocators`: os elementos `cdl:token` e `cdl:tokenLocator` são utilizados para identificar campos únicos das mensagens para controle de contexto. Como o gerente de coordenação utiliza uma forma de controle de contexto própria, estes elementos não são considerados pelo gerador de esqueletos.

2. `cdl:roleTypes`: o número de comportamentos (elementos `cdl:behavior`) relacionados a um `cdl:roleType` foi limitado a um. Isto implica que um único `cdl:portType` contenha toda a especificação da interface do `cdl:roleType` relacionado. Esta limitação não afeta o modelo de coordenação, pois o caso no qual um gerente de coordenação executa um papel com vários comportamentos pode ser tratado pelo caso no qual este gerente executa vários papéis.

3. `cdl:channelTypes`: no modelo de coordenação proposto, *endpoints* e informação de contexto são implicitamente tratados pelo gerente de coordenação. Além disso, BPEL não apresenta uma abstração de canal de comunicação, então as restrições impostas pelo `cdl:channelType` não são aplicáveis à arquitetura do gerente de coordenação. Por estas razões, `cdl:channelTypes` são desconsiderados pelo gerador de esqueletos.

[2]Nesta seção, o prefixo `cdl` será usado para referenciar elementos da linguagem WS-CDL. De forma análoga, o prefixo `bpel` será usado para referenciar elementos da linguagem BPEL.

4. `cdl:variables`: os elementos `cdl:variable` apresentam atributos que definem restrições ao uso das variáveis. Estas restrições não são aplicáveis ao elemento `bpel:variable`, sendo desconsideradas pelo gerador de esqueletos.

5. `cdl:exchanges`: um elemento `cdl:interaction` pode definir um número ilimitado de elementos `cdl:exchange`. Nestes casos, segundo a especificação da linguagem WS-CDL, deve haver uma escolha implícita entre os `cdl:exchanges` que efetivamente devem ser executados, o que implica na adição manual de código BPEL no mapeamento. Para evitar isto, o número de `cdl:exchanges` foi restringido para: um, no caso de interações do tipo *request*; ou, no máximo três, no caso de interações do tipo *request-respond*. No segundo caso, o primeiro `cdl:exchange` corresponde à invocação da operação, o segundo corresponde à resposta e o terceiro a uma possível mensagem de falta.

6. `Funções WS-CDL`: WS-CDL utiliza a linguagem XPath [35] para representação de expressões e a estende definindo funções próprias. Apenas duas dessas funções são suportadas pelas regras de mapeamento implementadas pelo gerador de esqueletos:

   - `cdl:getVariable(nome, parte, caminho, roleType)`: retorna a informação de uma variável com nome definido pelo primeiro parâmetro. O segundo e o terceiro parâmetros especificam a porção da informação desejada. O último identifica o `cdl:roleType` a quem a informação é pertinente.
   - `cdl:globalizedTrigger (exp1, role1, exp2, role2, ...)`: combina expressões definidas por diferentes *roleTypes*. Por exemplo, `exp1` é definida em termos das variáveis de `role1`, `exp2` em termos das variáveis de `role2` e assim por diante.

7. Em um primeiro momento, os seguintes recursos foram desconsiderados pelas regras de mapeamento do gerador de esqueletos por não serem considerados essenciais à validação do modelo de coordenação:

   - Coreografias aninhadas;
   - `cdl:timeouts`: um elemento `cdl:timeout`, dentro de um elemento `cdl:interaction`, é usado para especificar o intervalo de tempo no qual uma interação dever ser completada;
   - `cdl:records`: elementos `cdl:record`, dentro de um elemento `cdl:interaction`, são usados para criar ou alterar o valor de uma ou mais variáveis usando outras variáveis ou expressões. O mesmo resultado pode ser alcançado utilizando-se elementos `cdl:assign` em conjunto com o elemento `cdl:interaction`;

- `cdl:finalizerBlock`: definem atividades que devem ser executadas quando uma coreografia é finalizada. Efeito semelhante pode ser alcançado com a adição destas atividades no final da coreografia;
- Tratamento de exceções.

Estes recursos serão adicionados em trabalhos futuros.

A partir destas restrições, são definidas as regras de mapeamento de WS-CDL para BPEL e as regras de geração dos documentos WSDL específicos para cada esqueleto. O mapeamento é apresentado na Seção 4.3.1 e a geração dos documentos WSDL na Seção 4.3.2.

4.3.1 Regras de Mapeamento

Esta seção enfoca o mapeamento entre coreografias WS-CDL e os planos de coordenação BPEL, uma das principais contribuições deste trabalho. Embora as idéias básicas destas regras sejam aplicáveis ao mapeamento WS-CDL/BEPL genérico [7, 8], o mapeamento proposto gera código BPEL específico para gerentes de coordenação. De forma geral, para cada `cdl:roleType` da coreografia, gera-se um esqueleto de plano de coordenação. A seguir, são descritas as regras de mapeamento para cada elemento da coreografia.

Variáveis

Todo elemento `cdl:variable`, que referencia R_A, gera um novo elemento `bpel:variable`, com o mesmo nome, em EPC_A. O atributo `informationType` do elemento `cdl:variable` é mapeado para o atributo `messageType` do elemento `bpel:variable`, exceto pelo prefixo do *namespace* que deve ser substituído pelo prefixo relativo ao $WSDL_A$. A Figura 4.16 apresenta um exemplo deste mapeamento.

Expressões

Tanto WS-CDL quanto BPEL utilizam XPath para representação de expressões. No entanto, algumas funções são particulares para cada linguagem. O gerador de esqueletos de planos de coordenação considera duas funções da linguagem WS-CDL nas regras de mapeamento, `cdl:getVariable` e `cdl:globalizedTrigger`. A primeira é diretamente mapeada para a função `bpel:getVariableData`. Já o mapeamento da função `cdl:globalizedTrigger` gera o mapeamento da subexpressão correspondente ao *roleType* específico. Os dois casos são ilustrados pela Figura 4.17.

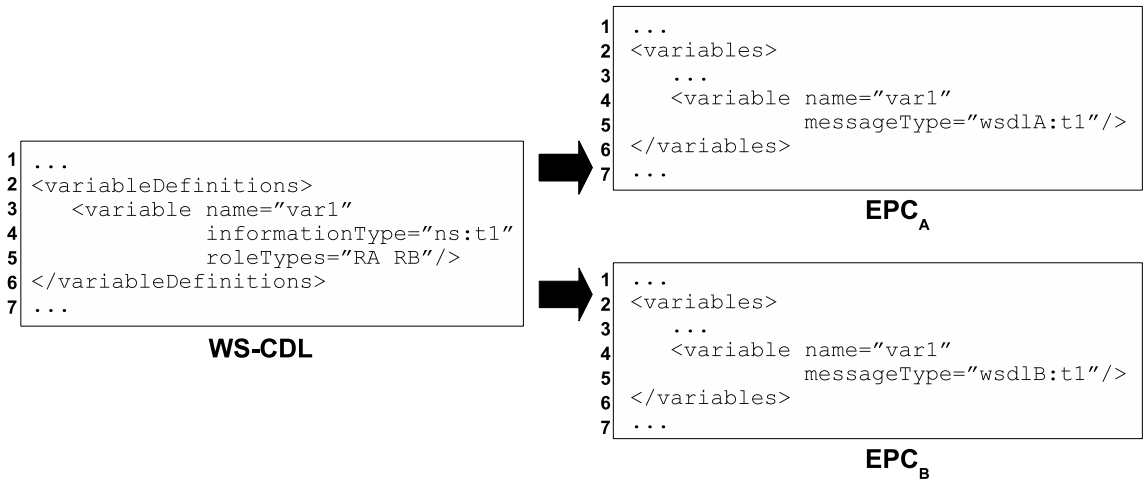

Figura 4.16: Exemplo de mapeamento do elemento `cdl:variable`.

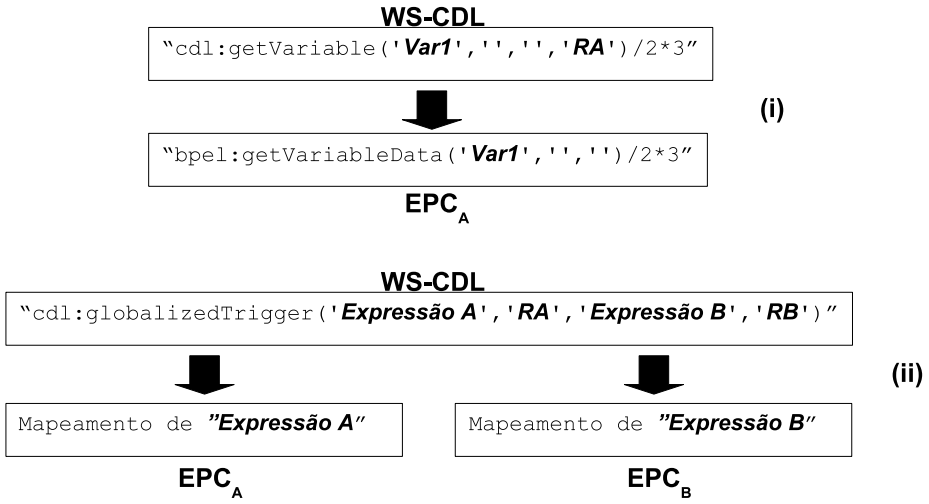

Figura 4.17: (i) Exemplos de mapeamento de expressãoõess baseadas em: (i) função `cdl:getVariable` e (ii) função `cdl:globalizedTrigger`.

Controle de Fluxo

Na linguagem WS-CDL, o controle de fluxo é suportado pelos elementos `cdl:sequence`, `cdl:parallel`, `cdl:choice` e `cdl:workunit`. As seguintes regras são utilizadas para o mapeamento destes elementos:

1. `Sequence`: um elemento `cdl:sequence` gera um elemento `bpel:sequence`. As atividades internas ao `cdl:sequence` são mapeadas para o interior do `bpel:sequence` gerado.

2. `Parallel`: um elemento `cdl:parallel` gera um elemento `bpel:flow` em cada *EPC*. As atividades internas ao `cdl:parallel` são mapeadas para o interior do `bpel:flow` gerado.

3. `Choice`: um elemento `cdl:choice` gera um elemento `bpel:switch`. As atividades internas ao `cdl:choice` devem ser mapeadas e encapsuladas em elementos do tipo `bpel:case`, no interior do `bpel:switch`. As condições dos elementos `bpel:case` devem ser adicionadas manualmente por um programador. A Figura 4.18 mostra um exemplo deste mapeamento.

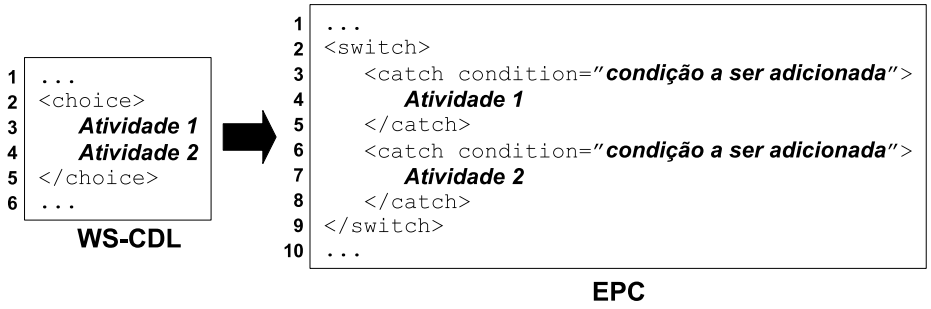

Figura 4.18: Exemplo de mapeamento do elemento `cdl:choice`.

4. `cdl:workunit`: o mapeamento do elemento `cdl:workunit` apresenta casos diferentes, dependendo do valor dos atributos `block`, `guard` e `repeat`. Em qualquer desses casos, o código BPEL gerado deve ser encapsulado por um elemento `bpel:sequence`. Se o valor do atributo `block` for "verdadeiro", gera-se um *loop* (elemento `bpel:while`) que tem o objetivo de interromper o fluxo de execução do plano de coordenação até que a condição de guarda seja satisfeita. A condição de guarda (`guard`) é mapeada para um elemento `bpel:case`, interno a um elemento `bpel:switch`. Caso o atributo `repeat` seja "verdadeiro", gera-se um `bpel:while` cuja condição é o mapeamento da condição de repetição. No interior deste `bpel:while`, gera-se outra condição de guarda (`bpel:switch` + `bpel:case`). A Figura 4.19 apresenta um exemplo.

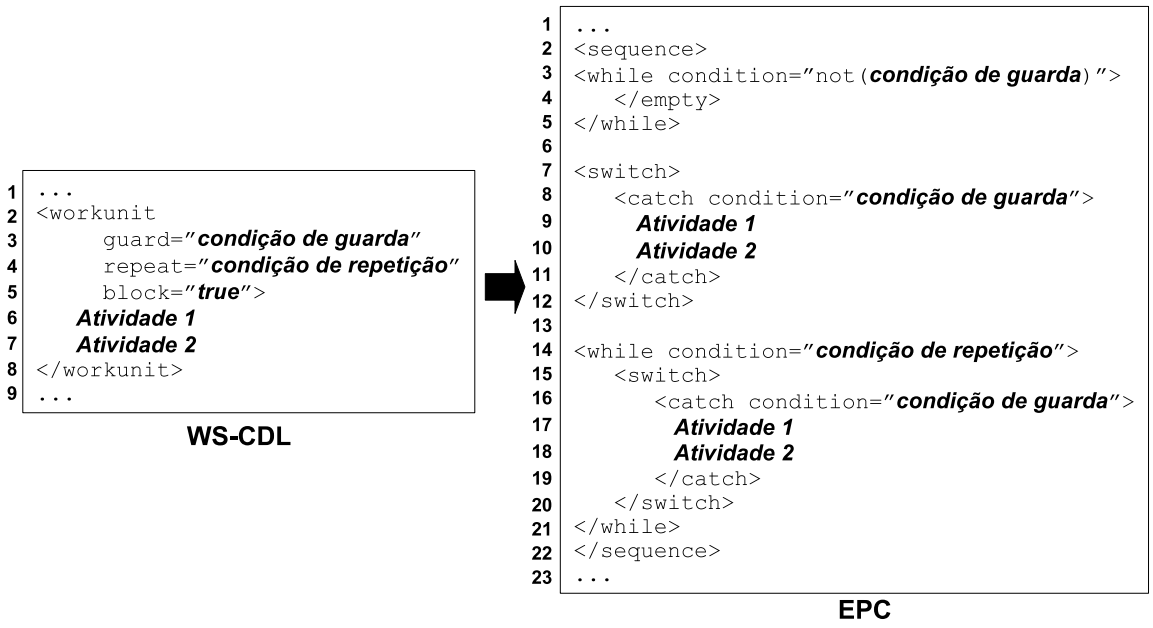

Figura 4.19: Exemplo de mapeamento do elemento `cdl:workunit`.

Atividades Básicas

O gerador de esqueletos de planos de coordenação suporta as atividades básicas `cdl:silentAction`, `cdl:assign` e `cdl:interaction`. As seguintes regras de mapeamento referem-se a estas atividades:

1. `cdl:silentAction`: um elemento `cdl:silentAction` que referencia R_A é mapeado para um elemento `bpel:empty` em EPC_A.

2. `cdl:assign`: elementos `cdl:assign` e `cdl:copy` que referenciam R_A são mapeados para elementos `bpel:assign` e `bpelcopy`, respectivamente, em EPC_A. Elementos `cdl:source` e `cdl:target` são mapeados para elementos `bpel:from` e `bpel:to`, respectivamente. Esta regra é exemplificada pela Figura 4.20.

3. `cdl:Interaction`: cada elemento `cdl:exchange` interno a um `cdl:interaction` deve ser mapeado para um elemento `bpel:invoke` ou `bpel:response`, dependendo do tipo da ação (*request* ou *respond*) e se o `cdl:roleType` é o invocador ou o provedor da operação relacionada. Considere que R_A é o invocador e R_B é o provedor. No caso de um `cdl:exchange` de invocação, gera-se um `bpel:invoke` em EPC_A e um `bpel:receive` em EPC_B. Este caso é ilustrado pela Figura 4.21. Note que o `cdl:exchange` de invocação não define a variável de saída do elemento `bpel:invoke` correspondente (`variável4` no exemplo). Esta variável deve ser obtida do `cdl:exchange` de resposta.

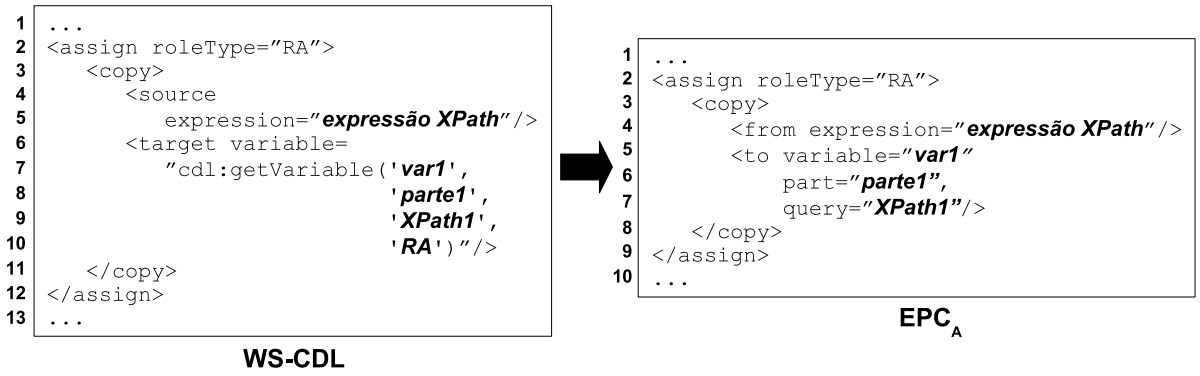

Figura 4.20: Exemplo de mapeamento do elemento `cdl:assign`.

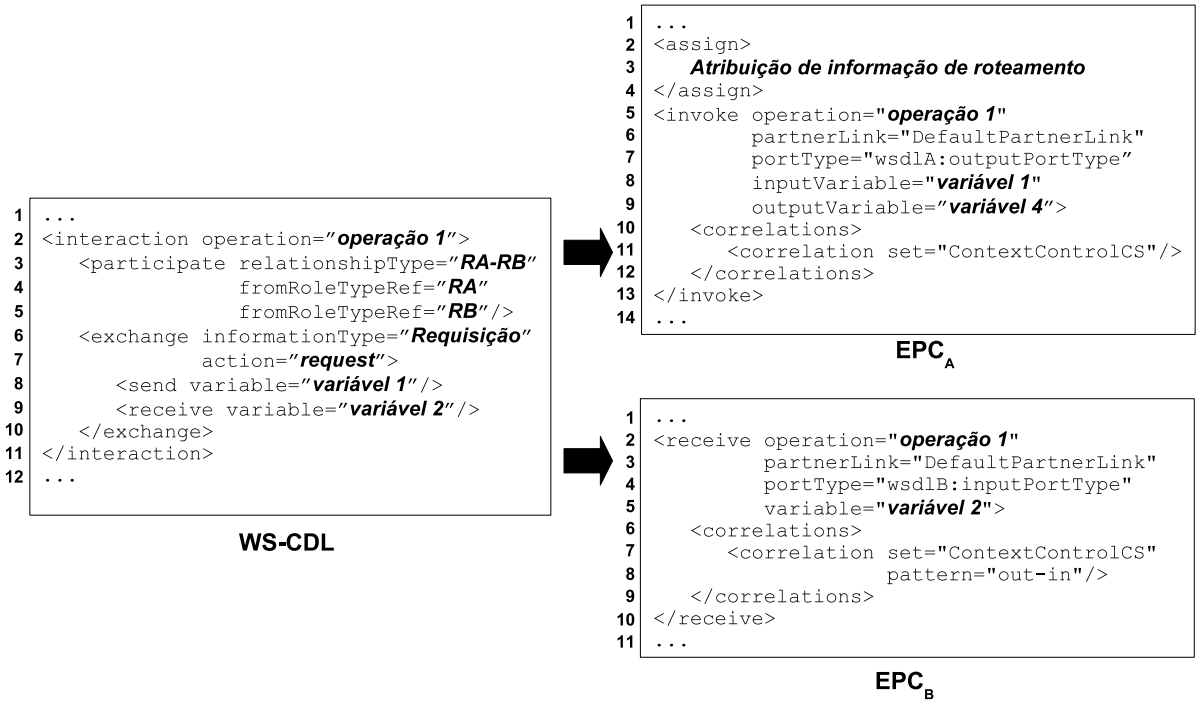

Figura 4.21: Exemplo de mapeamento do elemento `cdl:interaction` no caso de ação do tipo *request*.

Nos planos de coordenação, todos os elementos `bpel:invoke`, `bpel:receive` e `bpel:reply` devem referenciar um `bpel:partnerLink` padrão chamado *Default-PartnerLink*. Este `bpel:partnerLink` represnta a ligação entre a máquina de execução e o módulo gerente. O `bpel:correlationSet` *ContextControlCS* também deve ser referenciado. Este elemento especifica os campos padrão utilizados para o controle de contexto.

Se o `cdl:exchange` for de resposta, gera-se um `bpel:reply` em EPC_B. Existe também um terceiro caso, no qual o `cdl:exchange` é de resposta e retorna uma falta. Neste caso, gera-se um `bpel:catch` interno ao `bpel:invoke` de invocação, em EPC_A, e um `bpel:reply` em EPC_B. Quando mais de um `cdl:exchange` de resposta são definidos dentro de um mesmo `cdl:interaction`, os elementos `bpel:reply` correspondentes devem ser encapsulados por elementos `bpel:case`, e as condições para que cada um seja executado devem ser adicionadas manualmente. A Figura 4.22 apresenta um exemplo deste caso.

4.3.2 Geração de WSDL Referente a Esqueleto

Um documento WSDL referente a um plano de coordenação define dois `wsdl:portTypes` que suportam as trocas de mensagens entre a máquina de execução e o módulo gerente durante a execução do plano. São eles: *inputPortType* e *outputPortType*. O *inputPortType* contém todas as operações providas pela máquina de execução, enquanto o *outputPortType* contém todas as operações invocadas. A seguir, são descritas as regras para geração destes `wsdl:portTypes`. Supõe-se estar gerando um documento WSDL para o papel R_A.

1. O *inputPortType* deve conter as declarações das operações definidas pelo `wsdl:portType` referenciado pelo elemento `cdl:behavior` de R_A;

2. O *outputPortType* deve conter as declarações de todas as operações definidas por todos os `portTypes` referenciados pelos elementos `cdl:behavior` dos `cdl:roleTypes` que participam de relacionamentos com $\texttt{R}_A$.

A Figura 4.23 mostra um exemplo de geração dos `portTypes` *inputPortType* e *outputPortType*. Definidos os `portTypes`, é possível identificar as declarações de tipos de mensagens que devem ser incluídos em $WSDL_A$. Seguem as regras para declaração de tipos, que são ilustradas pela Figura 4.24.

1. Todas as declarações de tipo referenciadas por qualquer operação presente em *inputPortType* ou em *outputPortType* devem ser incluídas em $WSDL_A$;

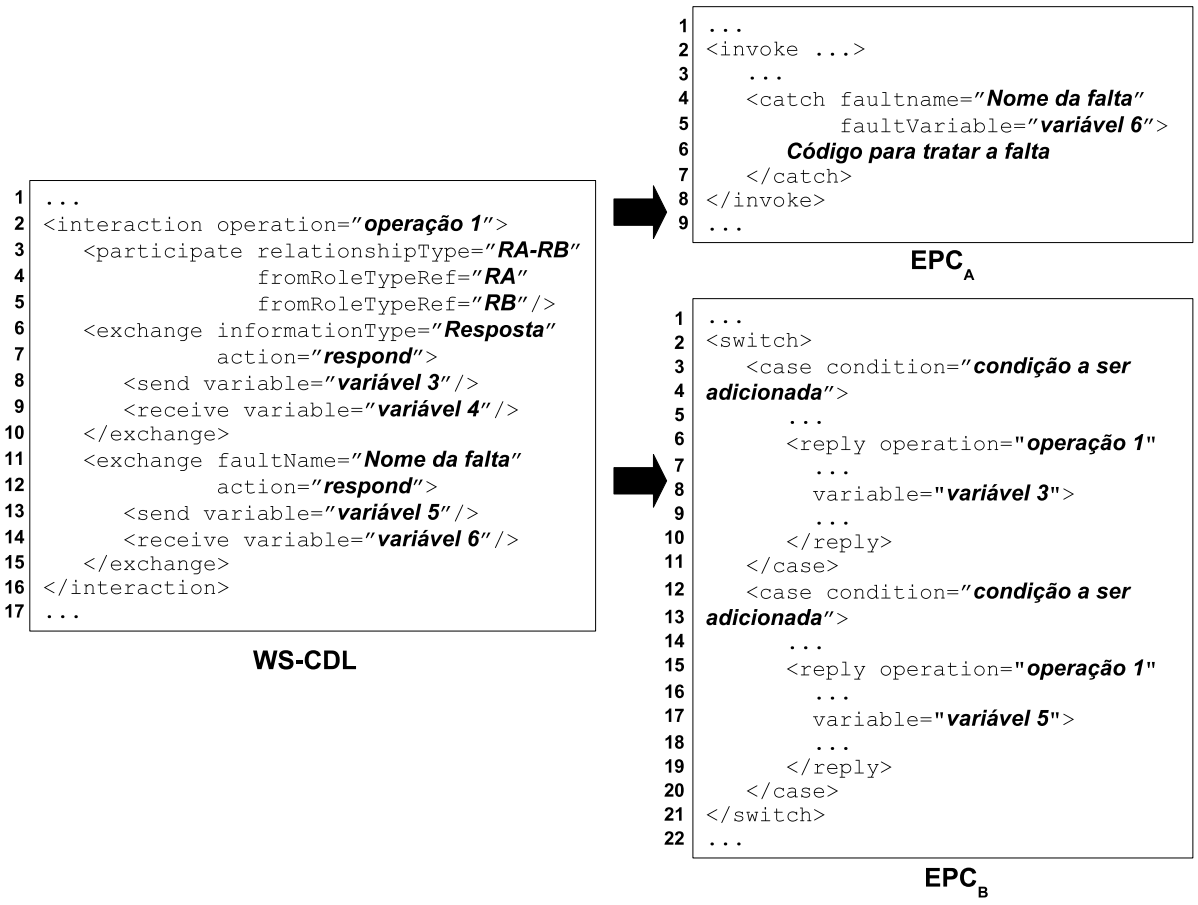

Figura 4.22: Exemplo de mapeamento do elemento `cdl:interaction` no caso de ações do tipo *respond*.

2. Toda declaração de tipo incluída deve ser acrescida de um novo `wsdl:part`, que contém os dados para encaminhamento de mensagens. Este `wsdl:part` tem o nome *coordinationData*.

Caso alguma declaração de tipo referencie um esquema XML, este esquema também deve ser incluído em $WSDL_A$. O esquema `http://sca/schemas/CMBpelSchema.xsd`, que define o elemento `coordinationData`, referenciado pelo `wsdl:part` com mesmo nome, sempre deve ser incluído. A inclusão de esquemas XML é exemplificada na Figura 4.25.

$WSDL_A$ deve definir três propriedades (elementos `bpel:property`) que são usadas no controle de contexto do plano de coordenação. São elas: *BPIDProperty*, que referencia o identificador do processo de negócios, *CIDProperty*, que referencia o identificador da instância do processo e *RIDProperty*, que referencia o identificador do papel (R_A). Para cada tipo de mensagem, deve-se incluir um `bpel:propertyAlias` referente a cada uma destas propriedades. A Figura 4.26 apresenta um exemplo que ilustra a adição das

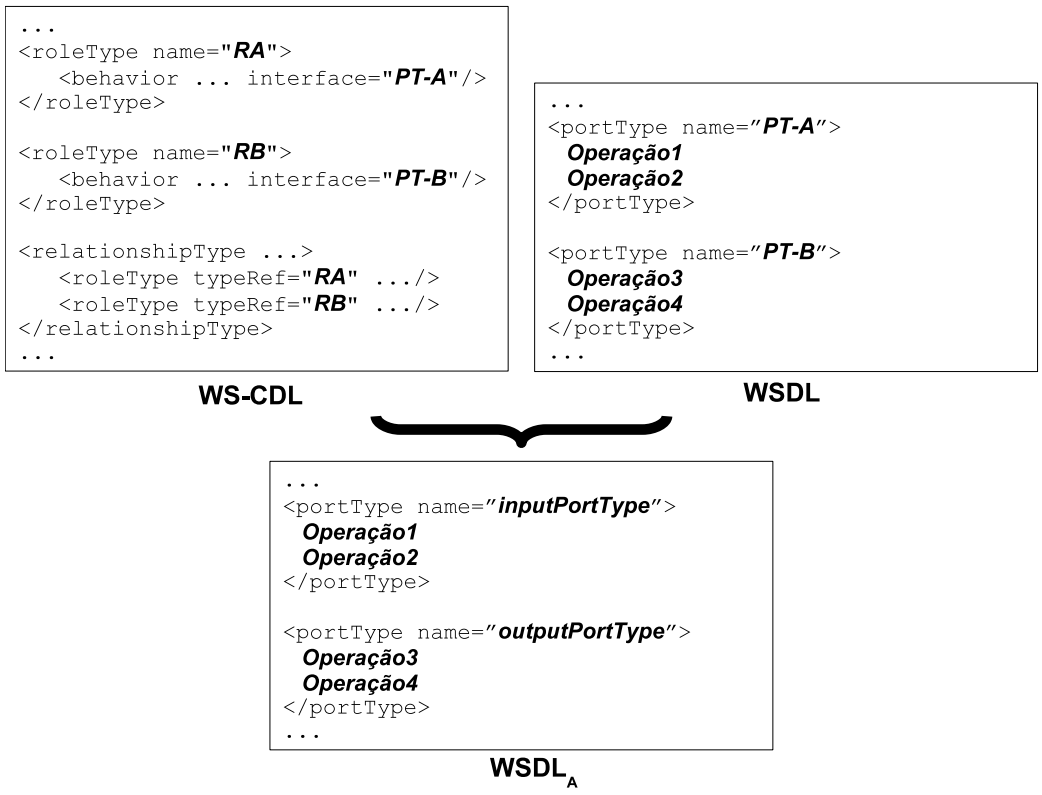

Figura 4.23: Geração dos `portTypes` *inputPortType* e *outputPortType*.

propriedades e dos elementos `propertyAlias` em $WSDL_A$.

$WSDL_A$ também deve incluir um `bpel:partnerLinkType` que especifica a ligação entre a máquina de execução e o módulo gerente. O código para este elemento é apresentado na Figura 4.27.

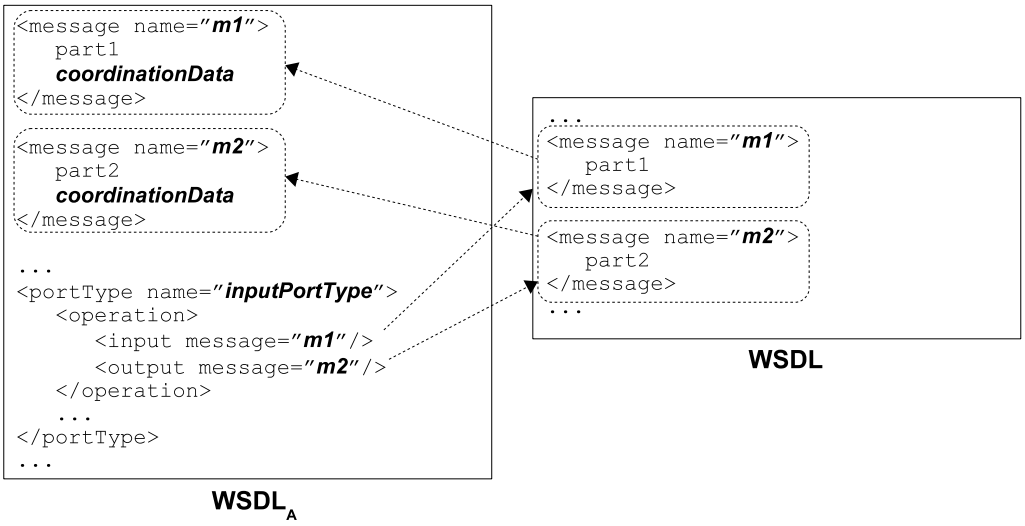

Figura 4.24: Inclusão de declarações de tipos de mensagens.

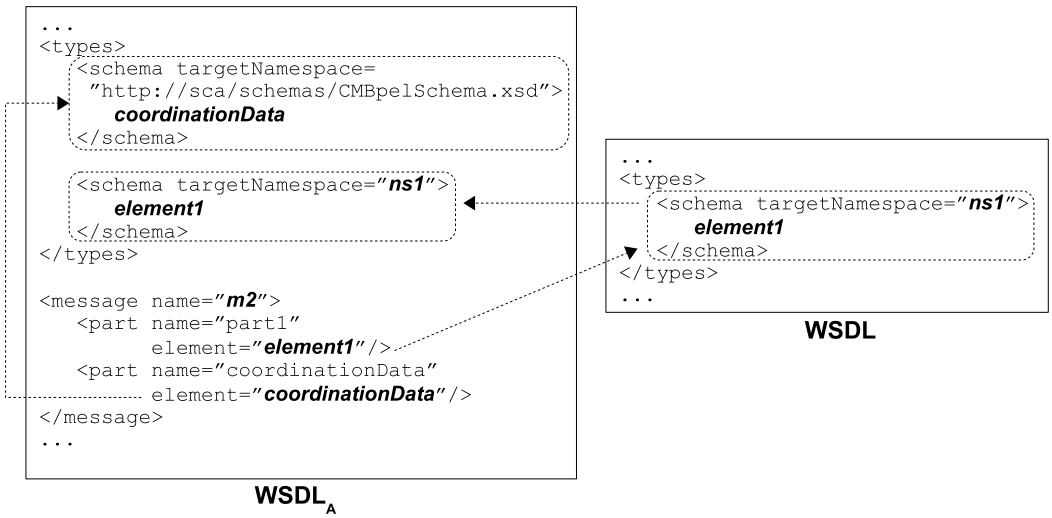

Figura 4.25: Declaração de esquemas XML.

Figura 4.26: Inclusão de elementos `bpel:property` e `propertyAlias`.

```
...
<partnerLinkType name="defaultPLT">
   <role name="ExecutionEngine">
      <portType name="inputPortType"/>
   </role>
   <role name="ManagerModule">
      <portType name="outputPortType"/>
   </role>
</partnerLinkType>
...
```

WSDL$_A$

Figura 4.27: Código do elemento `partnerLinkType` padrão.

Capítulo 5

Implementação

Com o objetivo de validar o modelo de coordenação proposto, foi implementado um protótipo da infra-estrutura apresentada no Capítulo 4. Não foi possível contemplar todas as funcionalidades da infra-estrutura, portanto foram implementadas somente as principais. Os protótipos do gerente de coordenação, do repositório de participantes e do gerador de esqueletos são apresentados nas Seções 5.1, 5.2 e 5.3, respectivamente.

5.1 Protótipo do Gerente de Coordenação

A Figura 5.1 esquematiza o protótipo do gerente de coordenação. Utilizou-se uma máquina de execução BPEL convencional[1] para realizar a função de máquina de execução. Aspectos sobre o uso desta máquina são abordados na Seção 5.1.1. O módulo gerente foi implementado utilizando a linguagem Java e apresenta uma interface *servlet* para receber e responder mensagens SOAP sobre o protocolo HTTP. O processamento das mensagens SOAP foi implementado utilizando-se a API JAXM (*Java API for XML Messaging*). O servidor Apache Tomcat foi adotado como conteiner para a máquina de execução e para o módulo gerente. A persistência dos dados é provida por um banco de dados Postgres. Detalhes a respeito do módulo gerente são apresentados na Seção 5.1.2.

5.1.1 Aspectos da Máquina de Execução

Como a máquina de execução adotada não foi especificamente projetada para a arquitetura do gerente de coordenação, a lógica referente à interface *EngineToMMIF* (Seção 4.1.1) precisa ser suprida pelo código dos planos de coordenação. Portanto, é necessário que os planos apresentem uma estrutura predefinida. A Figura 5.2 mostra a estrutura básica

[1] ActiveBPEL, disponível em http://activebpel.org.

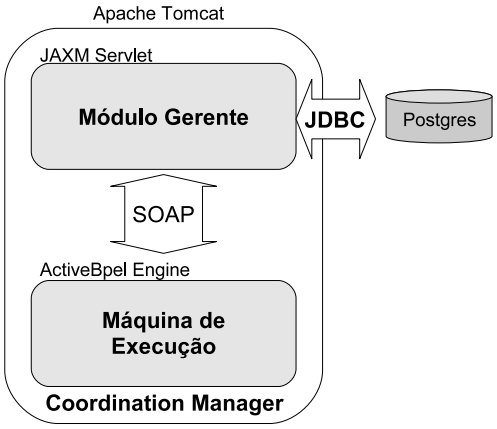

Figura 5.1: Protótipo do gerente de coordenação.

que um plano deve apresentar para que o comportamento da máquina de execução, ao executá-lo, esteja em conformidade com a arquitetura do gerente de coordenação.

O fluxo normal do processo BPEL é iniciado pela invocação da operação `ExecutePlan`. Em seguida, a lógica específica do processo de negócio é executada. Esta parte do código é gerada a partir da descrição da coreografia, pelo gerador de esqueletos, e pode ser customizada por um programador. O fluxo normal termina com a invocação da operação `PlanConclusion`. As operações `GetStatus` e `Abort` são implementadas como tratadores de eventos (`bpel:eventHandlers`). Os tratadores de eventos são elementos da linguagem BPEL que permitem a execução de atividades concorrentes ao fluxo normal do processo. Estas atividades são disparadas por determinados eventos, como a recepção de uma mensagem. A operação `GetStatus` retorna o valor de uma variável padrão que recebe um identificador do estado do processo em pontos específicos do fluxo normal. `Abort` força o término do processo, porém executa um trecho de código customizado antes.

5.1.2 Implementação do Módulo Gerente

A Figura 5.3 mostra o diagrama das classes que compõem o módulo gerente. O diagrama foi simplificado para facilitar seu entendimento. Foram suprimidos os atributos e os métodos. Segue uma breve descrição destas classes.[2]

CMServlet Representa a interface de serviço Web do módulo gerente. Permite receber e responder mensagens SOAP sobre o protocolo HTTP. As mensagens recebidas são

[2]As interfaces destas classes são apresentadas no Apêndice A.

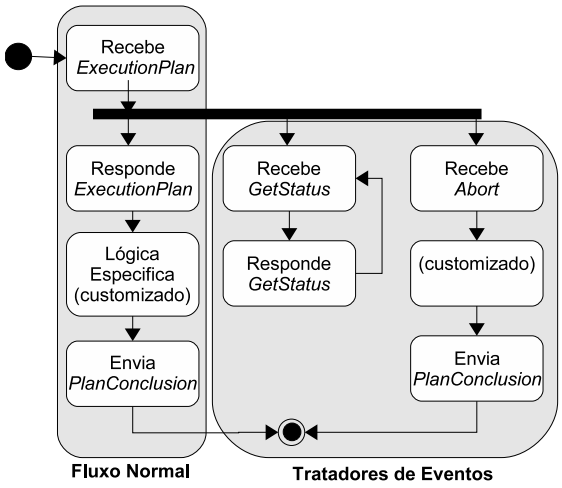

Figura 5.2: Estrutura básica para planos de coordenação.

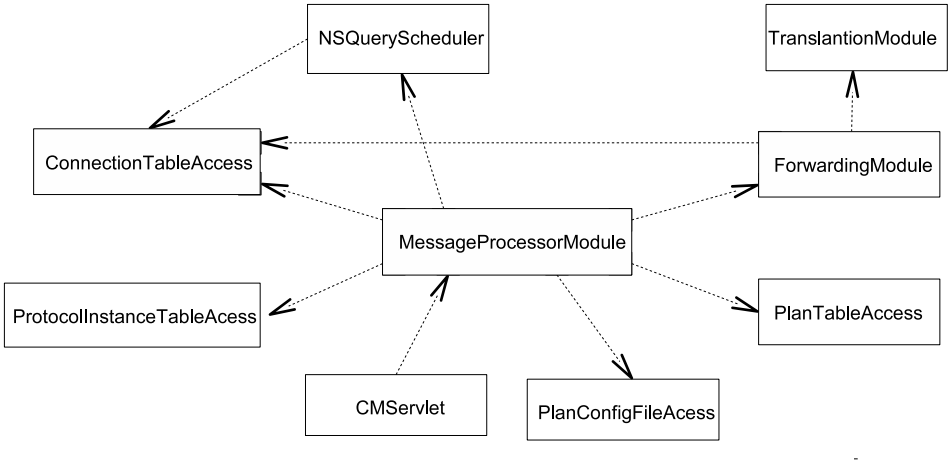

Figura 5.3: Diagrama de classes para o módulo gerente.

processadas por uma instância da classe `MessageProcessorModule`.

MessageProcessorModule Esta é a classe que efetivamente processa as mensagens SOAP recebidas. Dependendo do tipo da mensagem, estabelece e finaliza conexões com outros gerentes (utilizando as classes `ConnectionTableAccess`, `ProtocolInstaceTableAccess` e `NSQueryScheduler`), inicia planos de coordenação (utilizando as classes `PlanTableAccess` e `PlanConfigFileAccess`) e encaminha mensagens de aplicação (por meio da classe `ForwardingModule`).

ConnectionTableAccess O módulo gerente mantém em seu banco de dados uma tabela que armazena dados das conexões entre o gerente de coordenação e outros gerentes. O código SQL para criação desta tabela é apresentado no Código Fonte 5.1. Cada conexão é identificada pela combinação do identificador do processo de negócios (atributo `coordinationProtocol`), do identificador do contexto do processo (atributo `coordinationContext`), do identificador do papel que está sendo executado pelo gerente de coordenação (atributo `roleId`) e do identificador do papel do parceiro (atributo `partnerId`). Pelo registro da conexão, é possível recuperar o identificador do plano de coordenação que contempla o papel executado (atributo `planID`), o *endpoint* do parceiro (`partnerEPR`) e o estado da conexão (atributo `status`). A classe `ConnectionTableAccess` provê métodos para gerência da tabela de conexões.

Código Fonte 5.1: Código SQL referente à tabela de conexões.

```
1  create table connection (
2          coordinationProtocol varchar(40),
3          coordinationContext varchar(40),
4          roleId varchar(40),
5          planId varchar(40),
6          partnerId varchar(40),
7          partnerEPR text,
8          status varchar(15),
9          primary key (coordinationProtocol, coordinationContext,
               roleId, partnerId)
10 );
```

NSQueryScheduler Esta classe consulta o repositório de participantes a fim de obter o *endpoint* do gerente de coordenação que desempenha um papel específico. Caso tal gerente ainda não tenha se registrado no repositório, a instância da classe `NSQueryScheduler`

passa a realizar consultas periódicas. Após obter o *endpoint* do gerente, a instância da classe `NSQueryScheduler` atualiza a tabela de conexões.

ForwardingModule Classe responsável pelo encaminhamento das mensagens de aplicação emitidas pela máquina de execução para os gerentes de coordenação destino e pela entrega das mensagens recebidas para as instâncias corretas dos planos de coordenação.

TranslationModule A máquina de execução utilizada na implementação do gerente de coordenação não permite a utilização do cabeçalho das mensagens de aplicação. A função da classe `TranslationModule` é transformar as mensagens de aplicação recebidas para o formato que pode ser usado pela máquina de execução para controlar o contexto das mensagens. Uma instância desta classe insere os dados originalmente carregados pelo cabeçalho para o corpo das mensagens. Por outro lado, transforma as mensagens emitidas pela máquina de execução para o formato que pode ser processado pelo módulo gerente.

PlanTableAccess O módulo gerente mantém uma tabela de planos de coordenação onde registra todos os planos que o gerente de coordenação é capaz de executar. O Código Fonte 5.2 mostra o código SQL para esta tabela. Cada registro identifica o arquivo de configuração (atributo `planConfigFile`) referente a um plano de coordenação. Este plano deve ser executado para que o gerente de coordenação desempenhe o papel definido pelo atributo `roleID` no contexto do processo de negócio correspondente ao identificador `coordinationProtocol`. A classe `PlanTableAccess` provê métodos para gerência da tabela de planos de coordenação.

Código Fonte 5.2: Código SQL referente à tabela de planos de coordenação.

```
1 create table plan (
2         coordinationProtocol varchar(40),
3         roleId varchar(40),
4         planConfigFile varchar(100),
5         primary key (coordinationProtocol, roleId)
6 );
```

PlanConfigFileAccess Esta classe fornece métodos para acessar um arquivo de configuração e extrair as informações necessárias para a execução do plano de coordenação correspondente.

ProtocolInstanceTableAccess O gerente de coordenação mantém uma tabela de instâncias de processos de negócio que armazena o *endpoint* do repositório de participantes que está sendo usado em cada instância de processo. O Código Fonte 5.3 ilustra o código SQL para geração desta tabela. Os atributos `coordinationProtocol` e `coordinationContext` identificam a instância do processo. O atributo `PREpr` armazena o *endpoint* do repositório de participantes. A classe `ProtocolInstanceTableAccess` provê métodos para gerência desta tabela.

Código Fonte 5.3: Código SQL referente à tabela de instâncias de processos de negócio.

```
1 create table protocolInstance (
2         coordinationProtocol varchar(40),
3         coordinationContext varchar(40),
4         PREpr text,
5         primary key (coordinationProtocol, coordinationContext)
6 );
```

5.2 Protótipo do Repositório de Participantes

O protótipo do repositório de participantes não foi implementado conforme a especificação apresentada na Seção 4.2. O objetivo foi ter um repositório simples que permitisse testar o gerente de coordenação e por isso o repositório de participantes foi implementado por um serviço Web que contempla o registro e a busca de *endpoints* dos gerentes de coordenação. A interface deste serviço Web é apresentado pela Tabela 5.1.

NameServerIF			
Operação	Descrição	Parâmetros Específicos	Parâmetros Comuns
`Registration`	Registra um gerente de coordenação como o participante que desempenha o papel especificado como parâmetro.	**EPR**: endpoint do gerente de coordenação. **AuthKey**: chave de autenticação.	**BusinessProcessID**: Identificador do processo de negócios. Distingue unicamente um processo de negócios.
`Unregistration`	Remove um registro.	**AuthKey**: chave de autenticação.	**ContextID**: Identificador de controle de contexto. Distingue unicamente uma instância de um processo de negócios. **RoleID**: Identificador do papel.
`Query`	Retorna o endpoint de um gerente registrado, de acordo com o papel especificado como parâmetro.		

Tabela 5.1: Interface do protótipo do repositório de participantes.

Este serviço Web foi implementado utilizando a linguagem Java e a API JAXM. A Figura 5.4 mostra o diagrama de classes correspondente. Estas classes são descritas a seguir.[3]

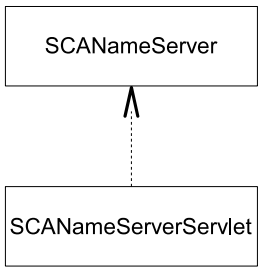

Figura 5.4: Diagrama de classes para o repositório de participantes.

SCANameServerServlet Representa a interface de serviço Web. Permite receber e responder mensagens SOAP sobre o protocolo HTTP. As mensagens recebidas são processadas por uma instância da classe `SCANameServer`.

SCANameServer Implementa a interface *NameServerIF* (Tabela 5.1). A persistência dos dados é realizada por meio de um banco de dados Postgres.

5.3 Protótipo do Gerador de Esqueletos

O protótipo do gerador de esqueletos implementa parcialmente as funcionalidades descritas na Seção 4.3. A geração automática de esqueletos foi contemplada, porém a geração de documentos WSDL correspondentes foi apenas parcialmente implementada. O protótipo gera apenas a estrutura inicial do documento WSDL, exigindo a adição manual de código WSDL.

A linguagem utilizada na implementação do protótipo foi Java. Foram utilizadas as APIs DOM (`org.w3c.dom`) e XPath (`javax.xml.xpath.XPath`) para tratamento de XML. A Figura 5.5 mostra as principais classes do protótipo. Os atributos e métodos foram suprimidos para facilitar o entendimento. As classes do diagrama são descritas a seguir.[4]

WSCDLTranslator Implementa as regras de mapeamento descritas na Seção 4.3.1. Esta classe é responsável pela tradução de coreografias WS-CDL para os esqueletos de planos de coordenação em BPEL.

[3] As interfaces destas classes são apresentadas no Apêndice A.
[4] As interfaces destas classes são apresentadas no Apêndice A.

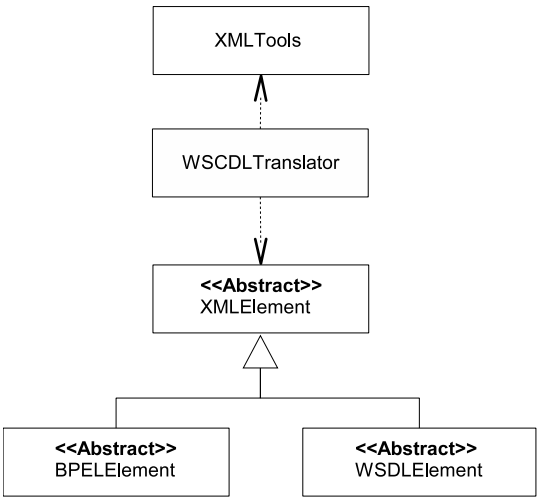

Figura 5.5: Diagrama de classes para o protótipo do gerador de esqueletos.

XMLTools Implementa diversos métodos que auxiliam no processamento de documentos XML.

XMLElement Implementa métodos básicos para criação de documentos XML.

BPELElement Classe abstrata que apresenta 28 especializações que representam elementos da linguagem BPEL. Estas especializações, que foram omitidas no diagrama (Figura 5.5), são usadas na geração do código dos esqueletos de planos de coordenação.

WSDLElement Classe abstrata que apresenta 12 especializações que representam elementos da linguagem WSDL. Estas especializações, que foram omitidas no diagrama (Figura 5.5), são na geração dos documentos WSDL associados aos esqueletos de planos de coordenação.

Capítulo 6

Trabalhos Relacionados

Este capítulo apresenta alguns trabalhos que enfocam questões relacionadas ao trabalho realizado. Estas questões foram agrupadas como: modelagem de cadeias produtivas (Seção 6.1), gerência de cadeias produtivas (Seção 6.2), integração de tecnologias para serviços Web (Seção 6.3) e execução de orquestrações e coreografias (Seção 6.4). Os pontos relevantes de cada trabalho são comparados ao trabalho realizado.

6.1 Modelagem de Cadeias Produtivas

Os trabalhos [1] e [2] tratam o problema da modelagem e simulação de cadeias produtivas. Rossetti e Chan [1] introduzem o SCSF (*Suppy Chain Simulation Framework*), um arcabouço que utiliza um modelo de objetos para representar os participantes da cadeia produtiva e seus relacionamentos. Este modelo é representado por diagramas UML (*Unified Modelling Language*) e pode ser mapeado para classes Java, permitindo a execução de simulações. A capacidade de simulação do arcabouço é provida pela biblioteca JSL (*Java Simulation Library*).

Chatfield et al. [2] propõem uma linguagem XML específica para modelagem de cadeias produtivas. Esta linguagem, chamada de SCML (*Supply Chain Modelling Language*), permite descrever informações estruturais e lógicas da cadeia. Os autores também apresentam o protótiopo de um sistema que mapeia a SCML para um modelo de simulação executável.

Apesar do modelo de coordenação proposto não ter como objetivo a simulação de cadeias produtivas, ele pode ser comparado com [1] e [2] em relação à modelagem das cadeias. Enquanto [1] e [2] utilizam UML e XML (respectivamente) para modelar tanto a estrutura quanto a dinâmica das cadeias produtivas, a solução proposta modela somente a parte dinâmica. A parte estrutural é modelada utilizando-se a arquitetura de Bacarin (Seção 2.3.2). A Tabela 6.1 sumariza a comparação entre [1], [2] e o modelo de coordenação

proposto.

Tabela 6.1: Comparação da infra-estrutura proposta com as soluções apresentadas por [1] e [2].

	Rosseti e Chan [1]	**Chatfield et al.[2]**	**Modelo de coordenação Proposto**
Forma de modelagem	UML	XML	Modelo de Bacarin + Coreografias de serv. Web

6.2 Gerência de Cadeias Produtivas

O problema da gerência de cadeias produtivas é abordado em [3], [4] e [5]. Yang et al. [3] propõem um modelo para gerência de cadeias produtivas (ASCM - *Agile Supply Chain Management*) que separa a lógica de negócios do processo de transações. Neste modelo, um sistema de agentes múltiplos (*multi-agent system*) é implementado para processar as transações de negócio enquanto um sistema baseado em regras é projetado para controlar o *workflow* de negócios e a cooperação entre os agentes. Se as regras do sistema são alteradas, o comportamento dos agentes também é alterado, não sendo necessário modificar suas implementações.

O trabalho de Hassan e Soh [4] também se baseia em uma arquitetura de agentes de *software* para gerência de cadeias produtivas. Nesta arquitetura, os agentes são encapsulados por serviços Web, buscando promover maior interoperabilidade entre os participantes da cadeia.

Huhns e Stephens [5] apresentam uma metodologia, baseada em agentes de *software*, para automação de cadeias produtivas. Na primeira etapa da metodologia, diagramas de interação UML são utilizados para representar as interações de negócio das entidades participantes. Em seguida, as mensagens do diagrama de interação são convertidas em grafos que representam a conversação de cada participante. A partir dos grafos de conversação, são geradas máquinas de estados que descrevem o comportamento dos agentes.

A Tabela 6.2 mostra uma comparação entre a infra-estrutura proposta e as soluções apresentados por [3], [4] e [5]. Todos os trabalhos comparados utilizam uma abordagem baseada em agentes de *software*. O uso de agentes de *software* fornece maior autonomia ao sistema de gerência da cadeia, porém, nada impede que agentes de *software* sejam encapsulados em serviços Web e utilizados na lógica dos planos de coordenação. A separação entre o ambiente de execução e a lógica de negócios é importante para agilizar a adaptação da cadeia produtiva às suas mudanças. Esta característica é claramente presente em [3], porém não foi apresentada pelos outros autores. Diferente dos outros

trabalhos, [4] apresenta a preocupação com a interoperabilidade entre os participantes da cadeia. Assim como a infra-estrutura proposta, [4] propõe uma abordagem que utiliza serviços Web para comunicação entre os participantes da cadeia. A metodologia que utiliza UML para modelar o comportamento dos agentes de *software* proposta por [5] facilita o projeto da lógica de negócios da cadeia, e pode ser comparada à metodologia de criação de planos de coordenação da infra-estrutura proposta. A respeito da ligação dinâmica entre parceiros de negócio, nenhum dos trabalhos comparados apresenta esta característica.

Tabela 6.2: Comparação da infra-estrutura proposta com as soluções apresentadas por [3], [4] e [5].

	Yang et al. [3]	**Hassan e Soh [4]**	**Huhns e Stephens[5]**	**Infra-estrutura Proposta**
Abordagem utilizada	Agentes de soft.	Agentes de soft. + Serv. Web	Agentes de soft.	Coreografia de Serv. Web
Separação entre ambiente e lógica de execução	Sim	Não	Não	Sim
Facilidades de interoperabilidade	Não	Serviços Web	Não	Serviços Web
Facilidades para projeto da lógica de negócios	Não	Não	Utilização de UML	Utilização de WS-CDL
Ligação dinâmica com parceiros	Não	Não	Não	Sim

6.3 Integração de Tecnologias para Serviços Web

Uma das características do modelo de coordenação proposto é a integração de tecnologias que correspondem a diferentes camadas da pilha de protocolos de serviços Web. A Figura 6.1 mostra a equivalência entre as camadas do modelo e desta pilha.

Exemplos de trabalhos relacionados à integração de tecnologias da pilha de protocolos de serviços Web são [6], [7] e [8]. Curbera et al. [6] propõem o uso combinado de BPEL (camada de orequestração) com WS-Coordination (camada de qualidade de serviço). Nesta combinação, WS-Coordination coordena o tratamento de faltas e de atividades de compensação de diversas implementações BPEL e define o identificador de contexto para a interação entre os processos.

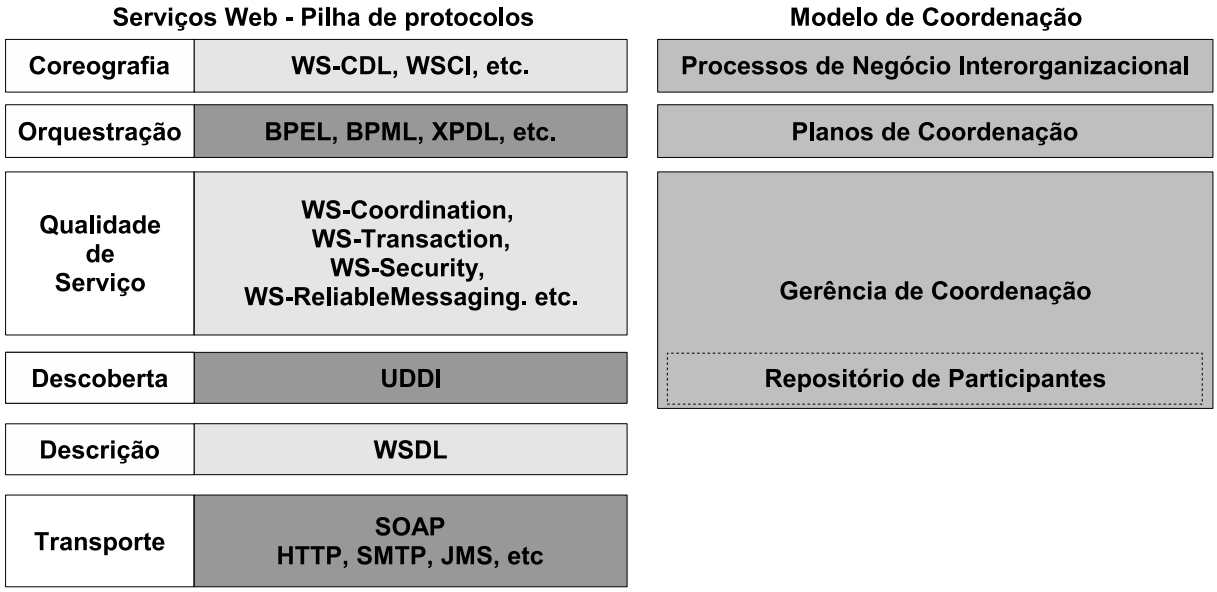

Figura 6.1: Pilha de protocolos de serviços Web X Modelo de Coordenação.

Tanto Mendling e Hafner [7] quanto Diaz et al. [8] apresentam regras para derivação de esqueletos BPEL a partir de coreografias WS-CDL. A abordagem de [7] é o mapeamento direto entre as duas linguagens. Os autores apresentam o relacionamento entre os elementos das duas linguagens e argumentam a respeito das limitações do mapeamento. Diaz et al. [8] propõem um mecanismo que usa autômatos temporais como representação intermediária entre WS-CDL e BPEL. Esta abordagem permite a verificação e validação automática dos processos gerados.

A Tabela 6.3 mostra uma comparação das camadas da pilha de protocolos tratadas por [6], [7], [8] e pelo modelo de coordenação proposto. Apesar do modelo de coordenação não utilizar uma especificação conhecida na camada de qualidade de serviço, a gerência de coordenação desempenha uma função semelhante a WS-Coordination, com relação à conexão entre gerentes de coordenação e troca de identificadores de contexto. Além disso, o gerente de coordenação pode ser estendido a fim de contemplar outros requisitos de qualidade de serviço, como autenticação, autorização e segurança. Conforme pode-se perceber, WS-CDL e BPEL são as únicas especificações abordadas nas camadas de coreografia e orquestração, respectivamente.

6.4 Execução de Coreografias e Orquestrações

Os trabalhos apresentados por [9], [10] e [11] apresentam infra-estruturas para orquestração e coreografia de serviços Web. Chung et al. [9] apresentam uma arquitetura para

Tabela 6.3: Comparação da infra-estrutura proposta com as soluções apresentadas por [6], [7] e [8].

	Curbera et al [6]	**Mendling e Hafner [7]**	**Diaz et al. [8]**	**Modelo de coord. proposto**
Coreografia	—	WS-CDL	WS-CDL	WS-CDL
Orquestração	BPEL	BPEL	BPEL	BPEL
Qualidade de serviço	WS-Coordination	—	—	Gerência de coordenação

gerência de processos de negócio baseada em orquestração de serviços Web. Esta arquitetura, chamada WSCPC (*Web Service oriented Collaborative Product Commerce*), utiliza uma gramática para representar os processos. Esta gramática permite especificar as dependências entre tarefas e suas entradas e saídas. A arquitetura provê recursos para procurar servidores de serviços, atribuir-lhes tarefas e coordená-los. O sistema utiliza semântica para facilitar a descoberta de serviços que contemplam as tarefas do processo.

Caituiro-Monge e Rodríguez-Martinez [10] introduzem um arcabouço para coreografia de serviços Web, voltado para *e-Government*. A abordagem deste arcabouço inclui dados de controle em um documento XML anexado a mensagens que carregam resultados parciais ou requisições para serviços. Este documento XML indica o próximo serviço Web a ser invocado, o destino dos resultados parciais e a forma de como estes devem ser processados. Coreografia é alcançada pelo estabelecimento de federações *ad-hoc*, uma coleção de serviços Web que colaboram temporariamente a fim de responder à requisição de um usuário.

Jung et al. [11] propõem uma metodologia para coreografia de processos de negócio que incorpora *workflows* internos de uma organização em processos de negócio colaborativos interorganizacionais. Os autores também apresentam uma arquitetura para execução destes processos que utiliza a lógica de negócios externa representada em BPML.

A Tabela 6.4 mostra a comparação entre estes trabalhos e a infra-estrutura proposta. Apesar de Jung et al. mencionarem a necessidade de um "contrato de interoperabilidade" entre os participantes da coreografia, nenhum dos trabalhos comparados apresentou uma forma de representação global da coreografia, como WS-CDL. A arquitetura de Chung et al. fornece suporte à descoberta dinâmica de serviços. Esta característica foi mencionada por Caituiro-Monge e Rodríguez-Martinez, porém a solução não foi apresentada. A respeito da interação com o usuário, somente o trabalho de Caituiro-Monge e Rodríguez-Martinez não apresentou um mecanismo para este fim.

Tabela 6.4: Comparação da infra-estrutura proposta com as soluções apresentadas por [9], [10] e [11].

	Chung et al. [9]	**Caituiro e Rodríguez [10]**	**Jung et al. [11]**	**Infra-estrutura Proposta**
Representação da coreografia	Não	Não	Não	WS-CDL
Comportamento Individual	Gramática de Processo	Estrutura XML nas mensagens	BPML	Plano de coord. (BPEL)
Descoberta autom. de parceiros	Sim	Não	Não	Sim
Suporte p/ interação com usuário	Sim	Não	Sim	Sim

Capítulo 7

Conclusão

A integração dos participantes de uma cadeia produtiva visa otimizar o desempenho coletivo na criação, distribuição e suporte ao produto final, apresentando reflexos importantes na competitividade comercial dos envolvidos. Este trabalho foi motivado pela necessidade de mecanismos para coordenação das atividades que compõem os processos de negócio interorganizacionais das cadeias produtivas.

A solução apresentada foi uma infra-estrutura baseada em coreografias de serviços Web. A tecnologia de serviços Web proporciona a interoperabilidade necessária para contornar a dificuldade de integração causada pela natureza distribuída, autônoma e heterogênea das cadeias. Uma das contribuições desta infra-estrutura é a possibilidade de executar processos de negócio interorganizacionais definidos por coreografias WS-CDL como um conjunto de planos de coordenação descritos em BPEL.

Outra contribuição é o modelo de coordenação (Seção 3) dividido em três camadas que, combinadas, favorecem o projeto, a implantação e a execução de processos de negócio interorganizacionais de forma eficiente e consistente. Este modelo provê flexibilidade para implantação dos processos, o que é uma característica importante, dada a dinamicidade das cadeias.

Na primeira camada do modelo, a utilização de descrições de coreografias WS-CDL auxilia na geração consistente de planos de coordenação (segunda camada), definindo de maneira clara e formal as interações entre os participantes da cadeia produtiva, as restrições sobre as quais essas interações acontecem e as informações trocadas durante a colaboração. O gerador de esqueletos de planos de coordenação (Seção 4.3) possibilita a derivação automática dos esqueletos, o que aumenta a rapidez do processo de geração dos planos de coordenação e minimiza o risco de inconsistências. O fato dos planos de coordenação serem descritos em BPEL proporciona facilidades para a integração de aplicações e *workflows* internos. Tais aplicações e *workflows* podem ser encapsulados por serviços Web e suas invocações adicionadas aos planos de coordenação.

A lógica de coordenação, implementada na terceira camada pelo gerente de coordenação (4.1), facilita o projeto dos processos de negócio. A ligação entre participantes da cadeia produtiva é transparente para o projetista. As interações do processo de negócio podem ser descritas no nível dos papéis a serem desempenhados, sem a preocupação com a lógica necessária para atribuição de *enpoints* a esses papéis. O uso de dados padronizados para o controle de contexto também facilita o trabalho do projetista, não sendo necessária a definição de campos específicos de cada mensagem para este fim.

A especificação do repositório de participantes (4.2) é considerada outra contribuição do trabalho. Do conhecimento dos autores, a UDDI nunca foi utilizada como na especificação deste repositório. Ao invés de utilizá-la para a simples descoberta de serviços, o repositório de participantes utiliza a UDDI para possibilitar a descoberta de instâncias de planos de coordenação. Esta característica permite a ligação dinâmica de parceiros de negócio. Além disso, a abstração de papéis fornecida pela linguagem WS-CDL foi utilizada em conjunto com o mecanismo de categorização da UDDI, possibilitando a descoberta automática de parceiros de negócio que contemplam a execução de um processo de negócio específico.

O protótipo apresentado na Capítulo 5 não abrangiu todas as funcionalidades da infra-estrutura proposta, porém comprova a sua viabilidade. O uso da máquina de execução *ActiveBPEL* se mostrou viável nos testes iniciais do protótipo, porém apresentou problemas de validação de esquemas XML em testes com controle de contexto, não sendo recomendada para futuras implementações.

Uma possível extensão do trabalho é a especificação de mecanismos de autenticação, autorização e segurança para o modelo de coordenação e a integração destes mecanismos à arquitetura do gerente de coordenação. Outro trabalho futuro é a aplicação de semântica para busca de parceiros de negócios no repositório de participantes e de descrições de coreografias nos repositórios de coreografias. A metodologia de geração de planos de coordenação seria mais amigável ao usuário se fosse definida uma representação visual do processo de negócio interorganizacional, como diagramas de atividades da UML, que fosse automaticamente mapeável para "esqueletos" de coreografias WS-CDL. Outro trabalho futuro seria estudar a possibilidade de estender a infra-estrutura proposta a fim de desenvolver uma solução para aplicações que envolvem o uso de coreografias de serviços Web, além do escopo da arquitetura para integração de cadeias produtivas.

Referências Bibliográficas

[1] M. D. Rossetti e H. T. Chan. A Prototype Object-Oriented Supply Chain Simulation Framework. In *Simulation Conference, 2003. Proceedings of the 2003 Winter*, volume 2, pp. 1612–1620, Dezembro 2003.

[2] D. C Chatfield, T. P. Harrison, e J. C. Hayya. XML-based Supply Chain Simulation Modeling. In *Simulation Conference, 2004. Proceedings of the 2004 Winter*, volume 2, pp. 1485–1493, Dezembro 2004.

[3] T. Yang, C. Lu, e W. Li. Study on Agile Supply Chain Management. In *2004 IEEE International Conference on Systems, Man and Cybernetics*, pp. 6031–6035, 2004.

[4] Umair Hassan e Ben Soh. Web Service Intelligent Agent Structuring for Supply Chain Management (SCM). In *EEE'05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*, pp. 329–332, Washington, DC, Estados Unidos, 2005. IEEE Computer Society.

[5] Michael N. Huhns e Larry M. Stephens. Automating Supply Chains. *IEEE Internet Computing*, 5(4):90–93, 2001.

[6] F. Curbera et al. The next step in web services. *Commun. ACM*, 46(10):29–34, 2003.

[7] J. Mendling e M. Hafner. From inter-organizational workflows to process execution: Generating BPEL from WS-CDL. *Proceedings of OTM 2005 Workshops. Lecture Notes in Computer Science*, 3762:506–515, 2005.

[8] G. Diaz, V. Valero M. E. Cambronero, J. J. Pardo, e F. Cuartero. Automatic generation of correct web services choreographies and orchestrations with model checking techniques. In *Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, pp. 186–192, 2006.

[9] M. J. Chung et al. A framework for collaborative product commerce using web services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pp. 52–60, 2004.

[10] Hillary Caituiro-Monge e Manuel Rodríguez-Martinez. Net Traveler: A Framework for Autonomic Web Services Collaboration, Orchestration and Choreography in E-Government Information Systems. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pp. 2–10, Washington, DC, Estados Unidos, 2004. IEEE Computer Society.

[11] Jae-Yoon Jung, Wonchang Hur, Suk-Ho Kang, e Hoontae Kim. Business Process Choreography for B2B Collaboration. *IEEE Internet Computing*, 8(1):37–45, 2004.

[12] C. Ferris e J. Farrell. What are web services? *Commun. ACM*, 46(6):31, 2003.

[13] B. M. Beamon. Supply chain design and analysis: Models and methods. *International Journal of Production Economics*, 55:281–294, 1998.

[14] E. Bacarin, C.B. Medeiros, e E.R.M. Madeira. A Collaborative Model for Agricultural Supply Chains. In R. Meersman and Z. Tari, editors, *CoopIS/DOA/ODBASE 2004, LNCS 3290*, pp. 319–336. Springer-Verlag, 2004.

[15] R. R. Lummus e R. J. Vokurka. Defining supply chain management: a historical perspective and practical guidelines. *Industrial Management e Data Systems*, 99(1):11–17, 1999.

[16] National Research Concil. *Surviving Supply Chain Integration - Strategies for Small Manufactues*. National Academy Press, Washington, DC - Estados Unidos, 2000.

[17] C. Peltz. Web services orchestration and choreography. *Computer*, 36:46–52, 2003.

[18] G. Alonso et al. *Web Services: Concepsts Architectures and Applications*. Springer, Germany, 2004.

[19] W3C. Web Services Architecture Requirements, 2004. Disponível em: <http://www.w3.org/TR/wsa-reqs/>. Acesso em: 15/10/2006.

[20] W3C. Extensible markup language (xml) 1.0, 2004. Disponível em: <http://www.w3.org/TR/REC-xml/>. Acesso em: 15/10/2006.

[21] F. Curbera et al. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *Internet Computing, IEEE*, 6:86–93, 2002.

[22] W3C. Soap version 1.2 part 1: Messaging framework, 2003. Disponível em:<http://www.w3.org/TR/soap12-part1>. Acesso em: 15/10/2006.

[23] W3C. Web services description language (wsdl) 1.1, 2004. Disponível em: <http://www.w3.org/TR/wsdl>. Acesso em: 15/10/2006.

[24] OASIS. Uddi version 3.0.2, 2004. Disponível em: <http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm>. Acessado em 25/01/2007.

[25] W3C. Xml schema part 0: Primer second edition, Outubro 2004. Disponível em: <http://www.w3.org/TR/xmlschema-0/>. Acesso em 20/10/2006.

[26] W. V. Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Inteligent Systems*, pp. 72–76, 2003.

[27] W3C. Web Services Choreography Description Language version 1.0, November 2005. Disponível em: <http://www.w3c.org/TR/ws-cdl-10>. Acesso em 07/07/2006.

[28] W3C. Web service choreography interface (wsci) 1.0, Agosto 2002. Disponível em: <http://www.w3.org/TR/wsci/>. Acesso em: 25/01/2007.

[29] T. Andrews et al. Business Process Execution Language for Web Services - version 1.1, May 2003. Disponível em: <http://www-128.ibm.com/developerworks/library/ws-bpel/>. Acesso em: 07/07/2006.

[30] A. Arkin et al. Business process modeling language, Novembro 2002. Disponível em: <http://www.bpmi.org/bpml-spec.htm/>. Acesso em: 11 de agosto de 2004.

[31] WFMC. Process definition interface - xml process definition language, Outubro 2005. Disponível em: <http://www.wfmc.org/standards/docs/TC-1025_xpdl_2_2005-10-03.pdf>. Acesso em: 25/01/2007.

[32] P. Kaminsky D. Simchi-Levi e E. Simchi-Levi. *Designing and managing the supply chain : concepts, strategies, and case studies.* Irwin/McGraw-Hill, 2000.

[33] A. A. Kondo et al. Web services-based traceability in food supply chains. In *Proceedings of the 3rd International Conference on Web Information Systems and Technologies*, 2007.

[34] L. F. Cabrera et al. Web services coordination (ws-coordination), Agosto 2005. Disponível em:<ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf>. Acesso em: 04/02/2006.

[35] W3C. Xml path language (xpath) 2.0, Novembro 1999. Disponível em: <http://www.w3.org/TR/xpath>. Acesso em: 10/11/2006.

Apêndice A

Lista de Interfaces do Protótipo

Este apêndice apresenta as interfaces públicas das classes do protótipo (Capítulo 5).

A.1 Gerente de Coordenação

CMServlet		
Operação	**Descrição**	**Parâmetros individuais**
`onMessage`	Recebe uma mensagem SOAP via HTTP, repassa para uma instância da classe MessageProcessorModule e devolve a resposta fornecida	**Message**: mensagem SOAP, representada como uma classe javax.xml.soap.SOAPMessage

MessageProcessorModule		
Operação	**Descrição**	**Parâmetros**
`processMessage`	Avalia e processa uma mensagem recebida pelo CMServlet	**msg**: mensagem SOAP a ser processada, representada como uma classe javax.xml.soap.SOAP-Message

ConnectionTableAccess			
Operação	**Descrição**	**Parâmetros individuais**	**Parâmetros comuns**
`insertConnection`	Insere uma nova entrada na tabela de conexões	**pid**: identificador do plano de coordenação referente ao papel executado **pepr**: endpoint do parceiro **status**: estado da conexão	**cp**: identificador do processo de negócio **cx**: identificador de contexto **ri**: identificador do papel que está sendo executado pelo gerente de coordenação **pi**: identificador do papel do parceiro
`deleteConnection`	Remove uma entrada da tabela de conexões		
`getStatus`	Retorna o estado da conexão		
`getPartnerEPR`	Retorna o *endpoint* do parceiro		
`updateStatus`	Atualiza o estado de uma conexão	**status**: estado da conexão	
`getPlanId`	retorna o identificador do plano de coordenação correspondente ao papel que está sendo executado em uma conexão		

NSQueryScheduler		
Operação	**Descrição**	**Parâmetros individuais**
`start`	Inicia a consulta ao repositório de participantes	**cp**: identificador do processo de negócio **cx**: identificador de contexto **ri**: identificador do papel que está sendo executado pelo gerente de coordenação **pi**: identificador do papel do parceiro
`getEPR`	Retorna o *endpoint* obtido.	

ForwardingModule			
Operação	**Descrição**	**Parâmetros individuais**	**Parâmetros comuns**
`forwardAppMeg`	Entrega uma mensagem de aplicação para a instância do plano de coordenação a que se destina		**Message**: mensagem SOAP, representada como uma classe javax.xml.soap.SOAPMessage
`forwardBpelMesg`	Encaminha uma mensagem de aplicação emitida pela máquina de execução para o gerente de coordenação destino		
`forwardForWS`	Encaminha uma mensagem de aplicação emitida pela máquina de execução para um serviço Web com *endpoint* arbitrário	**Epr**: *endpoint* do serviço Web	

TranslationModule		
Operação	**Descrição**	**Parâmetros comuns**
`app2bpel`	Transforma uma mensagem do formato utilizado pelo modelo de coordenação para o formato utilizado pela máquina de execução	**Message**: mensagem SOAP a ser transformada, representada como uma classe javax.xml.soap.SOAPMessage
`bpel2app`	transforma uma mensagem do formato utilizado pela máquina de execução para o formato utilizado pelo modelo de coordenação	

PlanTableAccess			
Operação	**Descrição**	**Parâmetros individuais**	**Parâmetros comuns**
`insertPlan`	Insere um novo registro na tabela de planos	**pconf**: localização do arquivo de configuração correspondete ao plano	**cp**: identificador do processo de negócio
`deletePlan`	Remove um registro da tabela de planos		**ri**: identificador do papel que está sendo executado pelo gerente de coordenação
`isValidPlan`	Verifica se um papel é suportado pelo gerente de coordenação, ou seja, se ele está apto a executar o plano de coordenação correspondente		
`getPCFilePath`	retorna a localização do arquivo de configuração		

PlanConfigFileAccess		
Operação	**Descrição**	**Parâmetros individuais**
`getPlanId`	Retorna o identificador do plano de coordenação correspondente ao arquivo de configuração correspondete ao objeto	
`getEndpoint`	retorna o *endpoint* de um parceiro, desde que seja estático	**role**: papel do parceiro
`getEndpointType`	Retorna a forma de conexão com um parceiro	**role**: papel do parceiro
`getPartnerType`	Retorna o tipo de um parceiro	**role**: papel do parceiro

ProtocolInstanceTableAccess			
Operação	**Descrição**	**Parâmetros individuais**	**Parâmetros comuns**
`instantiate`	Insere um registro na tabela de instâncias de processos	**PREpr**: *endpoint* do repositório de participantes	**cp**: identificador do processo de negócio
`deleteInstance`	Remove um registro da tabela de instâncias de processos		**cx**: identificador de contexto
`isInstantieted`	verifica se uma instância especifica de um processo de negócios está ativa		
`getPREpr`	Retorna o *endpoint* do repositório de participantes		

A.2 Repositório de Participantes

SCANameServerServlet		
Operação	**Descrição**	**Parâmetros individuais**
`onMessage`	Recebe uma mensagem SOAP via HTTP, repassa para uma instância da classe ParticipanteRepository e devolve a resposta fornecida	**Message**: mensagem SOAP, representada como uma classe javax.xml.soap.SOAPMessage

SCANameServer		
Operação	**Descrição**	**Parâmetros individuais**
`processMessage`	Avalia e processa uma mensagem recebida pelo PRServlet	**msg**: mensagem SOAP a ser processada, representada como uma classe javax.xml.soap.SOAP-Message

A.3 Gerador de Esqueletos

WSCDLTranslator		
Operação	**Descrição**	**Parâmetros individuais**
`generateBpel`	Gera o esqueleto de plano de coordenação para o papel especificado como parâmetro	**roleType**: identificador do papel para o qual será gerado o esqueleto de plano de coordenação **fineName**: localização do arquivo que contém a descrição da coreografia

XMLTools		
Operação	**Descrição**	**Parâmetros individuais**
`stringToElement`	Gera uma árvore de objetos DOM a partir de um documento XML contido em uma *string*	**str**: *string* que contém o documento XML
`elementToString`	Gera uma *string* contendo o documento XML representado por uma árvore de objetos DOM	**ele**: elemento raiz da árvore de objetos DOM
`fileToDocument`	Gera uma árvore de objetos DOM a partir de um documento XML armazenado em um arquivo	**file**: localização do arquivo que contém o documento XML
`getNodeList`	Retorna uma lista de nós que contemplem uma expressão XPath	**node**: objeto DOM a partir do qual será feita a busca **expression**: expressão XPath
`getNode`	retorna um nó que contemple uma expressão XPath	**node**: objeto DOM a partir do qual será feita a busca **expression**: expressão XPath
`getAttribute`	retorna o valor de um atributo que contemple uma expressão XPath	**node**: objeto DOM a partir do qual será feita a busca **expression**: expressão XPath

XMLElement		
Operação	**Descrição**	**Parâmetros individuais**
`writeTo`	Método abstrato que escreve o código referente ao elemento XML e aos seus filhos e irmãos, em um *buffer*	**doc**: *buffer* onde o código referente ao elemento XML deve ser escrito
`getFirstChild`	Retorna o primeiro elemento filho do elemento XML	
`getFirstSibling`	Retorna o primeiro elemento irmão do elemento XML	
`insertChild`	Insere um elemento filho em um elemento XML	**element**: elementoque deseja-se inserir
`insertSibling`	Insere um elemento irmão em um elemento XML	**element**: elemento que deseja-se inserir
`insertNameSpace`	Insere uma declaração de *namespace* em um elemento XML	**ns**: *namespace* a ser inserido